django-fobi Documentation

Release 0.10.7

Artur Barseghyan <artur.barseghyan@gmail.com>

May 08, 2017

Contents

django-fobi (or just *fobi*) is a customisable, modular, user- and developer- friendly form generator/builder application for Django. With *fobi* you can build Django forms using an intuitive GUI, save or mail posted form data or even export forms into JSON format and import them on other instances. API allows you to build your own form elements and form handlers (mechanisms for handling the submitted form data).

CHAPTER 1

Prerequisites

Present

- Django 1.8, 1.9, 1.10 and 1.11.
- Python 2.7, 3.4, 3.5, 3.6 and PyPy.

Note, that Django 1.11 is not yet proclaimed to be flawlessly supported. The core and contrib packages have been tested against the alpha Django 1.11a1 PyPI release. All tests have successfully passed, although it's yet too early to claim that Django 1.11 is fully supported. Certain dependencies (`django-formtools` and `easy-thumbnails`) have been installed from source (since versions supporting Django 1.11 are not yet released on PyPI.)

Past

- Dropping support of Django 1.5, 1.6 has been announced in version 0.9.13. Dropping support of Django 1.7 has been announced in version 0.9.17. As of 0.9.17 everything is still backwards compatible with versions 1.5, 1.6 and 1.7, but in future versions compatibility with these versions will be wiped out.
- Dropping support of Python 2.6 has been announced in version 0.9.17. As of 0.9.17 everything is still backwards compatible with Python 2.6, but in future versions compatibility with it will be wiped out.
- Since version 0.10.4 support for Python 3.3 has been dropped.

CHAPTER 2

Key concepts

- Each form consists of elements. Form elements are divided into two groups:
    1. form fields (input field, textarea, hidden field, file field, etc.).
    2. content (presentational) elements (text, image, embed video, etc.).
- Number of form elements is not limited.
- Each form may contain handlers. Handler processes the form data (for example, saves it or mails it). Number of the handlers is not limited.
- Both form elements and form handlers are made with Django permission system in mind.
- As an addition to form handlers, form callbacks are implemented. Form callbacks are fired on various stages of pre- and post-processing the form data (on POST). Form callbacks do not make use of permission system (unless you intentionally do so in the code of your callback) and are fired for all forms (unlike form handlers, that are executed only if assigned).
- Each plugin (form element or form handler) or a callback - is a Django micro-app.

Note, that *django-fobi* does not require django-admin and administrative rights/permissions to access the UI, although almost seamless integration with django-admin is implemented through the `simple` theme.

CHAPTER 3

Main features and highlights

- User-friendly GUI to quickly build forms.
- Large variety of *Bundled form element plugins*. Most of the Django fields are supported. *HTML5 fields* are supported as well.
- *Form wizards*. Combine your forms into wizards. Form wizards may contain handlers. Handler processes the form wizard data (for example, saves it or mails it). Number of the form wizard handlers is not limited.
- Anti-spam solutions like CAPTCHA, ReCAPTCHA or Honeypot come out of the box (CAPTCHA and ReCAPTCHA do require additional third-party apps to be installed).
- In addition to standard form elements, there are cosmetic (presentational) form elements (for adding a piece of text, image or a embed video) alongside standard form elements.
- Data handling in plugins (form handlers). Save the data, mail it to some address or repost it to some other endpoint. See the *Bundled form handler plugins* for more information.
- Developer-friendly API, which allows to edit existing or build new form fields and handlers without touching the core.
- Support for custom user model.
- *Theming*. There are 4 ready to use *Bundled themes*: "Bootstrap 3", "Foundation 5", "Simple" (with editing interface in style of Django admin) and "DjangoCMS admin style" theme (which is another simple theme with editing interface in style of djangocms-admin-style).
- Implemented integration with FeinCMS (in a form of a FeinCMS page widget).
- Implemented integration with DjangoCMS (in a form of a DjangoCMS page plugin).
- Implemented integration with Mezzanine (in a form of a Mezzanine page).
- Reordering of form elements using drag-n-drop.
- Data export (DB store form handler plugin) into XLS/CSV format.
- *Dynamic initial values* for form elements.
- Import/export forms to/from JSON format.
- Import forms from MailChimp using mailchimp importer.

CHAPTER 4

Roadmap

Some of the upcoming/in-development features/improvements are:

- Integration with *django-rest-framework* (in version 0.11).
- Bootstrap 4 and Foundation 6 support (in version 0.12).
- Wagtail integration (in version 0.13).

See the TODOS for the full list of planned-, pending- in-development- or to-be-implemented features.

CHAPTER 5

Some screenshots

See the documentation for some screen shots:

- PythonHosted
- ReadTheDocs

CHAPTER 6

Demo

Live demo

See the live demo app on Heroku.

Credentials:

- username: test_user
- password: test_user

Run demo locally

In order to be able to quickly evaluate the *django-fobi*, a demo app (with a quick installer) has been created (works on Ubuntu/Debian, may work on other Linux systems as well, although not guaranteed). Follow the instructions below for having the demo running within a minute.

Grab the latest *django_fobi_example_app_installer.sh*:

```
wget https://raw.github.com/barseghyanartur/django-fobi/stable/examples/django_fobi_example_app_insta
```

Assign execute rights to the installer and run the *django_fobi_example_app_installer.sh*:

```
chmod +x django_fobi_example_app_installer.sh
./django_fobi_example_app_installer.sh
```

Open your browser and test the app.

Dashboard:

- URL: http://127.0.0.1:8001/fobi/
- Admin username: test_admin
- Admin password: test

Django admin interface:

- URL: http://127.0.0.1:8001/admin/
- Admin username: test_admin
- Admin password: test

If quick installer doesn't work for you, see the manual steps on running the example project.

CHAPTER 7

Quick start

See the quick start.

CHAPTER 8

Installation

1. Install latest stable version from PyPI:

```
pip install django-fobi
```

Or latest stable version from GitHub:

```
pip install https://github.com/barseghyanartur/django-fobi/archive/stable.tar.gz
```

Or latest stable version from BitBucket:

```
pip install https://bitbucket.org/barseghyanartur/django-fobi/get/stable.tar.gz
```

2. Add *fobi* to `INSTALLED_APPS` of the your projects' Django settings. Furthermore, all themes and plugins to be used, shall be added to the `INSTALLED_APPS` as well. Note, that if a plugin has additional dependencies, you should be mentioning those in the `INSTALLED_APPS` as well.

```
INSTALLED_APPS = (
    # Used by fobi
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',

    # ...
    # `django-fobi` core
    'fobi',

    # `django-fobi` themes
    'fobi.contrib.themes.bootstrap3', # Bootstrap 3 theme
    'fobi.contrib.themes.foundation5', # Foundation 5 theme
    'fobi.contrib.themes.simple', # Simple theme

    # `django-fobi` form elements - fields
    'fobi.contrib.plugins.form_elements.fields.boolean',
    'fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple',
    'fobi.contrib.plugins.form_elements.fields.date',
    'fobi.contrib.plugins.form_elements.fields.date_drop_down',
    'fobi.contrib.plugins.form_elements.fields.datetime',
    'fobi.contrib.plugins.form_elements.fields.decimal',
    'fobi.contrib.plugins.form_elements.fields.email',
    'fobi.contrib.plugins.form_elements.fields.file',
```

```
    'fobi.contrib.plugins.form_elements.fields.float',
    'fobi.contrib.plugins.form_elements.fields.hidden',
    'fobi.contrib.plugins.form_elements.fields.input',
    'fobi.contrib.plugins.form_elements.fields.integer',
    'fobi.contrib.plugins.form_elements.fields.ip_address',
    'fobi.contrib.plugins.form_elements.fields.null_boolean',
    'fobi.contrib.plugins.form_elements.fields.password',
    'fobi.contrib.plugins.form_elements.fields.radio',
    'fobi.contrib.plugins.form_elements.fields.regex',
    'fobi.contrib.plugins.form_elements.fields.select',
    'fobi.contrib.plugins.form_elements.fields.select_model_object',
    'fobi.contrib.plugins.form_elements.fields.select_multiple',
    'fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects',
    'fobi.contrib.plugins.form_elements.fields.slug',
    'fobi.contrib.plugins.form_elements.fields.text',
    'fobi.contrib.plugins.form_elements.fields.textarea',
    'fobi.contrib.plugins.form_elements.fields.time',
    'fobi.contrib.plugins.form_elements.fields.url',

    # `django-fobi` form elements - content elements
    'fobi.contrib.plugins.form_elements.test.dummy',
    'easy_thumbnails', # Required by `content_image` plugin
    'fobi.contrib.plugins.form_elements.content.content_image',
    'fobi.contrib.plugins.form_elements.content.content_text',
    'fobi.contrib.plugins.form_elements.content.content_video',

    # `django-fobo` form handlers
    'fobi.contrib.plugins.form_handlers.db_store',
    'fobi.contrib.plugins.form_handlers.http_repost',
    'fobi.contrib.plugins.form_handlers.mail',

    # Other project specific apps
    'foo', # Test app
    # ...
)
```

3. Make appropriate changes to the `TEMPLATE_CONTEXT_PROCESSORS` of the your projects' Django settings.

And the following to the context processors.

```
TEMPLATE_CONTEXT_PROCESSORS = (
    # ...
    "fobi.context_processors.theme",
    # ...
)
```

Make sure that `django.core.context_processors.request` is in `TEMPLATE_CONTEXT_PROCESSORS` too.

4. Configure URLs

Add the following line to urlpatterns of your *urls* module.

```
# View URLs
url(r'^fobi/', include('fobi.urls.view')),

# Edit URLs
url(r'^fobi/', include('fobi.urls.edit')),
```

Note, that some plugins require additional URL includes. For instance, if you listed the

fobi.contrib.plugins.form_handlers.db_store form handler plugin in the `INSTALLED_APPS`, you should mention the following in *urls* module.

```
# DB Store plugin URLs
url(r'^fobi/plugins/form-handlers/db-store/',
    include('fobi.contrib.plugins.form_handlers.db_store.urls')),
```

View URLs are put separately from edit URLs in order to make it possible to prefix the edit URLs differently. For example, if you're using the "Simple" theme, you would likely want to prefix the edit URLs with "admin/" so that it looks more like django-admin.

CHAPTER 9

Creating a new form element plugin

Form element plugins represent the elements of which the forms is made: Inputs, checkboxes, textareas, files, hidden fields, as well as pure presentational elements (text or image). Number of form elements in a form is not limited.

Presentational form elements are inherited from `fobi.base.FormElementPlugin`.

The rest (real form elements, that are supposed to have a value) are inherited from `fobi.base.FormFieldPlugin`.

You should see a form element plugin as a Django micro app, which could have its' own models, admin interface, etc.

django-fobi comes with several bundled form element plugins. Do check the source code as example.

Let's say, you want to create a textarea form element plugin.

There are several properties, each textarea should have. They are:

- *label* (string): HTML label of the textarea.
- *name* (string): HTML name of the textarea.
- *initial* (string): Initial value of the textarea.
- *required* (bool): Flag, which tells us whether the field is required or optional.

Let's name that plugin *sample_textarea*. The plugin directory should then have the following structure.

```
path/to/sample_textarea/
-- __init__.py
-- fobi_form_elements.py # Where plugins are defined and registered
-- forms.py # Plugin configuration form
-- widgets.py # Where plugins widgets are defined
```

Form element plugins should be registered in "fobi_form_elements.py" file. Each plugin module should be put into the `INSTALLED_APPS` of your Django projects' settings.

In some cases, you would need plugin specific overridable settings (see `fobi.contrib.form_elements.fields.content.content_image` plugin as an example). You are advised to write your settings in such a way, that variables of your Django project settings module would have *FOBI_PLUGIN_* prefix.

Define and register the form element plugin

Step by step review of a how to create and register a plugin and plugin widgets. Note, that *django-fobi* auto-discovers your plugins if you place them into a file named *fobi_form_elements.py* of any Django app listed in `INSTALLED_APPS` of your Django projects' settings module.

path/to/sample_textarea/fobi_form_elements.py

A single form element plugin is registered by its' UID.

Required imports.

```
from django import forms
from fobi.base import FormFieldPlugin, form_element_plugin_registry
from path.to.sample_textarea.forms import SampleTextareaForm
```

Defining the Sample textarea plugin.

```
class SampleTextareaPlugin(FormFieldPlugin):
    """Sample textarea plugin."""

    uid = "sample_textarea"
    name = "Sample Textarea"
    form = SampleTextareaForm
    group = "Samples" # Group to which the plugin belongs to

    def get_form_field_instances(self, request=None, form_entry=None,
                                 form_element_entries=None, **kwargs):
        kwargs = {
            'required': self.data.required,
            'label': self.data.label,
            'initial': self.data.initial,
            'widget': forms.widgets.Textarea(attrs={})
        }

        return [(self.data.name, forms.CharField, kwargs),]
```

Registering the `SampleTextareaPlugin` plugin.

```
form_element_plugin_registry.register(SampleTextareaPlugin)
```

Note, that in case you want to define a pure presentational element, make use of `fobi.base.FormElementPlugin` for subclassing, instead of `fobi.base.FormFieldPlugin`. See the source of the content plugins (fobi.contrib.plugins.form_elements.content) as a an example.

For instance, the `captcha` and `honeypot` fields are implemented as form elements (subclasses the `fobi.base.FormElementPlugin`). The `db_store` form handler plugin does not save the form data of those elements. If you want the form element data to be saved, do inherit from `fobi.base.FormFieldPlugin`.

Hidden form element plugins, should be also having set the `is_hidden` property to True. By default it's set to False. That makes the hidden form elements to be rendered using as `django.forms.widgets.TextInput` widget in edit mode. In the view mode, the original widget that you assigned in your form element plugin would be used.

There might be cases, when you need to do additional handling of the data upon the successful form submission. In such cases, you will need to define a `submit_plugin_form_data` method in the plugin, which accepts the following arguments:

- *form_entry* (fobi.models.FormEntry): Form entry, which is being submitted.
- *request* (django.http.HttpRequest): The Django HTTP request.
- *form* (django.forms.Form): Form object (a valid one, which contains the `cleaned_data` attribute).
- *form_element_entries* (fobi.models.FormElementEntry): Form element entries for the *form_entry* given.
- (**)kwargs : Additional arguments.

Example (taken from fobi.contrib.plugins.form_elements.fields.file):

```
def submit_plugin_form_data(self, form_entry, request, form,
                            form_element_entries=None, **kwargs):
    """Submit plugin form data."""
    # Get the file path
    file_path = form.cleaned_data.get(self.data.name, None)
    if file_path:
        # Handle the upload
        saved_file = handle_uploaded_file(FILES_UPLOAD_DIR, file_path)
        # Overwrite ``cleaned_data`` of the ``form`` with path to moved
        # file.
        form.cleaned_data[self.data.name] = "{0}{1}".format(
            settings.MEDIA_URL, saved_file
        )

    # It's critically important to return the ``form`` with updated
    # ``cleaned_data``
    return form
```

In the example below, the original form is being modified. If you don't want the original form to be modified, do not return anything.

Check the file form element plugin (fobi.contrib.plugins.form_elements.fields.file) for complete example.

path/to/sample_textarea/forms.py

Why to have another file for defining forms? Just to keep the code clean and less messy, although you could perfectly define all your plugin forms in the module *fobi_form_elements.py*, it's recommended to keep it separate.

Take into consideration, that *forms.py* is not an auto-discovered file pattern. All your form element plugins should be registered in modules named *fobi_form_elements.py*.

Required imports.

```
from django import forms
from fobi.base import BasePluginForm
```

Form for for `SampleTextareaPlugin` form element plugin.

```
class SampleTextareaForm(forms.Form, BasePluginForm):
    """Sample textarea form."""

    plugin_data_fields = [
        ("name", ""),
        ("label", ""),
        ("initial", ""),
        ("required", False)
    ]

    name = forms.CharField(label="Name", required=True)
    label = forms.CharField(label="Label", required=True)
    initial = forms.CharField(label="Initial", required=False)
    required = forms.BooleanField(label="Required", required=False)
```

Note that although it's not being checked in the code, but for form field plugins the following fields should be present in the plugin form (`BasePluginForm`) and the form plugin (`FormFieldPlugin`):

- name

In some cases, you might want to do something with the data before it gets saved. For that purpose, `save_plugin_data` method has been introduced.

See the following example.

```
def save_plugin_data(self, request=None):
    """Saving the plugin data and moving the file."""
    file_path = self.cleaned_data.get('file', None)
    if file_path:
        saved_image = handle_uploaded_file(IMAGES_UPLOAD_DIR, file_path)
        self.cleaned_data['file'] = saved_image
```

path/to/sample_textarea/widgets.py

Required imports.

```
from fobi.base import FormElementPluginWidget
```

Defining the base plugin widget.

```
class BaseSampleTextareaPluginWidget(FormElementPluginWidget):
    """Base sample textarea plugin widget."""

    # Same as ``uid`` value of the ``SampleTextareaPlugin``.
    plugin_uid = "sample_textarea"
```

path/to/sample_layout/fobi_form_elements.py

Register in the registry (in some module which is for sure to be loaded; it's handy to do it in the theme module).

Required imports.

```
from fobi.base import form_element_plugin_widget_registry
from path.to.sample_textarea.widgets import BaseSampleTextareaPluginWidget
```

Define the theme specific plugin.

```
class SampleTextareaPluginWidget(BaseSampleTextareaPluginWidget):
    """Sample textarea plugin widget."""

    theme_uid = 'bootstrap3' # Theme for which the widget is loaded
    media_js = [
        'sample_layout/js/fobi.plugins.form_elements.sample_textarea.js',
    ]
    media_css = [
        'sample_layout/css/fobi.plugins.form_elements.sample_textarea.css',
    ]
```

Register the widget.

```
form_element_plugin_widget_registry.register(SampleTextareaPluginWidget)
```

Form element plugin final steps

Now, that everything is ready, make sure your plugin module is added to `INSTALLED_APPS`.

```
INSTALLED_APPS = (
    # ...
    'path.to.sample_textarea',
    # ...
)
```

Afterwards, go to terminal and type the following command.

```
./manage.py fobi_sync_plugins
```

If your HTTP server is running, you would then be able to see the new plugin in the edit form interface.

Dashboard URL: http://127.0.0.1:8000/fobi/

Note, that you have to be logged in, in order to use the dashboard. If your new plugin doesn't appear, set the `FOBI_DEBUG` to True in your Django's local settings module, re-run your code and check console for error notifications.

CHAPTER 10

Creating a new form handler plugin

Form handler plugins handle the form data. *django-fobi* comes with several bundled form handler plugins, among which is the `db_store` and `mail` plugins, which are responsible for saving the submitted form data into the database and mailing the data to recipients specified. Number of form handlers in a form is not limited. Certain form handlers are not configurable (for example the `db_store` form handler isn't), while others are (`mail`, `http_repost`).

You should see a form handler as a Django micro app, which could have its' own models, admin interface, etc.

By default, it's possible to use a form handler plugin multiple times per form. If you wish to allow form handler plugin to be used only once in a form, set the `allow_multiple` property of the plugin to False.

As said above, *django-fobi* comes with several bundled form handler plugins. Do check the source code as example.

Define and register the form handler plugin

Let's name that plugin *sample_mail*. The plugin directory should then have the following structure.

```
path/to/sample_mail/
-- __init__.py
-- fobi_form_handlers.py  # Where plugins are defined and registered
-- forms.py  # Plugin configuration form
```

Form handler plugins should be registered in "fobi_form_handlers.py" file. Each plugin module should be put into the `INSTALLED_APPS` of your Django projects' settings.

path/to/sample_mail/fobi_form_handlers.py

A single form handler plugin is registered by its' UID.

Required imports.

```
import json
from django.core.mail import send_mail
from fobi.base import FormHandlerPlugin, form_handler_plugin_registry
from path.to.sample_mail.forms import SampleMailForm
```

Defining the Sample mail handler plugin.

```
class SampleMailHandlerPlugin(FormHandlerPlugin):
    """Sample mail handler plugin."""

    uid = "sample_mail"
```

```
    name = _("Sample mail")
    form = SampleMailForm

    def run(self, form_entry, request, form):
        """To be executed by handler."""
        send_mail(
            self.data.subject,
            json.dumps(form.cleaned_data),
            self.data.from_email,
            [self.data.to_email],
            fail_silently=True
        )
```

Some form handlers are configurable, some others not. In order to have a user friendly way of showing the form handler settings, what's sometimes needed, a `plugin_data_repr` method has been introduced. Simplest implementation of it would look as follows:

```
def plugin_data_repr(self):
    """Human readable representation of plugin data.

    :return string:
    """
    return self.data.__dict__
```

path/to/sample_mail/forms.py

If plugin is configurable, it has configuration data. A single form may have unlimited number of same plugins. Imagine, you want to have different subjects and additional body texts for different user groups. You could then assign two form handler `mail` plugins to the form. Of course, saving the posted form data many times does not make sense, but it's up to the user. So, in case if plugin is configurable, it should have a form.

Why to have another file for defining forms? Just to keep the code clean and less messy, although you could perfectly define all your plugin forms in the module *fobi_form_handlers.py*, it's recommended to keep it separate.

Take into consideration, that *forms.py* is not an auto-discovered file pattern. All your form handler plugins should be registered in modules named *fobi_form_handlers.py*.

Required imports.

```
from django import forms
from django.utils.translation import ugettext_lazy as _
from fobi.base import BasePluginForm
```

Defining the form for Sample mail handler plugin.

```
class MailForm(forms.Form, BasePluginForm):
    """Mail form."""

    plugin_data_fields = [
        ("from_name", ""),
        ("from_email", ""),
        ("to_name", ""),
        ("to_email", ""),
        ("subject", ""),
        ("body", ""),
    ]

    from_name = forms.CharField(label=_("From name"), required=True)
```

```
    from_email = forms.EmailField(label=_("From email"), required=True)
    to_name = forms.CharField(label=_("To name"), required=True)
    to_email = forms.EmailField(label=_("To email"), required=True)
    subject = forms.CharField(label=_("Subject"), required=True)
    body = forms.CharField(label=_("Body"), required=False,
                           widget=forms.widgets.Textarea)
```

After the plugin has been processed, all its' data is available in a `plugin_instance.data` container (for example, `plugin_instance.data.subject` or `plugin_instance.data.from_name`).

Prioritise the execution order

Some form handlers shall be executed prior others. A good example of such, is a combination of "mail" and "db_save" form handlers for the form. In case if large files are posted, submission of form data would fail if "mail" plugin would be executed after "db_save" has been executed. That's why it's possible to prioritise that ordering in a `FOBI_FORM_HANDLER_PLUGINS_EXECUTION_ORDER` setting variable.

If not specified or left empty, form handler plugins would be ran in the order of discovery. All form handler plugins that are not listed in the `FORM_HANDLER_PLUGINS_EXECUTION_ORDER`, would be ran after the plugins that are mentioned there.

```
FORM_HANDLER_PLUGINS_EXECUTION_ORDER = (
    'http_repost',
    'mail',
    # The 'db_store' is left out intentionally, since it should
    # be the last plugin to be executed.
)
```

Form handler plugin custom actions

By default, a single form handler plugin has at least a "delete" action. If plugin is configurable, it gets an "edit" action as well.

For some of your plugins, you may want to register a custom action. For example, the "db_store" plugin does have one, for showing a link to a listing page with saved form data for the form given.

For such cases, define a `custom_actions` method in your form handler plugin. That method shall return a list of triples. In each triple, first value is the URL, second value is the title and the third value is the icon of the URL.

The following example is taken from the "db_store" plugin.

```
def custom_actions(self):
    """Adding a link to view the saved form entries.

    :return iterable:
    """
    return (
        (
            reverse('fobi.contrib.plugins.form_handlers.db_store.view_saved_form_data_entries'),
            _("View entries"),
            'glyphicon glyphicon-list'
        ),
    )
```

Form handler plugin final steps

Do not forget to add the form handler plugin module to `INSTALLED_APPS`.

```
INSTALLED_APPS = (
    # ...
    'path.to.sample_mail',
    # ...
)
```

Afterwards, go to terminal and type the following command.

```
./manage.py fobi_sync_plugins
```

If your HTTP server is running, you would then be able to see the new plugin in the edit form interface.

CHAPTER 11

Creating a new form importer plugin

Form importer plugins import the forms from some external data source into *django-fobi* form format. Number of form importers is not limited. Form importers are implemented in forms of wizards (since they may contain several steps).

You should see a form importer as a Django micro app, which could have its' own models, admin interface, etc.

At the moment *django-fobi* comes with only one bundled form handler plugin, which is the `mailchimp_importer`, which is responsible for importing existing MailChimp forms into *django-fobi*.

Define and register the form importer plugin

Let's name that plugin *sample_importer*. The plugin directory should then have the following structure.

```
path/to/sample_importer/
-- templates
|   -- sample_importer
|       -- 0.html
|       -- 1.html
-- __init__.py
-- fobi_form_importers.py # Where plugins are defined and registered
-- forms.py # Wizard forms
-- views.py # Wizard views
```

Form importer plugins should be registered in "fobi_form_importers.py" file. Each plugin module should be put into the `INSTALLED_APPS` of your Django projects' settings.

path/to/sample_importer/fobi_form_importers.py

A single form importer plugin is registered by its' UID.

Required imports.

```
from django.utils.translation import ugettext_lazy as _
from fobi.form_importers import BaseFormImporter, form_importer_plugin_registry
from fobi.contrib.plugins.form_elements import fields
from path.to.sample_importer.views import SampleImporterWizardView
```

Defining the Sample importer plugin.

```
class SampleImporterPlugin(FormHandlerPlugin):
    """Sample importer plugin."""

    uid = 'sample_importer'
    name = _("Sample importer)
    wizard = SampleImporterWizardView
    templates = [
        'sample_importer/0.html',
        'sample_importer/1.html',
    ]

    # field_type (at importer): uid (django-fobi)
    fields_mapping = {
        # Implemented
        'email': fields.email.UID,
        'text': fields.text.UID,
        'number': fields.integer.UID,
        'dropdown': fields.select.UID,
        'date': fields.date.UID,
        'url': fields.url.UID,
        'radio': fields.radio.UID,

        # Transformed into something else
        'address': fields.text.UID,
        'zip': fields.text.UID,
        'phone': fields.text.UID,
    }

    # Django standard: remote
    field_properties_mapping = {
        'label': 'name',
        'name': 'tag',
        'help_text': 'helptext',
        'initial': 'default',
        'required': 'req',
        'choices': 'choices',
    }

    field_type_prop_name = 'field_type'
    position_prop_name = 'order'

    def extract_field_properties(self, field_data):
        field_properties = {}
        for prop, val in self.field_properties_mapping.items():
            if val in field_data:
                if 'choices' == val:
                    field_properties[prop] = "\n".join(field_data[val])
                else:
                    field_properties[prop] = field_data[val]
        return field_properties


form_importer_plugin_registry.register(SampleImporter)
```

path/to/sample_importer/forms.py

As mentioned above, form importers are implemented in form of wizards. The forms are the wizard steps.

Required imports.

```
from django import forms
from django.utils.translation import ugettext_lazy as _
from sample_service_api import sample_api # Just an imaginary API client
```

Defining the form for Sample importer plugin.

```
class SampleImporterStep1Form(forms.Form):
    """First form the the wizard."""

    api_key = forms.CharField(required=True)


class SampleImporterStep2Form(forms.Form):
    """Second form of the wizard."""

    list_id = forms.ChoiceField(required=True, choices=[])

    def __init__(self, *args, **kwargs):
        self._api_key = None

        if 'api_key' in kwargs:
            self._api_key = kwargs.pop('api_key', None)

        super(SampleImporterStep2Form, self).__init__(*args, **kwargs)

        if self._api_key:
            client = sample_api.Api(self._api_key)
            lists = client.lists.list()
            choices = [(l['id'], l['name']) for l in lists['data']]
            self.fields['list_id'].choices = choices
```

path/to/sample_importer/views.py

The wizard views.

Required imports.

```
from sample_service_api import sample_api  # Just an imaginary API client

from django.shortcuts import redirect
from django.core.urlresolvers import reverse
from django.contrib import messages
from django.utils.translation import ugettext_lazy as _

# For django LTE 1.8 import from `django.contrib.formtools.wizard.views`
from formtools.wizard.views import SessionWizardView

from path.to.sample_importer.forms import (
    SampleImporterStep1Form, SampleImporterStep2Form
)
```

Defining the wizard view for Sample importer plugin.

```
class SampleImporterWizardView(SessionWizardView):
    """Sample importer wizard view."""
```

```
    form_list = [SampleImporterStep1Form, SampleImporterStep2Form]

    def get_form_kwargs(self, step):
        """Get form kwargs (to be used internally)."""
        if '1' == step:
            data = self.get_cleaned_data_for_step('0') or {}
            api_key = data.get('api_key', None)
            return {'api_key': api_key}
        return {}

    def done(self, form_list, **kwargs):
        """After all forms are submitted."""
        # Merging cleaned data into one dict
        cleaned_data = {}
        for form in form_list:
            cleaned_data.update(form.cleaned_data)

        # Connecting to sample client API
        client = sample_client.Api(cleaned_data['api_key'])

        # Fetching the form data
        form_data = client.lists.merge_vars(
            id={'list_id': cleaned_data['list_id']}
        )

        # We need the first form only
        try:
            form_data = form_data['data'][0]
        except Exception as err:
            messages.warning(
                self.request,
                _('Selected form could not be imported due errors.')
            )
            return redirect(reverse('fobi.dashboard'))

        # Actually, import the form
        form_entry = self._form_importer.import_data(
            {'name': form_data['name'], 'user': self.request.user},
            form_data['merge_vars']
        )

        redirect_url = reverse(
            'fobi.edit_form_entry', kwargs={'form_entry_id': form_entry.pk}
        )

        messages.info(
            self.request,
            _('Form {0} imported successfully.').format(form_data['name'])
        )

        return redirect("{0}".format(redirect_url))
```

Form importer plugin final steps

Do not forget to add the form importer plugin module to `INSTALLED_APPS`.

```
INSTALLED_APPS = (
    # ...
    'path.to.sample_importer',
    # ...
)
```

Afterwards, go to terminal and type the following command.

```
./manage.py fobi_sync_plugins
```

If your HTTP server is running, you would then be able to see the new plugin in the dashboard form interface (implemented in all bundled themes).

CHAPTER 12

Creating a form callback

Form callbacks are additional hooks, that are executed on various stages of the form submission.

Let's place the callback in the *foo* module. The plugin directory should then have the following structure.

```
path/to/foo/
-- __init__.py
-- fobi_form_callbacks.py # Where callbacks are defined and registered
```

See the callback example below.

Required imports.

```
from fobi.constants import (
    CALLBACK_BEFORE_FORM_VALIDATION,
    CALLBACK_FORM_VALID_BEFORE_SUBMIT_PLUGIN_FORM_DATA,
    CALLBACK_FORM_VALID, CALLBACK_FORM_VALID_AFTER_FORM_HANDLERS,
    CALLBACK_FORM_INVALID
)
from fobi.base import FormCallback, form_callback_registry
```

Define and register the callback

```
class SampleFooCallback(FormCallback):
    """Sample foo callback."""

    stage = CALLBACK_FORM_VALID

    def callback(self, form_entry, request, form):
        """Define your callback code here."""
        print("Great! Your form is valid!")

form_callback_registry.register(SampleFooCallback)
```

Add the callback module to `INSTALLED_APPS`.

```
INSTALLED_APPS = (
    # ...
    'path.to.foo',
    # ...
)
```

CHAPTER 13

Suggestions

Custom action for the form

Sometimes, you would want to specify a different action for the form. Although it's possible to define a custom form action (`action` field in the "Form properties" tab), you're advised to use the `http_repost` plugin instead, since then the form would be still validated locally and only then the valid data, as is, would be sent to the desired endpoint.

Take in mind, that if both cases, if CSRF protection is enabled on the endpoint, your post request would result an error.

When you want to customise too many things

django-fobi, with its' flexible form elements, form handlers and form callbacks is very customisable. However, there might be cases when you need to override entire view to fit your needs. Take a look at the FeinCMS integration or DjangoCMS integration as a good example of such. You may also want to compare the code from original view `fobi.views.view_form_entry` with the code from the widget to get a better idea of what could be changed in your case. If need a good advice, just ask me.

CHAPTER 14

Theming

django-fobi comes with theming API. While there are several ready-to-use themes:

- "Bootstrap 3" theme
- "Foundation 5" theme
- "Simple" theme in (with editing interface in style of the Django admin)
- "DjangoCMS admin style" theme (which is another simple theme with editing interface in style of *djangocms-admin-style*)

Obviously, there are two sorts of views when it comes to editing and viewing the form.

- The "view-view", when the form as it has been made is exposed to the site end- users/visitors.
- The "edit-view" (builder view), where the authorised users build their forms.

Both "Bootstrap 3" and "Foundation 5" themes are making use of the same style for both "view-view" and "edit-view" views.

Both "Simple" and "DjangoCMS admin style" themes are styling for the "edit-view" only. The "view-view" is pretty much blank, as shown on the one of the screenshots *[2.6]*.

Have in mind, that creating a brand new theme could be time consuming. Instead, you are advised to extend existing themes or in the worst case, if too much customisation required, create your own themes based on existing ones (just copy the desired theme to your project directory and work it out further).

It's possible to use different templates for all "view" and "edit" actions (see the source code of the "simple" theme). Both "Bootstrap 3" and "Foundation 5" themes look great. Although if you can't use any of those, the "Simple" theme is the best start, since it looks just like django-admin.

Create a new theme

Let's place the theme in the *sample_theme* module. The theme directory should then have the following structure.

```
path/to/sample_theme/
-- static
|   -- css
|   |   -- sample_theme.css
|   -- js
|       -- sample_theme.js
-- templates
|   -- sample_theme
|       -- _base.html
```

```
|       -- add_form_element_entry.html
|       -- ...
|       -- view_form_entry_ajax.html
-- __init__.py
-- fobi_form_elements.py
-- fobi_themes.py # Where themes are defined and registered
```

See the theme example below.

```
from django.utils.translation import ugettext_lazy as _

from fobi.base import BaseTheme, theme_registry

class SampleTheme(BaseTheme):
    """Sample theme."""

    uid = 'sample'
    name = _("Sample")

    media_css = (
        'sample_theme/css/sample_theme.css',
        'css/fobi.core.css',
    )

    media_js = (
        'js/jquery-1.10.2.min.js',
        'jquery-ui/js/jquery-ui-1.10.3.custom.min.js',
        'js/jquery.slugify.js',
        'js/fobi.core.js',
        'sample_theme/js/sample_theme.js',
    )

    # Form element specific
    form_element_html_class = 'form-control'
    form_radio_element_html_class = 'radio'
    form_element_checkbox_html_class = 'checkbox'

    form_edit_form_entry_option_class = 'glyphicon glyphicon-edit'
    form_delete_form_entry_option_class = 'glyphicon glyphicon-remove'
    form_list_container_class = 'list-inline'

    # Templates
    master_base_template = 'sample_theme/_base.html'
    base_template = 'sample_theme/base.html'

    form_ajax = 'sample_theme/snippets/form_ajax.html'
    form_snippet_template_name = 'sample_theme/snippets/form_snippet.html'
    form_properties_snippet_template_name = 'sample_theme/snippets/form_properties_snippet.html'
    messages_snippet_template_name = 'sample_theme/snippets/messages_snippet.html'

    add_form_element_entry_template = 'sample_theme/add_form_element_entry.html'
    add_form_element_entry_ajax_template = 'sample_theme/add_form_element_entry_ajax.html'

    add_form_handler_entry_template = 'sample_theme/add_form_handler_entry.html'
    add_form_handler_entry_ajax_template = 'sample_theme/add_form_handler_entry_ajax.html'

    create_form_entry_template = 'sample_theme/create_form_entry.html'
    create_form_entry_ajax_template = 'bootstrap3/create_form_entry_ajax.html'
```

```
    dashboard_template = 'sample_theme/dashboard.html'

    edit_form_element_entry_template = 'sample_theme/edit_form_element_entry.html'
    edit_form_element_entry_ajax_template = 'sample_theme/edit_form_element_entry_ajax.html'

    edit_form_entry_template = 'sample_theme/edit_form_entry.html'
    edit_form_entry_ajax_template = 'sample_theme/edit_form_entry_ajax.html'

    edit_form_handler_entry_template = 'sample_theme/edit_form_handler_entry.html'
    edit_form_handler_entry_ajax_template = 'sample_theme/edit_form_handler_entry_ajax.html'

    form_entry_submitted_template = 'sample_theme/form_entry_submitted.html'
    form_entry_submitted_ajax_template = 'sample_theme/form_entry_submitted_ajax.html'

    view_form_entry_template = 'sample_theme/view_form_entry.html'
    view_form_entry_ajax_template = 'sample_theme/view_form_entry_ajax.html'
```

Registering the `SampleTheme` plugin.

```
theme_registry.register(SampleTheme)
```

Sometimes you would want to attach additional properties to the theme in order to use them later in templates (remember, current theme object is always available in templates under name *fobi_theme*).

For such cases you would need to define a variable in your project's settings module, called `FOBI_CUSTOM_THEME_DATA`. See the following code as example:

```
# `django-fobi` custom theme data for to be displayed in third party apps
# like `django-registraton`.
FOBI_CUSTOM_THEME_DATA = {
    'bootstrap3': {
        'page_header_html_class': '',
        'form_html_class': 'form-horizontal',
        'form_button_outer_wrapper_html_class': 'control-group',
        'form_button_wrapper_html_class': 'controls',
        'form_button_html_class': 'btn',
        'form_primary_button_html_class': 'btn-primary pull-right',
    },
    'foundation5': {
        'page_header_html_class': '',
        'form_html_class': 'form-horizontal',
        'form_button_outer_wrapper_html_class': 'control-group',
        'form_button_wrapper_html_class': 'controls',
        'form_button_html_class': 'radius button',
        'form_primary_button_html_class': 'btn-primary',
    },
    'simple': {
        'page_header_html_class': '',
        'form_html_class': 'form-horizontal',
        'form_button_outer_wrapper_html_class': 'control-group',
        'form_button_wrapper_html_class': 'submit-row',
        'form_button_html_class': 'btn',
        'form_primary_button_html_class': 'btn-primary',
    }
}
```

You would now be able to access the defined extra properties in templates as shown below.

```
<div class="{{ fobi_theme.custom_data.form_button_wrapper_html_class }}">
```

You likely would want to either remove the footer text or change it. Define a variable in your project's settings module, called `FOBI_THEME_FOOTER_TEXT`. See the following code as example:

```
FOBI_THEME_FOOTER_TEXT = gettext('&copy; django-fobi example site 2014')
```

Below follow the properties of the theme:

- `base_edit`
- `base_view`

There are generic templates made in order to simplify theming. Some of them you would never need to override. Some others, you would likely want to.

Templates that you likely would want to re-write in your custom theme implementation are marked with three asterisks (***):

```
generic
-- snippets
|   -- form_ajax.html
|   -- form_edit_ajax.html
|   -- *** form_properties_snippet.html
|   -- *** form_snippet.html
|   -- --- form_edit_snippet.html (does not exist in generic templates)
|   -- --- form_view_snippet.html (does not exist in generic templates)
|   -- form_view_ajax.html
|   -- messages_snippet.html
|
-- _base.html
-- add_form_element_entry.html
-- add_form_element_entry_ajax.html
-- add_form_handler_entry.html
-- add_form_handler_entry_ajax.html
-- base.html
-- create_form_entry.html
-- create_form_entry_ajax.html
-- *** dashboard.html
-- edit_form_element_entry.html
-- edit_form_element_entry_ajax.html
-- edit_form_entry.html
-- *** edit_form_entry_ajax.html
-- edit_form_handler_entry.html
-- edit_form_handler_entry_ajax.html
-- form_entry_submitted.html
-- *** form_entry_submitted_ajax.html
-- *** theme.html
-- view_form_entry.html
-- view_form_entry_ajax.html
```

From all of the templates listed above, the _base.html template is the most influenced by the Bootstrap 3 theme.

Make changes to an existing theme

As said above, making your own theme from scratch could be costly. Instead, you can override/reuse an existing one and change it to your needs with minimal efforts. See the override simple theme example. In order to see it in action, run the project with settings_override_simple_theme option:

```
./manage.py runserver --settings=settings_override_simple_theme
```

Details explained below.

Directory structure

```
override_simple_theme/
-- static
|   -- override_simple_theme
|       -- css
|       |   -- override-simple-theme.css
|       -- js
|           -- override-simple-theme.js
|
-- templates
|   -- override_simple_theme
|       -- snippets
|       |   -- form_ajax.html
|       -- base_view.html
-- __init__.py
-- fobi_themes.py # Where themes are defined and registered
```

fobi_themes.py

Overriding the "simple" theme.

```
__all__ = ('MySimpleTheme',)

from fobi.base import theme_registry

from fobi.contrib.themes.simple.fobi_themes import SimpleTheme

class MySimpleTheme(SimpleTheme):
    """My simple theme, inherited from `SimpleTheme` theme."""

    html_classes = ['my-simple-theme',]
    base_view_template = 'override_simple_theme/base_view.html'
    form_ajax = 'override_simple_theme/snippets/form_ajax.html'
```

Register the overridden theme. Note, that it's important to set the *force* argument to True, in order to override the original theme. Force can be applied only once (for an overridden element).

```
theme_registry.register(MySimpleTheme, force=True)
```

templates/override_simple_theme/base_view.html

```
{% extends "simple/base_view.html" %}

{% load static %}

{% block stylesheets %}
<link
  href="{% static 'override_simple_theme/css/override-simple-theme.css' %}"
  rel="stylesheet" media="all" />
```

```
{% endblock stylesheets %}

{% block main-wrapper %}
<div id="sidebar">
  <h2>It's easy to override a theme!</h2>
</div>

{{ block.super }}
{% endblock main-wrapper %}
```

templates/override_simple_theme/snippets/form_ajax.html

```
{% extends "fobi/generic/snippets/form_ajax.html" %}

{% block form_html_class %}basic-grey{% endblock %}
```

CHAPTER 15

Form wizards

Basics

With form wizards you can split forms across multiple pages. State is maintained in one of the backends (at the moment the Session backend). Data processing is delayed until the submission of the final form.

In *django-fobi* wizards work in the following way:

- Number of forms in a form wizard is not limited.
- Form callbacks, handlers are totally ignored in form wizards. Instead, the form-wizard specific handlers (form wizard handlers) take over handling of the form data on the final step.

Bundled form wizard handler plugins

Below a short overview of the form wizard handler plugins. See the README.rst file in directory of each plugin for details.

- DB store: Stores form data in a database.
- HTTP repost: Repost the POST request to another endpoint.
- Mail: Send the form data by email.

CHAPTER 16

Permissions

Plugin system allows administrators to specify the access rights to every plugin. *django-fobi* permissions are based on Django Users and User Groups. Access rights are manageable via Django admin ("/admin/fobi/formelement/", "/admin/fobi/formhandler/"). If user doesn't have the rights to access plugin, it doesn't appear on his form even if has been added to it (imagine, you have once granted the right to use the news plugin to all users, but later on decided to limit it to Staff members group only). Note, that superusers have access to all plugins.

Plugin access rights management interface in Django admin

`Plugin`	`Users`	`Groups`
Text	John Doe	Form builder users
Textarea		Form builder users
File	Oscar, John Doe	Staff members
URL		Form builder users
Hidden		Form builder users

CHAPTER 17

Management commands

There are several management commands available.

- *fobi_find_broken_entries*. Find broken form element/handler entries that occur when some plugin which did exist in the system, no longer exists.
- *fobi_sync_plugins*. Should be ran each time a new plugin is being added to the *django-fobi*.
- *fobi_update_plugin_data*. A mechanism to update existing plugin data in case if it had become invalid after a change in a plugin. In order for it to work, each plugin should implement and `update` method, in which the data update happens.

CHAPTER 18

Tuning

There are number of *django-fobi* settings you can override in the settings module of your Django project:

- *FOBI_RESTRICT_PLUGIN_ACCESS* (bool): If set to True, (Django) permission system for dash plugins is enabled. Defaults to True. Setting this to False makes all plugins available for all users.
- *FOBI_DEFAULT_THEME* (str): Active (default) theme UID. Defaults to "bootstrap3".
- *FORM_HANDLER_PLUGINS_EXECUTION_ORDER* (list of tuples): Order in which the form handlers are executed. See the "Prioritise the execution order" section for details.

For tuning of specific contrib plugin, see the docs in the plugin directory.

CHAPTER 19

Bundled plugins and themes

django-fobi ships with number of bundled form element- and form handler- plugins, as well as themes which are ready to be used as is.

Bundled form element plugins

Below a short overview of the form element plugins. See the README.rst file in directory of each plugin for details.

Fields

Fields marked with asterisk (*) fall under the definition of text elements. It's possible to provide *Dynamic initial values* for text elements.

- Boolean (checkbox)
- Date
- DateTime
- Date drop down (year, month, day selection drop-downs)
- Decimal
- Email*
- File
- Float
- Hidden*
- Input
- IP address*
- Integer
- Null boolean
- Password*
- Radio select (radio button)
- Range select
- Select (drop-down)

- Select model object (drop-down)
- Select multiple (drop-down)
- Select multiple model objects (drop-down)
- Slider
- Slug*
- Text*
- Textarea*
- Time
- URL*

Content/presentation

Content plugins are presentational plugins, that make your forms look more complete and content rich.

- Content image: Insert an image.
- Content text: Add text.
- Content video: Add an embed YouTube or Vimeo video.

Security

- CAPTCHA: Captcha integration, requires `django-simple-captcha` package.
- ReCAPTCHA: Captcha integration, requires `django-recaptcha` package.
- Honeypot: Anti-spam honeypot field.

MPTT fields

- Select MPTT model object (drop-down)
- Select multiple MPTT model objects (drop-down)

Test

Test plugins are made for dev purposes only.

- Dummy: Solely for dev purposes.

Bundled form handler plugins

Below a short overview of the form handler plugins. See the README.rst file in directory of each plugin for details.

- DB store: Stores form data in a database.
- HTTP repost: Repost the POST request to another endpoint.
- Mail: Send the form data by email.

Bundled themes

Below a short overview of the themes. See the README.rst file in directory of each theme for details.

- Bootstrap 3: Bootstrap 3 theme.
- Foundation 5: Foundation 5 theme.
- Simple: Basic theme with form editing is in a style of Django admin.
- DjangoCMS admin style: Basic theme with form editing is in a style of djangocms-admin-style.

CHAPTER 20

Third-party plugins and themes

List of remarkable third-party plugins:

- fobi-phonenumber - A Fobi PhoneNumber form field plugin. Makes use of the *phonenumber_field.formfields.PhoneNumberField* and *phonenumber_field.widgets.PhoneNumberPrefixWidget*.

CHAPTER 21

HTML5 fields

The following HTML5 fields are supported in corresponding bundled plugins:

- date
- datetime
- email
- max
- min
- number
- url
- placeholder
- type

With the *fobi.contrib.plugins.form_elements.fields.input* support for HTML5 fields is extended to the following fields:

- autocomplete
- autofocus
- list
- multiple
- pattern
- step

CHAPTER 22

Loading initial data using GET arguments

It's possible to provide initial data for the form using the GET arguments.

In that case, along with the field values, you should be providing an additional argument named "fobi_initial_data", which doesn't have to hold a value. For example, if your form contains of fields named "email" and "age" and you want to provide initial values for those using GET arguments, you should be constructing your URL to the form as follows:

http://127.0.0.1:8001/fobi/view/test-form/?fobi_initial_data&email=test@example.com&age=19

CHAPTER 23

Dynamic initial values

It's possible to provide a dynamic initial value for any of the text elements. In order to do that, you should use the build-in context processor or make your own one. The only requirement is that you should store all values that should be exposes in the form as a dict for *fobi_dynamic_values* dictionary key. Beware, that passing the original request object might be unsafe in many ways. Currently, a stripped down version of the request object is being passed as a context variable.

```
TEMPLATE_CONTEXT_PROCESSORS = (
    # ...
    "fobi.context_processors.dynamic_values",
    # ...
)
```

```
def dynamic_values(request):
    return {
        'fobi_dynamic_values': {
            'request': StrippedRequest(request),
            'now': datetime.datetime.now(),
            'today': datetime.date.today(),
        }
    }
```

In your GUI, you should be refering to the initial values in the following way:

```
{{ request.path }} {{ now }} {{ today }}
```

Note, that you should not provide the *fobi_dynamic_values.* as a prefix. Currently, the following variables are available in the *fobi.context_processors.dynamic_values* context processor:

```
- request: Stripped HttpRequest object.

    - request.path: A string representing the full path to the requested page,
      not including the scheme or domain.

    - request.get_full_path(): Returns the path, plus an appended query string,
      if applicable.

    - request.is_secure():  Returns True if the request is secure; that is, if
      it was made with HTTPS.

    - request.is_ajax(): Returns True if the request was made via an
      XMLHttpRequest, by checking the HTTP_X_REQUESTED_WITH header for the
      string 'XMLHttpRequest'.
```

```
    - request.META: A stripped down standard Python dictionary containing the
      available HTTP headers.

        - HTTP_ACCEPT_ENCODING: Acceptable encodings for the response.

        - HTTP_ACCEPT_LANGUAGE: Acceptable languages for the response.

        - HTTP_HOST: The HTTP Host header sent by the client.

        - HTTP_REFERER: The referring page, if any.

        - HTTP_USER_AGENT: The client's user-agent string.

        - QUERY_STRING: The query string, as a single (unparsed) string.

        - REMOTE_ADDR: The IP address of the client.

    - request.user: Authenticated user.

        - request.user.email:

        - request.user.get_username(): Returns the username for the user. Since
          the User model can be swapped out, you should use this method
          instead of referencing the username attribute directly.

        - request.user.get_full_name(): Returns the first_name plus the
          last_name, with a space in between.

        - request.user.get_short_name(): Returns the first_name.

        - request.user.is_anonymous():

- now: datetime.datetime.now()

- today: datetime.date.today()
```

CHAPTER 24

Submitted form element plugins values

While some values of form element plugins are submitted as is, some others need additional processing. There are 3 types of behaviour taken into consideration:

- "val": value is being sent as is.
- "repr": (human readable) representation of the value is used.
- "mix": mix of value as is and human readable representation.

The following plugins have been made configurable in such a way, that developers can choose the desired behaviour in projects' settings:

- `FOBI_FORM_ELEMENT_CHECKBOX_SELECT_MULTIPLE_SUBMIT_VALUE_AS`
- `FOBI_FORM_ELEMENT_RADIO_SUBMIT_VALUE_AS`
- `FOBI_FORM_ELEMENT_SELECT_SUBMIT_VALUE_AS`
- `FOBI_FORM_ELEMENT_SELECT_MULTIPLE_SUBMIT_VALUE_AS`
- `FOBI_FORM_ELEMENT_SELECT_MODEL_OBJECT_SUBMIT_VALUE_AS`
- `FOBI_FORM_ELEMENT_SELECT_MULTIPLE_MODEL_OBJECTS_SUBMIT_VALUE_AS`

See the README.rst in each of the following plugins for more information.

- Checkbox select multiple (multiple checkboxes)
- Radio select (radio button)
- Select (drop-down)
- Select model object (drop-down)
- Select MPTT model object (drop-down)
- Select multiple (drop-down)
- Select multiple model objects (drop-down)
- Select multiple MPTT model objects (drop-down)

CHAPTER 25

Rendering forms using third-party libraries

You might want to render your forms using third-party libraries such as django-crispy-forms, django-floppyforms or other alternatives.

For that purpose you should override the "snippets/form_snippet.html" used by the theme you have chosen. Your template would then look similar to the one below (make sure to setup/configure your third-party form rendering library prior doing this).

Using *django-crispy-forms*

```
{% load crispy_forms_tags fobi_tags %}

{% block form_non_field_and_hidden_errors %}
    {% get_form_hidden_fields_errors form as form_hidden_fields_errors %}
    {% if form.non_field_errors or form_hidden_fields_errors %}
        {% include fobi_theme.form_non_field_and_hidden_errors_snippet_template %}
    {% endif %}
{% endblock form_non_field_and_hidden_errors %}

{% crispy form %}
```

Using *django-floppyforms*

```
{% load floppyforms fobi_tags %}

{% block form_non_field_and_hidden_errors %}
    {% get_form_hidden_fields_errors form as form_hidden_fields_errors %}
    {% if form.non_field_errors or form_hidden_fields_errors %}
        {% include fobi_theme.form_non_field_and_hidden_errors_snippet_template %}
    {% endif %}
{% endblock form_non_field_and_hidden_errors %}

{% form form %}
```

See how it's done in the override simple theme example.

CHAPTER 26

Import/export forms

There might be cases when you have *django-fobi* running on multiple instances and have already spend some time on making forms on one of the instances, and want to reuse those forms on another. You could of course re-create entire form in the GUI, but we can do better than that. It's possible to export forms into JSON format and import the exported forms again. It's preferable that you run both instances on the same versions of *django-fobi*, otherwise imports might break (although it might just work). There are two scenarios to deal with missing plugin errors, which you have don't yet have full control of. If both instances have the same set of form element and form handler plugins imports should go smoothly. It is though possible to make an import ignoring missing form element and form handler plugins. You would get an appropriate notice about that, but import will continue leaving the broken plugin data out.

CHAPTER 27

Available translations

English is the primary language.

- Dutch (core and plugins)
- German (core and plugins)
- Russian (core and plugins)

CHAPTER 28

Debugging

By default debugging is turned off. It means that broken form entries, which are entries with broken data, that are not possible to be shown, are just skipped. That's safe in production. Although, you for sure would want to see the broken entries in development. Set the `FOBI_DEBUG` to True in the `settings.py` of your project in order to do so.

Most of the errors are logged (DEBUG). If you have written a plugin and it somehow doesn't appear in the list of available plugins, do run the following management command since it not only syncs your plugins into the database, but also is a great way of checking for possible errors.

```
./manage.py fobi_sync_plugins
```

Run the following command in order to identify the broken plugins.

```
./manage.py fobi_find_broken_entries
```

If you have forms referring to form element- of form handler- plugins that are currently missing (not registered, removed, failed to load - thus there would be a risk that your form would't be rendered properly/fully and the necessary data handling wouldn't happen either) you will get an appropriate exception. Although it's fine to get an instant error message about such failures in development, in production is wouldn't look appropriate. Thus, there are two settings related to the non-existing (not-found) form element- and form handler- plugins.

- FOBI_DEBUG: Set this to True in your development environment anyway. Watch error logs closely.
- FOBI_FAIL_ON_MISSING_FORM_ELEMENT_PLUGINS: If you want no error to be shown in case of missing form element plugins, set this to False in your settings module. Default value is True.
- FOBI_FAIL_ON_MISSING_FORM_HANDLER_PLUGINS: If you want no error to be shown in case of missing form element handlers, set this to False in your settings module. Default value is True.

CHAPTER 29

Testing

Project is covered by test (functional- and browser-tests).

To test with all supported Python/Django versions type:

```
tox
```

To test against specific environment, type:

```
tox -e pypy-django18
```

To test just your working environment type:

```
./runtests.py
```

It's assumed that you have all the requirements installed. If not, first install the test requirements:

```
pip install -r examples/requirements/common_test_requirements.txt
```

Browser tests

For browser tests you may choose between Firefox and PhantomJS. PhantomJS is faster, Firefox tests tell you more. Both cases require some effort and both have disadvantages regarding the installation (although once you have them installed they work perfect).

Latest versions of Firefox are often not supported by Selenium. Current version of the Selenium for Python (2.53.6) works fine with Firefox 47. Thus, instead of using system Firefox you could better use a custom one.

For PhantomJS you need to have NodeJS installed.

Set up Firefox 47

1. Download Firefox 47 from this location and unzip it into `/usr/lib/firefox47/`
2. Specify the full path to your Firefox in `FIREFOX_BIN_PATH` setting. Example:

```
FIREFOX_BIN_PATH = '/usr/lib/firefox47/firefox'
```

   If you set `FIREFOX_BIN_PATH` to None, system Firefox would be used.

After that your Selenium tests would work.

Setup PhantomJS

You could also run tests in headless mode (faster). For that you will need PhantomJS.

1. Install PhantomJS and dependencies.

```
curl -sL https://deb.nodesource.com/setup_6.x -o nodesource_setup.sh
sudo bash nodesource_setup.sh
sudo apt-get install nodejs
sudo apt-get install build-essential libssl-dev
sudo npm -g install phantomjs-prebuilt
```

2. Specify the `PHANTOM_JS_EXECUTABLE_PATH` setting. Example:

```
PHANTOM_JS_EXECUTABLE_PATH = ""
```

   If you want to use Firefox for testing, set `PHANTOM_JS_EXECUTABLE_PATH` to None.

CHAPTER 30

Troubleshooting

If you get a `FormElementPluginDoesNotExist` or a `FormHandlerPluginDoesNotExist` exception, make sure you have listed your plugin in the *settings* module of your project.

CHAPTER 31

License

GPL 2.0/LGPL 2.1

CHAPTER 32

Support

For any issues contact me at the e-mail given in the *Author* section.

CHAPTER 33

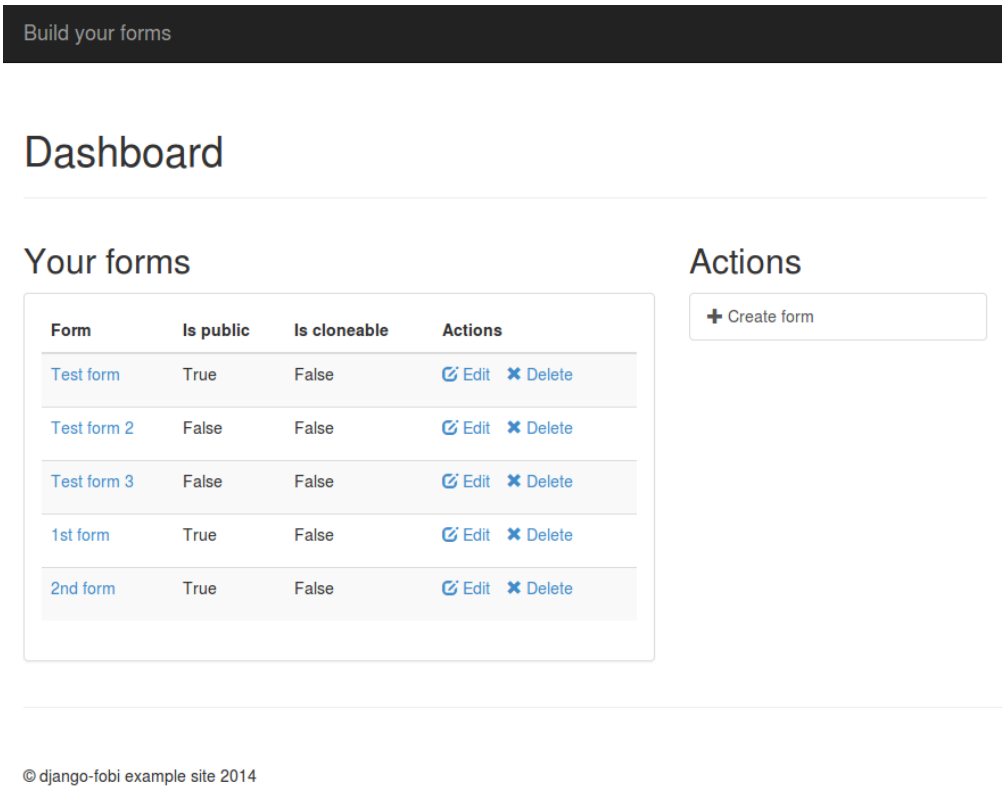
Author

Artur Barseghyan <artur.barseghyan@gmail.com>

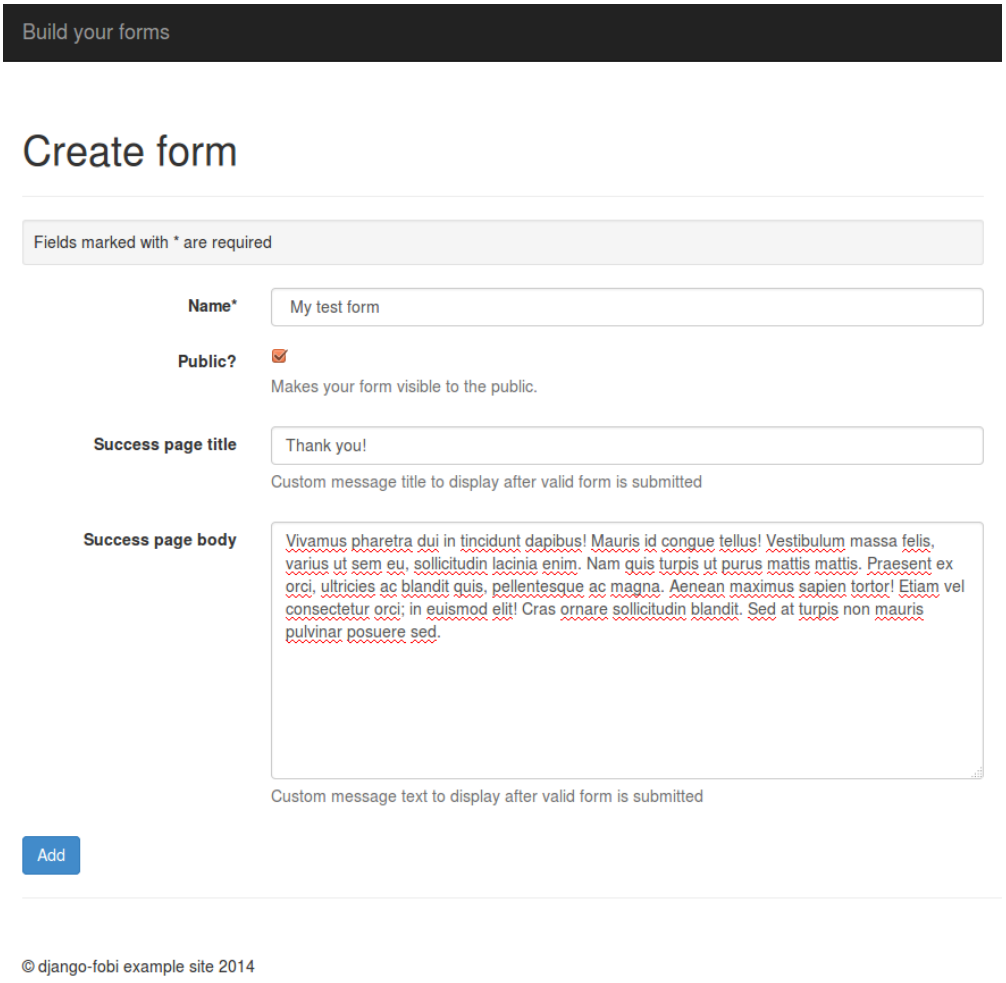
Screenshots

Bootstrap3 theme

Dashboard

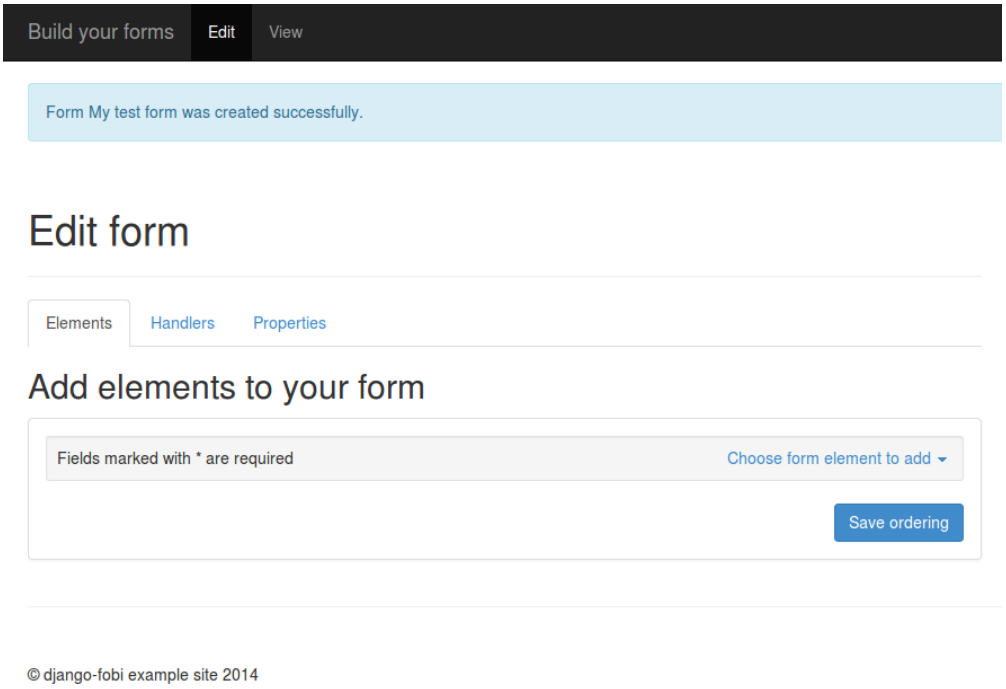


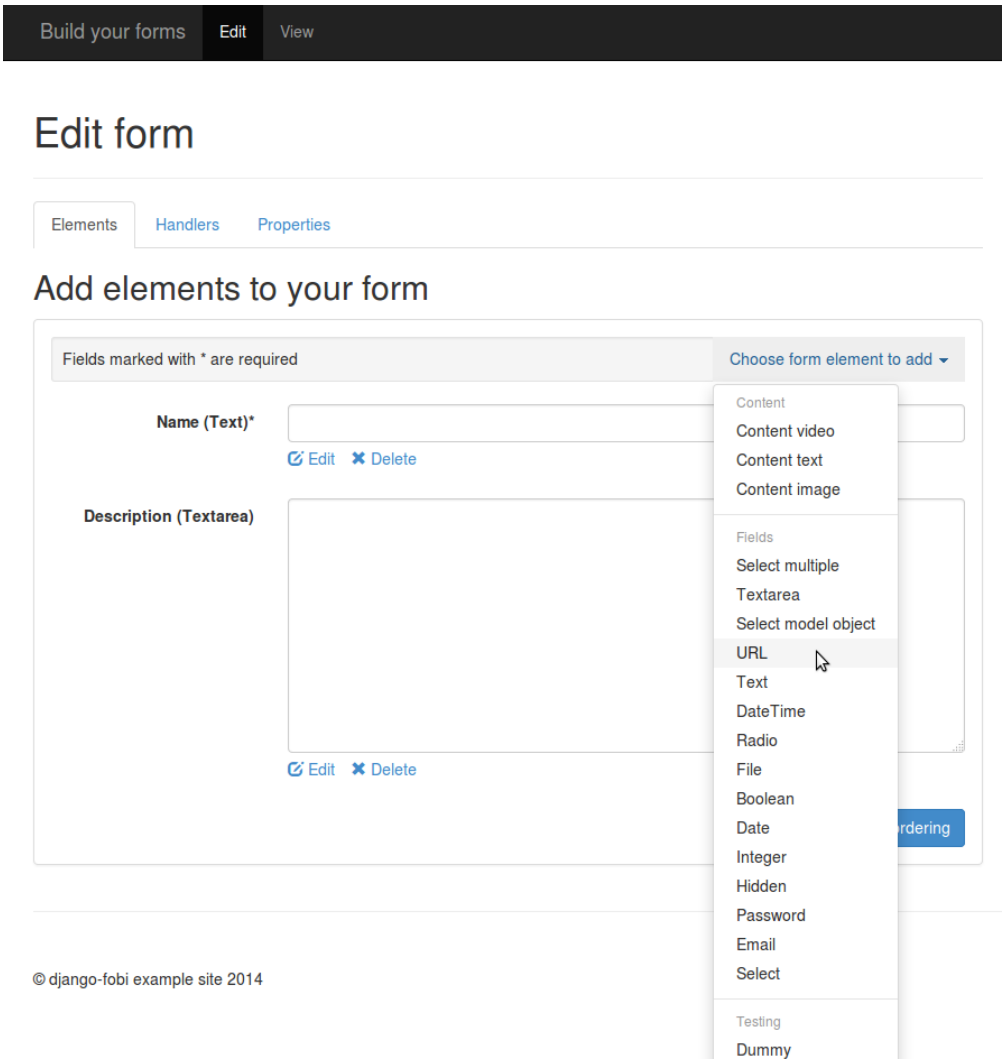
Create a form

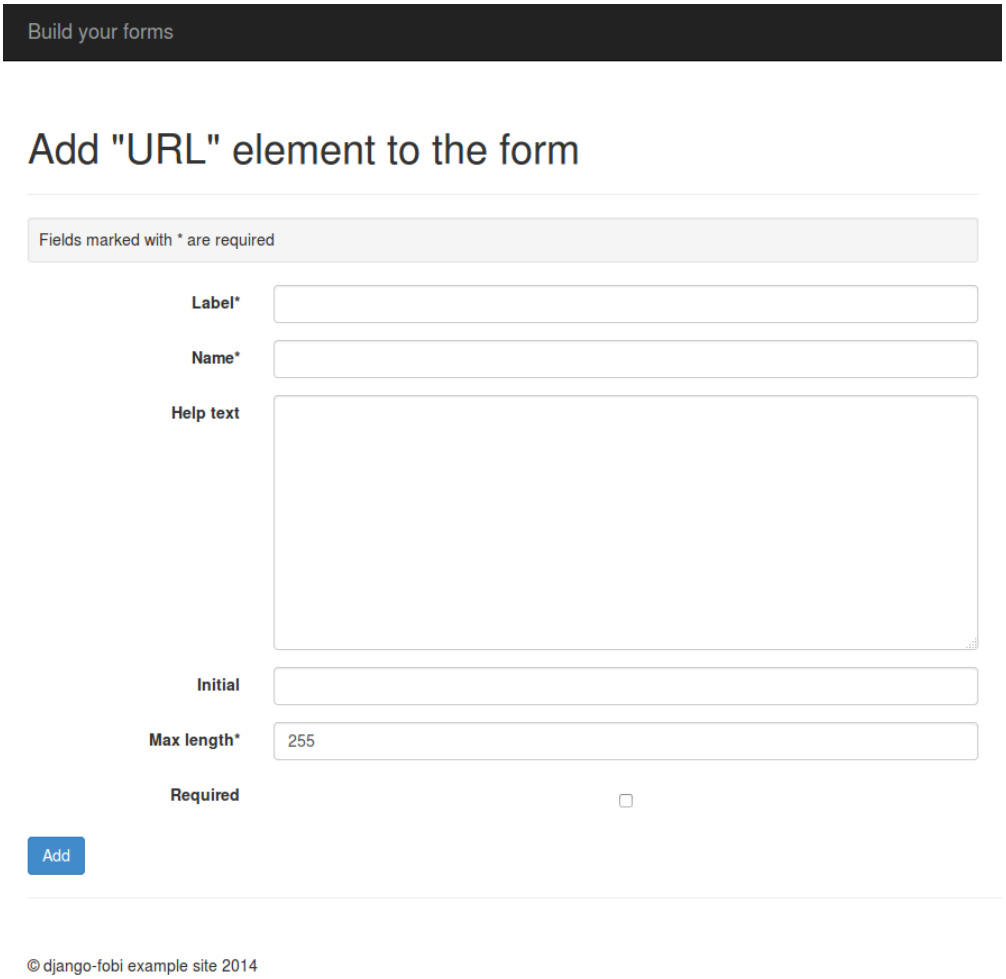


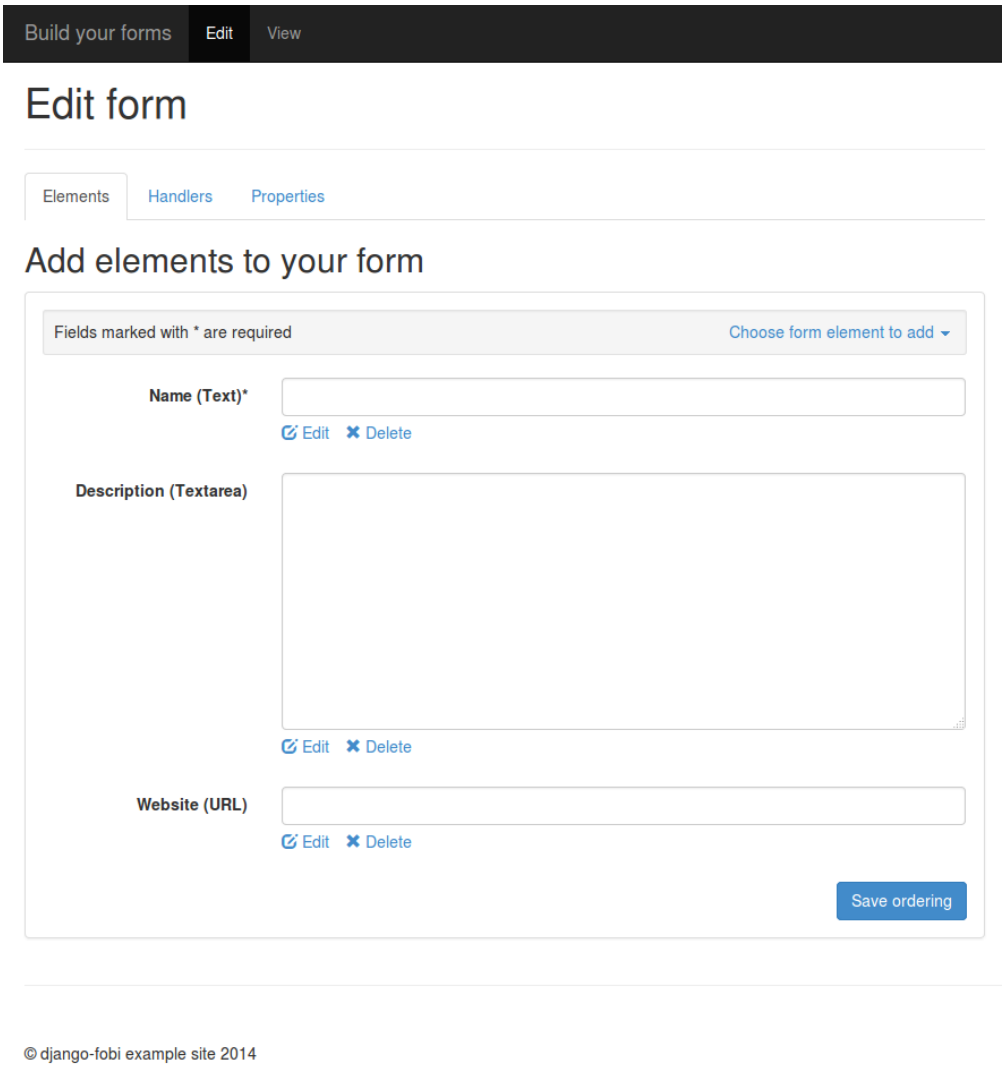
View/edit form

Form elements

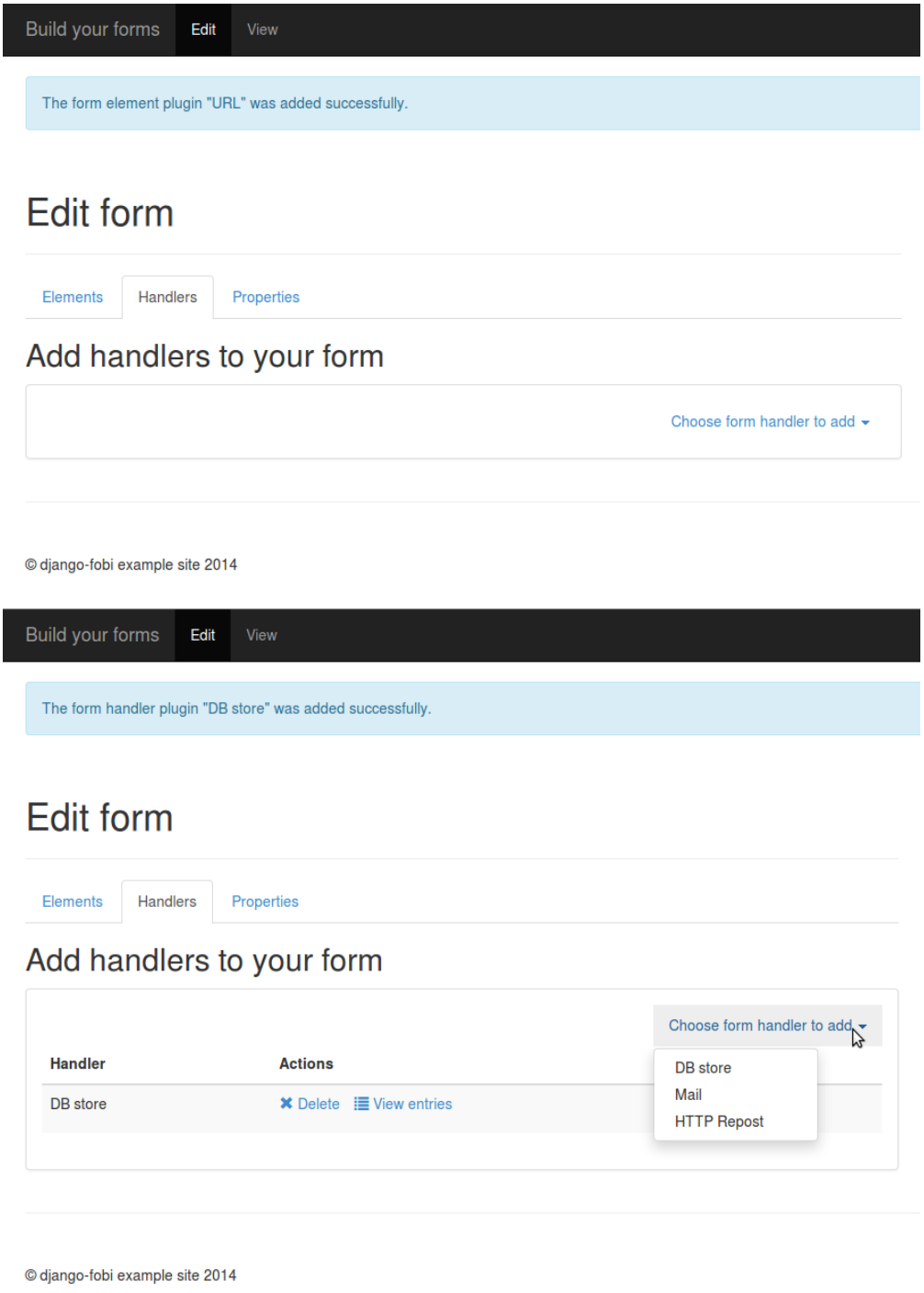


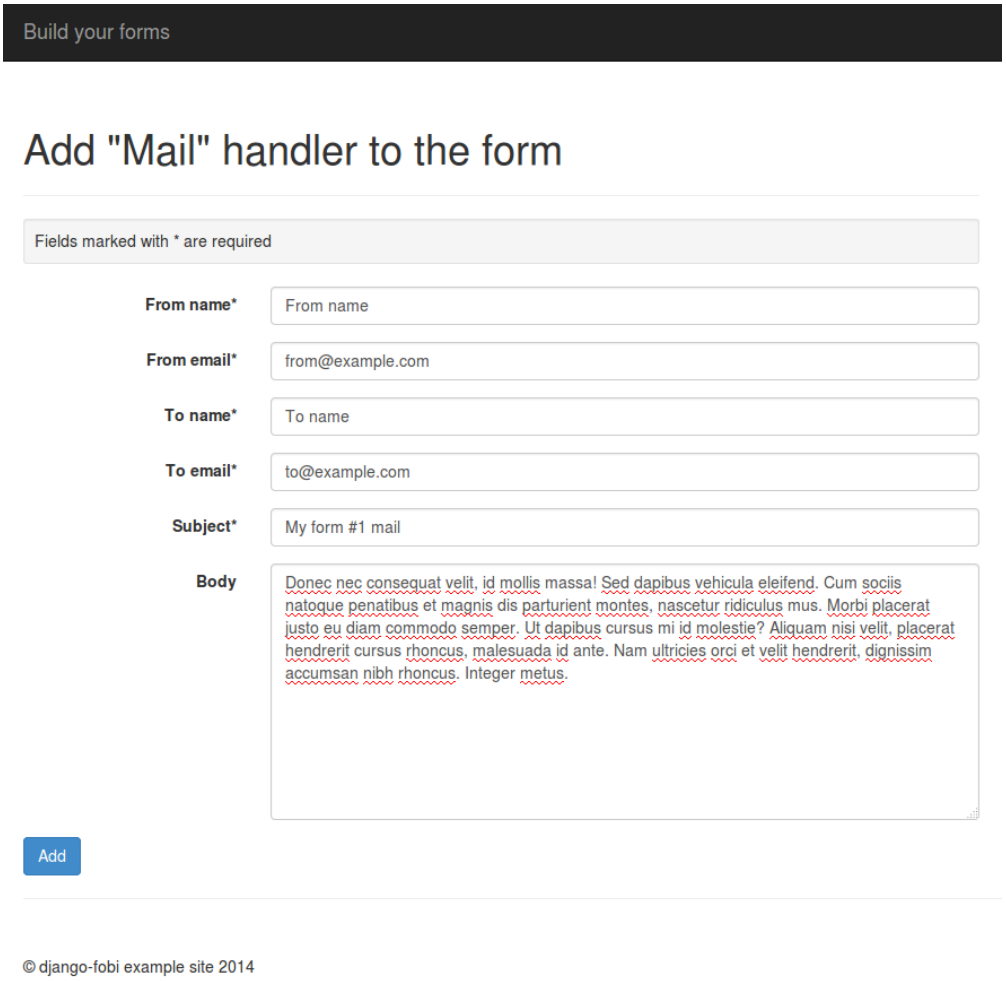

Build your forms
Edit
View
Edit form
Elements
Handlers
Properties
Add elements to your form
Fields marked with * are required
Choose form element to add
Name (Text)*
Edit
Delete
Description (Textarea)
Edit
Delete
ordering
Content
Content video
Content text
Content image
Fields
Select multiple
Textarea
Select model object
URL
Text
DateTime
Radio
File
Boolean
Date
Integer
Hidden
Password
Email
Select
Testing
Dummy
© django-fobi example site 2014

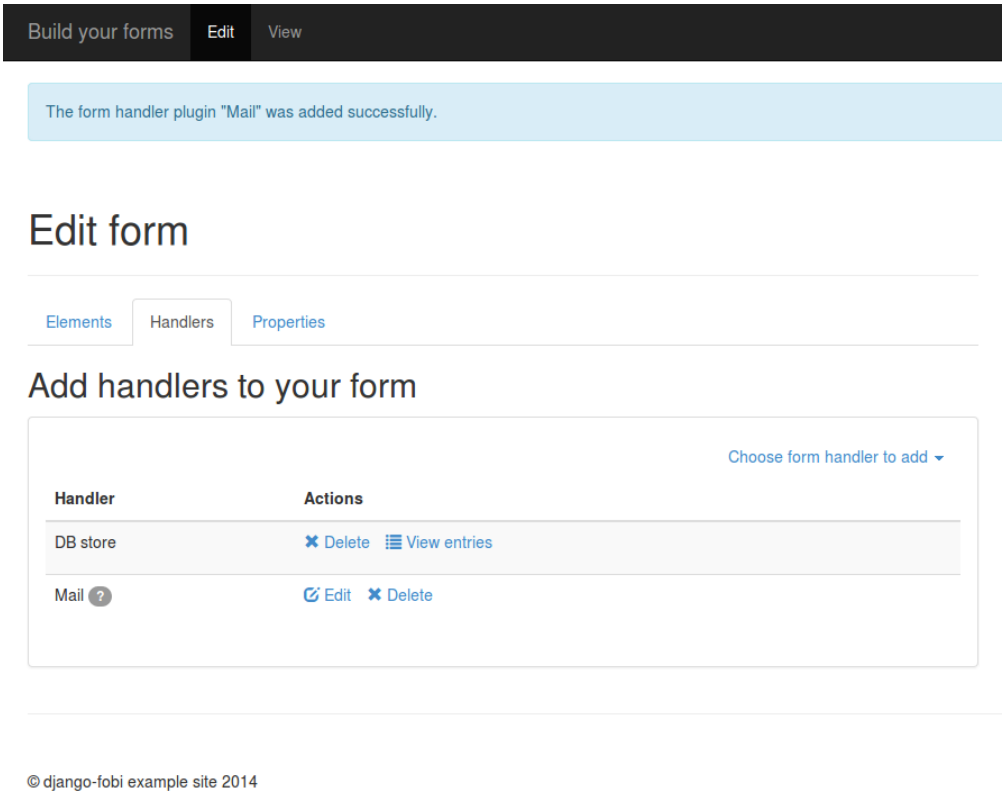

Build your forms
Add "URL" element to the form
Fields marked with * are required
Label*
Name*
Help text
Initial
Max length*
255
Required
Add
© django-fobi example site 2014

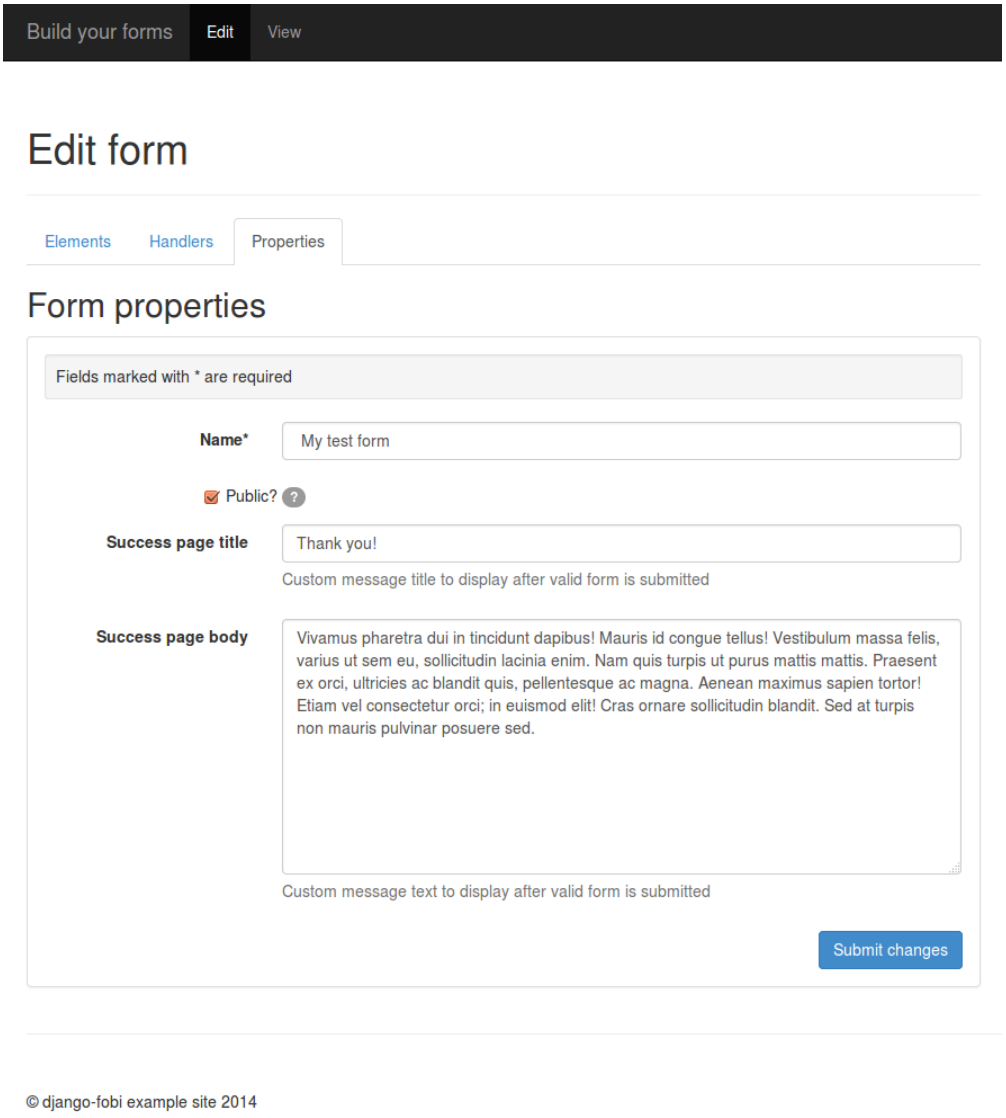

Build your forms
Edit
View
Edit form
Elements
Handlers
Properties
Add elements to your form
Fields marked with * are required
Choose form element to add
Name (Text)*
Edit
Delete
Description (Textarea)
Edit
Delete
Website (URL)
Edit
Delete
Save ordering
© django-fobi example site 2014

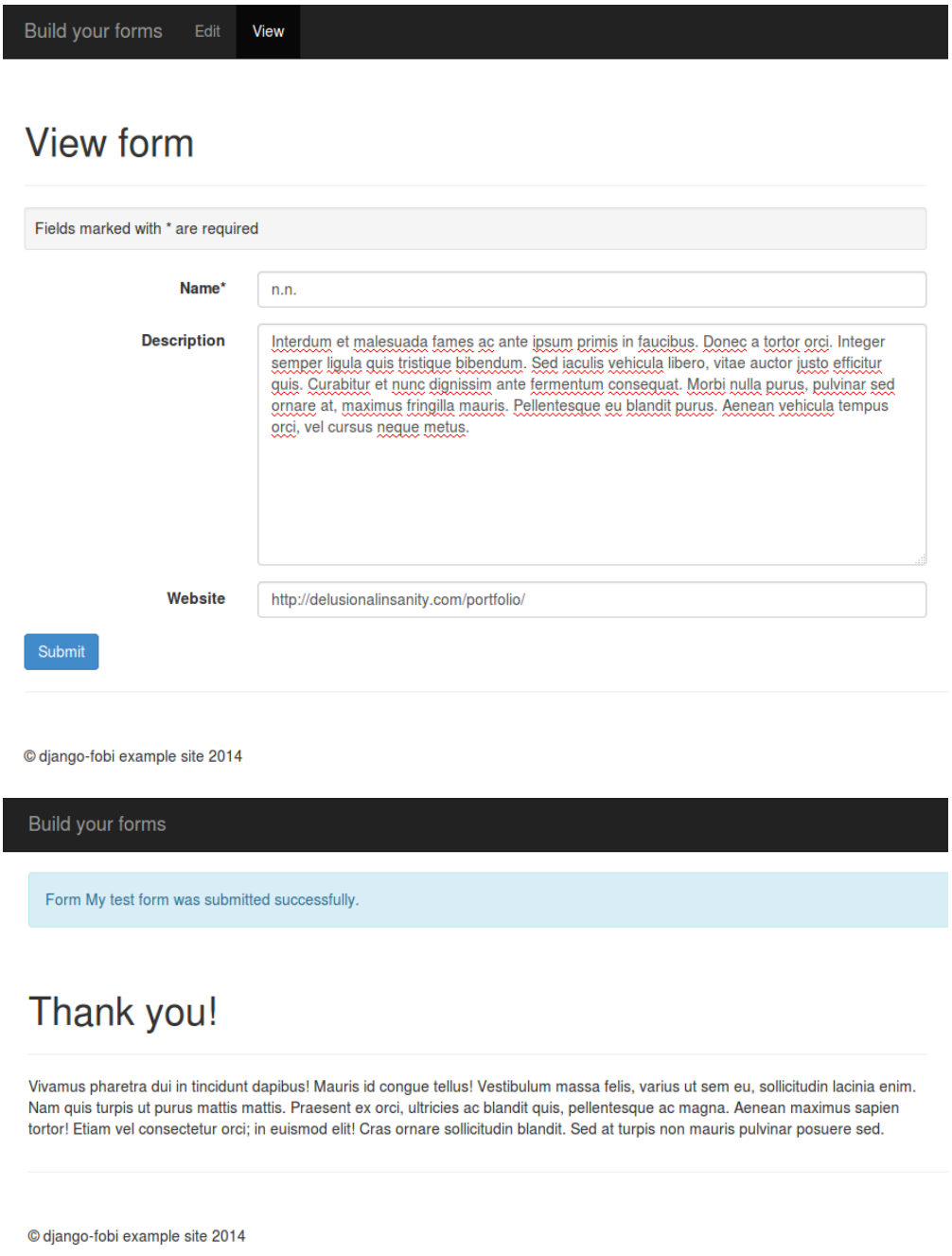
Form handlers

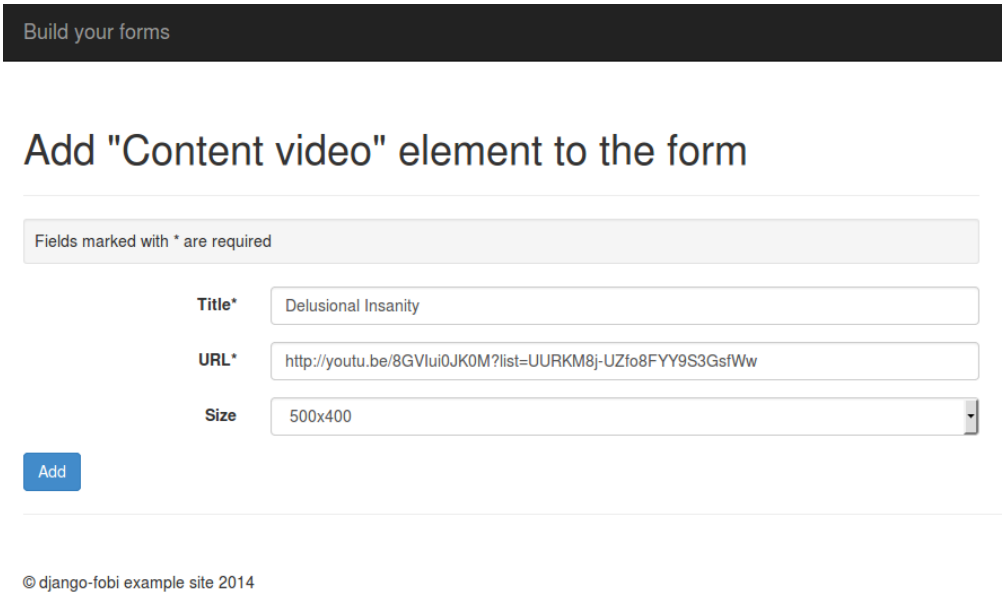


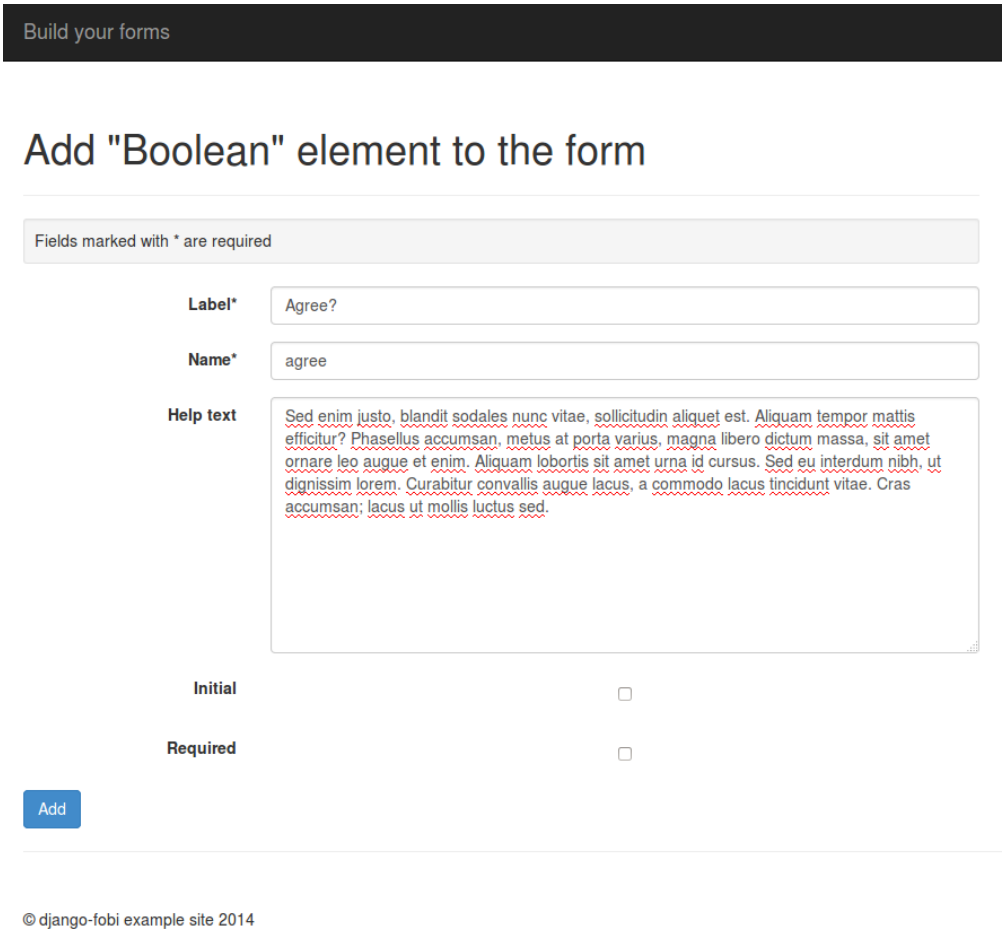

Build your forms
Add "Mail" handler to the form
Fields marked with * are required
From name*
From name
From email*
from@example.com
To name*
To name
To email*
to@example.com
Subject*
My form #1 mail
Body
Donec nec consequat velit, id mollis massa! Sed dapibus vehicula eleifend. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Morbi placerat justo eu diam commodo semper. Ut dapibus cursus mi id molestie? Aliquam nisi velit, placerat hendrerit cursus rhoncus, malesuada id ante. Nam ultricies orci et velit hendrerit, dignissim accumsan nibh rhoncus. Integer metus.
Add
© django-fobi example site 2014

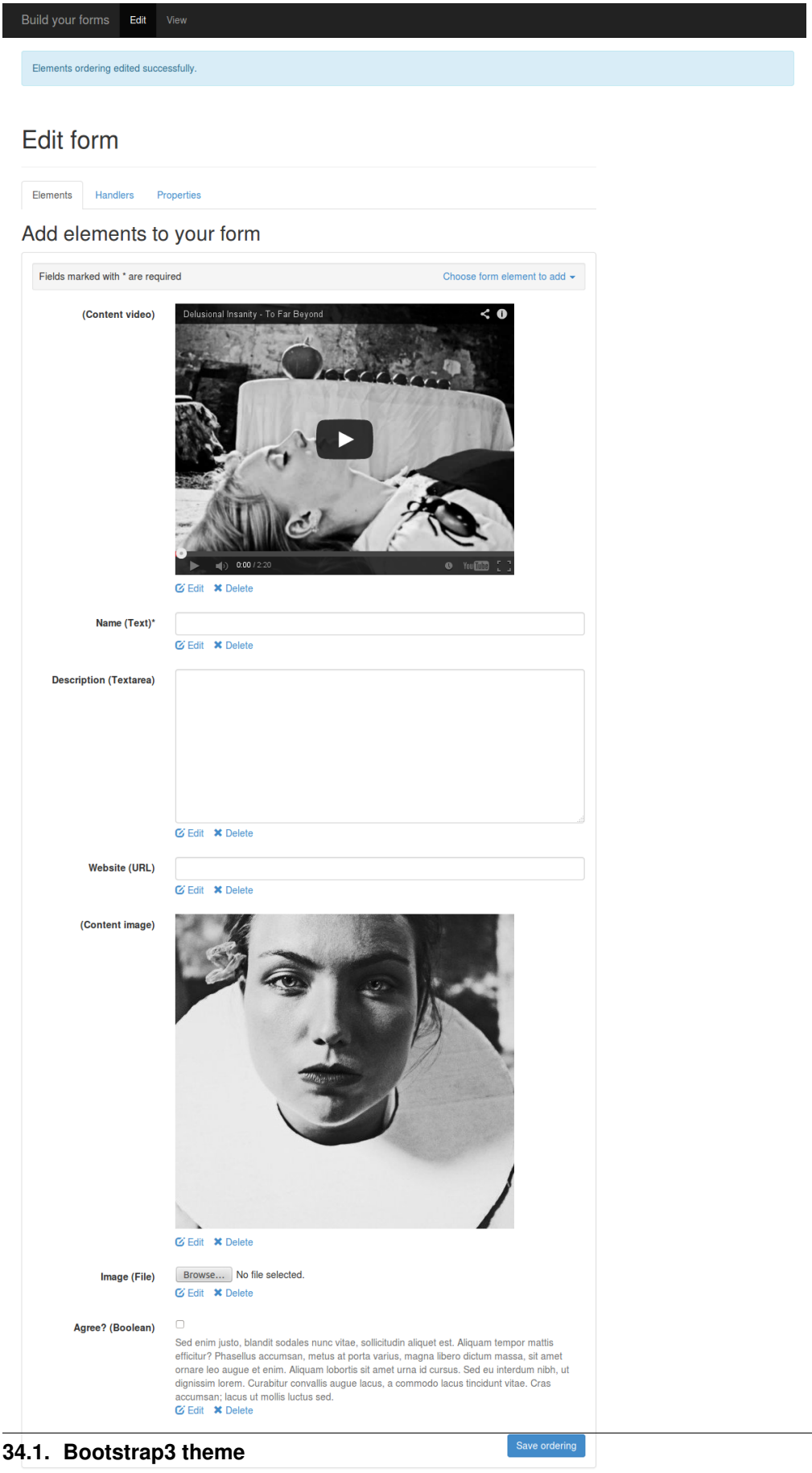

Build your forms
Edit
View
The form handler plugin "Mail" was added successfully.
Edit form
Elements
Handlers
Properties
Add handlers to your form
Choose form handler to add
Handler
Actions
DB store
Delete
View entries
Mail
Edit
Delete
© django-fobi example site 2014


Build your forms
Edit
View
Edit form
Elements
Handlers
Properties
Form properties
Fields marked with * are required
Name*
My test form
Public?
Success page title
Thank you!
Custom message title to display after valid form is submitted
Success page body
Vivamus pharetra dui in tincidunt dapibus! Mauris id congue tellus! Vestibulum massa felis, varius ut sem eu, sollicitudin lacinia enim. Nam quis turpis ut purus mattis mattis. Praesent ex orci, ultricies ac blandit quis, pellentesque ac magna. Aenean maximus sapien tortor! Etiam vel consectetur orci; in euismod elit! Cras ornare sollicitudin blandit. Sed at turpis non mauris pulvinar posuere sed.
Custom message text to display after valid form is submitted
Submit changes
© django-fobi example site 2014


Build your forms
Edit
View
View form
Fields marked with * are required
Name*
n.n.
Description
Interdum et malesuada fames ac ante ipsum primis in faucibus. Donec a tortor orci. Integer semper ligula quis tristique bibendum. Sed iaculis vehicula libero, vitae auctor justo efficitur quis. Curabitur et nunc dignissim ante fermentum consequat. Morbi nulla purus, pulvinar sed ornare at, maximus fringilla mauris. Pellentesque eu blandit purus. Aenean vehicula tempus orci, vel cursus neque metus.
Website
http://delusionalinsanity.com/portfolio/
Submit
© django-fobi example site 2014
Build your forms
Form My test form was submitted successfully.
Thank you!
Vivamus pharetra dui in tincidunt dapibus! Mauris id congue tellus! Vestibulum massa felis, varius ut sem eu, sollicitudin lacinia enim. Nam quis turpis ut purus mattis mattis. Praesent ex orci, ultricies ac blandit quis, pellentesque ac magna. Aenean maximus sapien tortor! Etiam vel consectetur orci; in euismod elit! Cras ornare sollicitudin blandit. Sed at turpis non mauris pulvinar posuere sed.
© django-fobi example site 2014


Build your forms
Add "Content video" element to the form
Fields marked with * are required
Title*
Delusional Insanity
URL*
http://youtu.be/8GVIui0JK0M?list=UURKM8j-UZfo8FYY9S3GsfWw
Size
500x400
Add
© django-fobi example site 2014


Build your forms
Add "Boolean" element to the form
Fields marked with * are required
Label*
Agree?
Name*
agree
Help text
Sed enim justo, blandit sodales nunc vitae, sollicitudin aliquet est. Aliquam tempor mattis efficitur? Phasellus accumsan, metus at porta varius, magna libero dictum massa, sit amet ornare leo augue et enim. Aliquam lobortis sit amet urna id cursus. Sed eu interdum nibh, ut dignissim lorem. Curabitur convallis augue lacus, a commodo lacus tincidunt vitae. Cras accumsan; lacus ut mollis luctus sed.
Initial
Required
Add
© django-fobi example site 2014


Build your forms
Edit
View
Elements ordering edited successfully.
Edit form
Elements
Handlers
Properties
Add elements to your form
Fields marked with * are required
Choose form element to add
(Content video)
Delusional Insanity - To Far Beyond
0:00 / 2:20
Edit
Delete
Name (Text)*
Edit
Delete
Description (Textarea)
Edit
Delete
Website (URL)
Edit
Delete
(Content image)
Edit
Delete
Image (File)
Browse...
No file selected.
Edit
Delete
Agree? (Boolean)
Sed enim justo, blandit sodales nunc vitae, sollicitudin aliquet est. Aliquam tempor mattis efficitur? Phasellus accumsan, metus at porta varius, magna libero dictum massa, sit amet ornare leo augue et enim. Aliquam lobortis sit amet urna id cursus. Sed eu interdum nibh, ut dignissim lorem. Curabitur convallis augue lacus, a commodo lacus tincidunt vitae. Cras accumsan; lacus ut mollis luctus sed.
Edit
Delete
Save ordering
© django-fobi example site 2014

Build your forms Edit View

View form

Fields marked with * are required

Name*

Description

Website

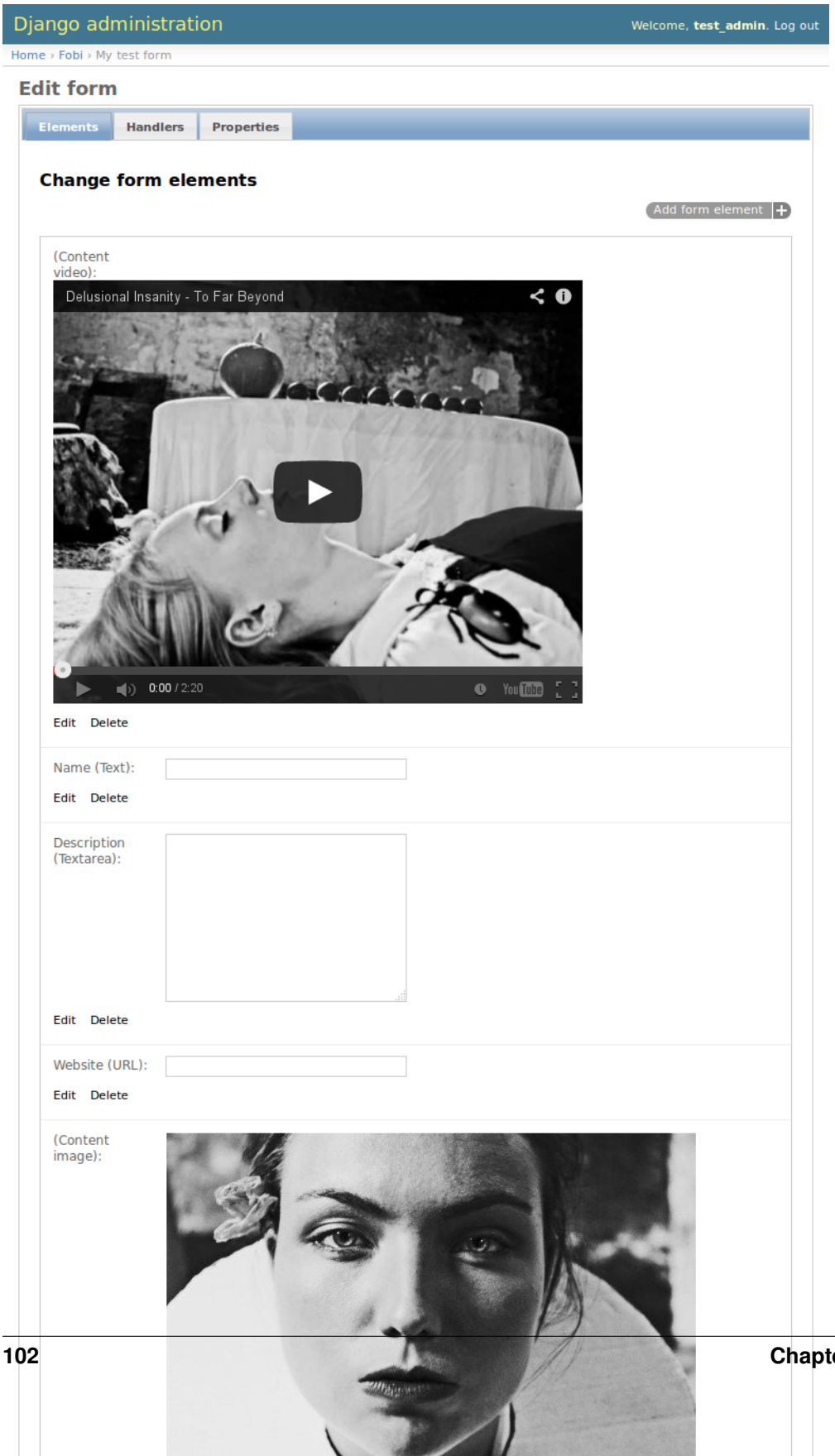
Image Browse… No file selected.

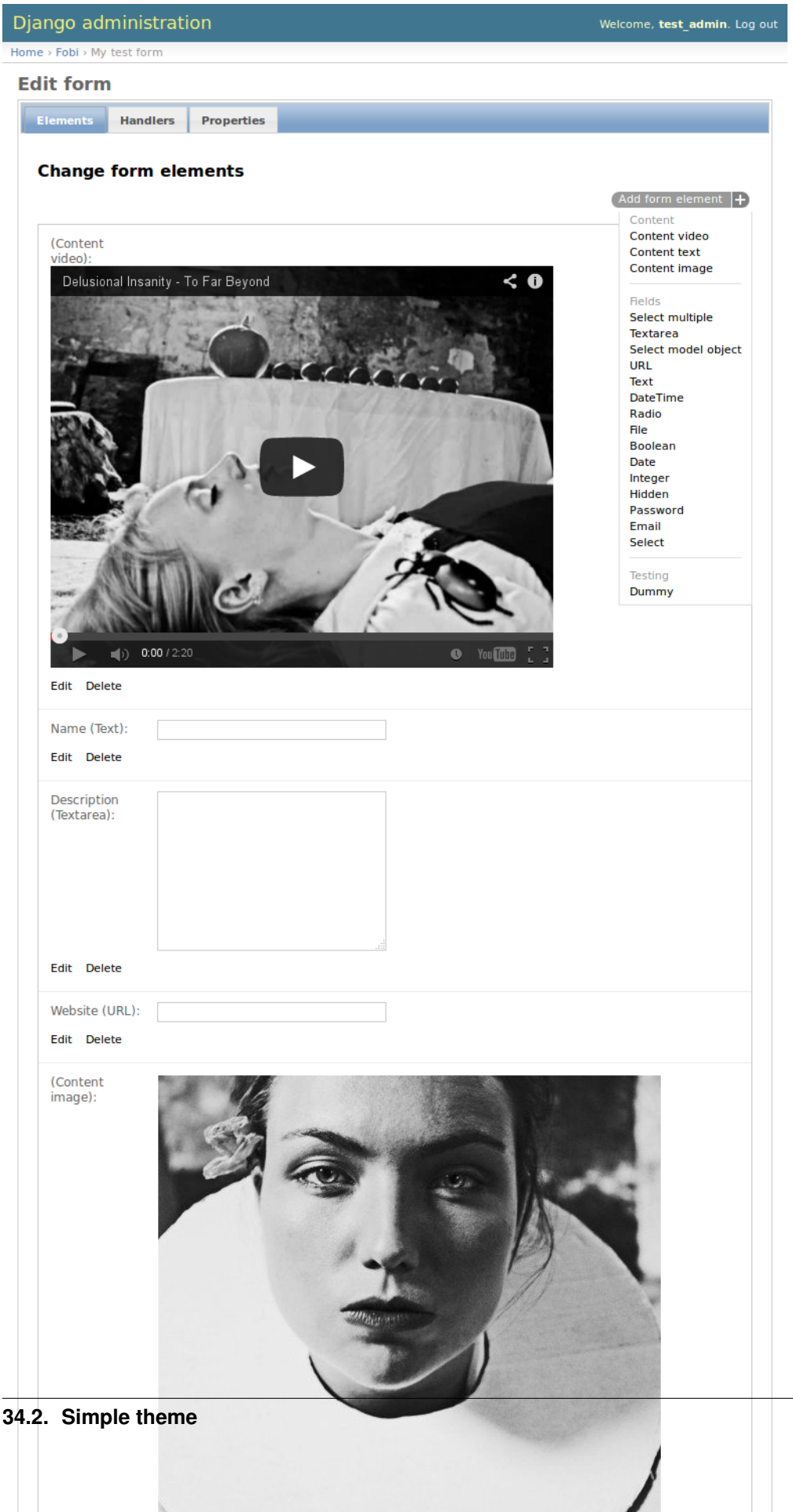
Agree?

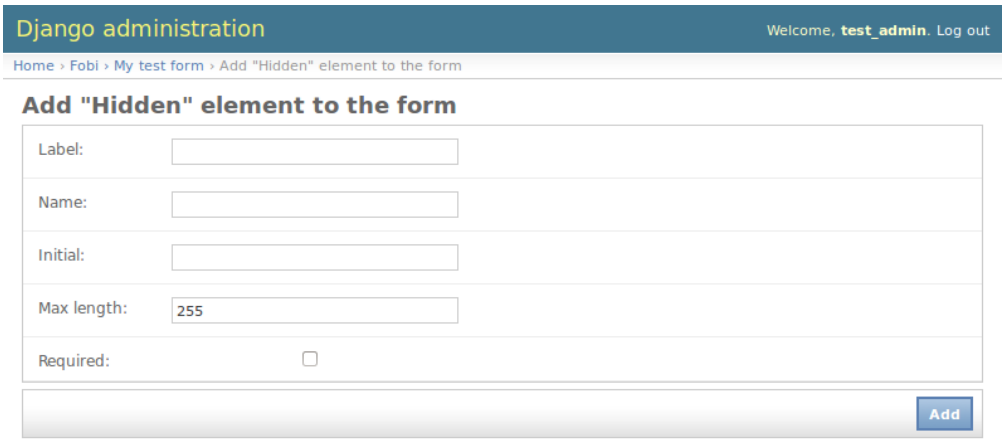
Sed enim justo, blandit sodales nunc vitae, sollicitudin aliquet est. Aliquam tempor mattis efficitur? Phasellus accumsan, metus at porta varius, magna libero dictum massa, sit amet ornare leo orci et enim. Aliquam lobortis sit amet urna id cursus. Sed eu interdum nibh, ut dignissim lorem. Curabitur convallis augue lacus, a commodo lacus tincidunt vitae. Cras accumsan; lacus ut mollis luctus sed.

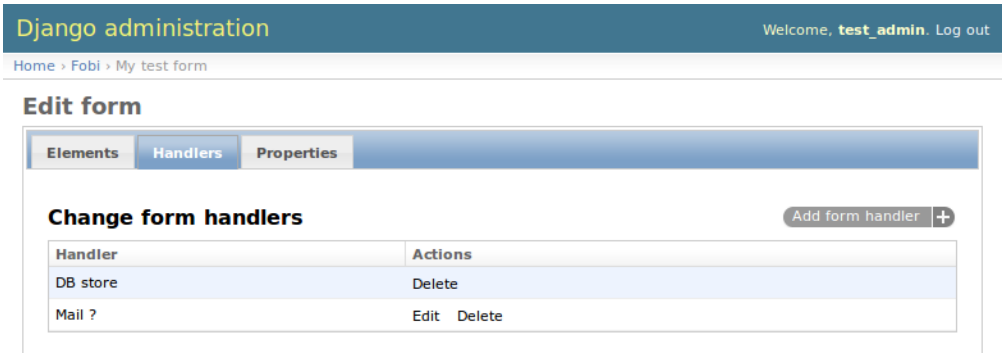
Simple theme

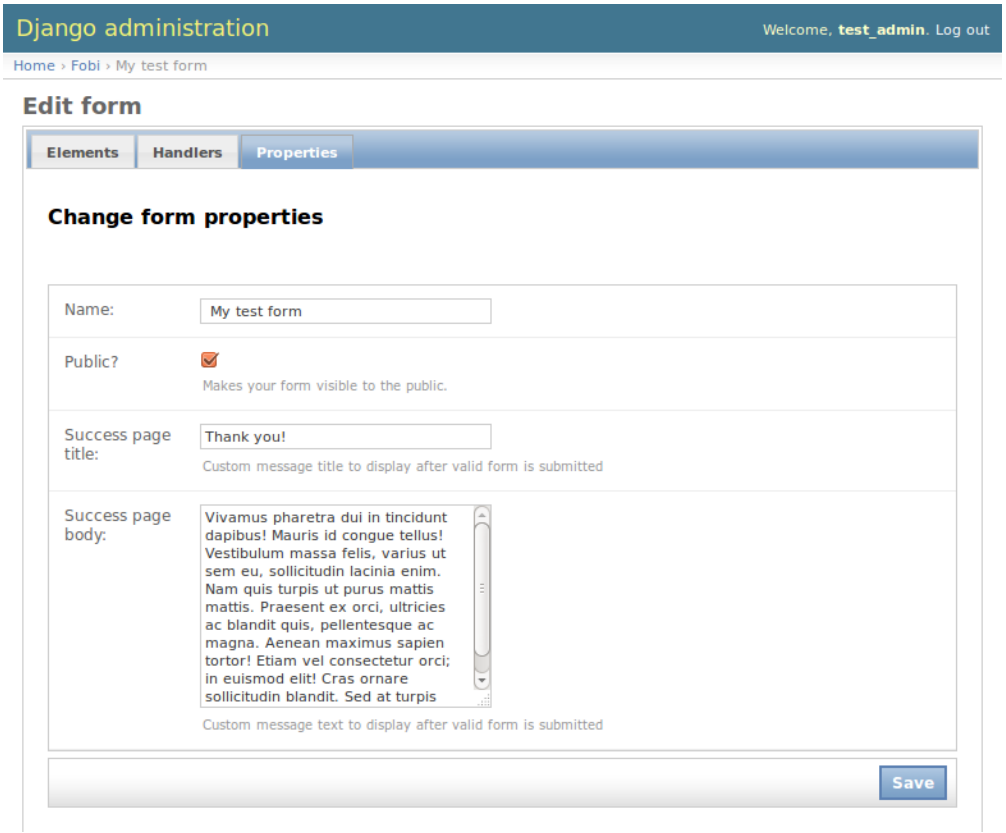
View/edit form




Django administration
Welcome, test_admin. Log out
Home › Fobi › My test form
Edit form
Elements
Handlers
Properties
Change form elements
Add form element
Content
Content video
Content text
Content image
Fields
Select multiple
Textarea
Select model object
URL
Text
DateTime
Radio
File
Boolean
Date
Integer
Hidden
Password
Email
Select
Testing
Dummy
(Content video):
Delusional Insanity - To Far Beyond
0:00 / 2:20
Edit Delete
Name (Text):
Edit Delete
Description (Textarea):
Edit Delete
Website (URL):
Edit Delete
(Content image):


Django administration
Welcome, test_admin. Log out
Home › Fobi › My test form › Add "Hidden" element to the form
Add "Hidden" element to the form
Label:
Name:
Initial:
Max length:
255
Required:
Add


Django administration
Welcome, test_admin. Log out
Home › Fobi › My test form
Edit form
Elements
Handlers
Properties
Change form handlers
Add form handler
Handler
Actions
DB store
Delete
Mail ?
Edit
Delete


Django administration
Welcome, test_admin. Log out
Home › Fobi › My test form
Edit form
Elements
Handlers
Properties
Change form properties
Name:
My test form
Public?
Makes your form visible to the public.
Success page title:
Thank you!
Custom message title to display after valid form is submitted
Success page body:
Vivamus pharetra dui in tincidunt dapibus! Mauris id congue tellus! Vestibulum massa felis, varius ut sem eu, sollicitudin lacinia enim. Nam quis turpis ut purus mattis mattis. Praesent ex orci, ultricies ac blandit quis, pellentesque ac magna. Aenean maximus sapien tortor! Etiam vel consectetur orci; in euismod elit! Cras ornare sollicitudin blandit. Sed at turpis
Custom message text to display after valid form is submitted
Save

View form

Name:

Description:

Website:

Image: Browse... No file selected.

Agree?

Sed enim justo, blandit sodales nunc vitae, sollicitudin aliquet est. Aliquam tempor mattis efficitur? Phasellus accumsan, metus at porta varius, magna libero dictum massa, sit amet ornare leo augue et enim. Aliquam lobortis sit amet urna id cursus. Sed eu interdum nibh, ut dignissim lorem. Curabitur convallis augue lacus, a commodo lacus tincidunt vitae. Cras accumsan; lacus ut mollis luctus sed.

Submit

CHAPTER 35

Documentation

Contents:

fobi package

Subpackages

fobi.contrib package

Subpackages

fobi.contrib.apps package

Subpackages

fobi.contrib.apps.djangocms_integration package

Submodules

fobi.contrib.apps.djangocms_integration.apps module

class fobi.contrib.apps.djangocms_integration.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_apps_djangocms_integration'

name = 'fobi.contrib.apps.djangocms_integration'

fobi.contrib.apps.djangocms_integration.cms_plugins module

fobi.contrib.apps.djangocms_integration.conf module

fobi.contrib.apps.djangocms_integration.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from fobi.contrib.apps.djangocms_integration conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.apps.djangocms_integration.defaults module

fobi.contrib.apps.djangocms_integration.helpers module

fobi.contrib.apps.djangocms_integration.helpers.**get_form_template_choices**()

Get the form template choices.

It's possible to provide theme templates per theme or just per project.

Return list

fobi.contrib.apps.djangocms_integration.helpers.**get_success_page_template_choices**()

Get success page template choices.

Return list

fobi.contrib.apps.djangocms_integration.models module

fobi.contrib.apps.djangocms_integration.settings module

- *WIDGET_FORM_SENT* (str): Name of the GET param indicating that form has been successfully sent.

Module contents

fobi.contrib.apps.feincms_integration package

Submodules

fobi.contrib.apps.feincms_integration.apps module

class fobi.contrib.apps.feincms_integration.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_apps_feincms_integration'

name = 'fobi.contrib.apps.feincms_integration'

fobi.contrib.apps.feincms_integration.conf module

fobi.contrib.apps.feincms_integration.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from fobi.contrib.apps.feincms_integration conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.apps.feincms_integration.defaults module

fobi.contrib.apps.feincms_integration.helpers module

fobi.contrib.apps.feincms_integration.helpers.**get_form_template_choices**()

Gets the form template choices.

It's possible to provide theme templates per theme or just per project.

Return list

fobi.contrib.apps.feincms_integration.helpers.**get_success_page_template_choices**()

Get success page template choices.

Return list

fobi.contrib.apps.feincms_integration.settings module

- *WIDGET_FORM_SENT_GET_PARAM* (str): Name of the GET param indicating that form has been successfully sent.

fobi.contrib.apps.feincms_integration.widgets module

class fobi.contrib.apps.feincms_integration.widgets.**FobiFormWidget**(**args*, ***kwargs*)

Bases: django.db.models.base.Model, *fobi.integration.processors.IntegrationProcessor*

Widget for to FeinCMS.

Property fobi.models.FormEntry form_entry Form entry to be rendered.

Property str template If given used for rendering the form.

class Meta

Meta class.

abstract = False

app_label = 'fobi'

FobiFormWidget.**can_redirect = True**

FobiFormWidget.**finalize**(*request*, *response*)

Finalize.

FobiFormWidget.**form_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

FobiFormWidget.**form_entry_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FobiFormWidget.**form_sent_get_param = 'sent'**

FobiFormWidget.**form_submit_button_text**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FobiFormWidget.**form_template_name**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FobiFormWidget.**form_title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FobiFormWidget.**get_form_template_name_display**(**moreargs*, ***morekwargs*)

FobiFormWidget.**get_success_page_template_name_display**(**moreargs*, ***morekwargs*)

FobiFormWidget.**hide_form_title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FobiFormWidget.**hide_success_page_title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FobiFormWidget.**process**(*request*, ***kwargs*)
This is where most of the form handling happens.

Parameters **request** (*django.http.HttpRequest*) –

Return django.http.HttpResponse | str

FobiFormWidget.**render**(***kwargs*)
Render.

FobiFormWidget.**success_page_template_name**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FobiFormWidget.**success_page_text**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FobiFormWidget.**success_page_title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

Module contents

fobi.contrib.apps.mezzanine_integration package

Submodules

fobi.contrib.apps.mezzanine_integration.admin module

fobi.contrib.apps.mezzanine_integration.apps module

class fobi.contrib.apps.mezzanine_integration.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_apps_mezzanine_integration'

name = 'fobi.contrib.apps.mezzanine_integration'

fobi.contrib.apps.mezzanine_integration.conf module

fobi.contrib.apps.mezzanine_integration.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from fobi.contrib.apps.mezzanine_integration conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.apps.mezzanine_integration.defaults module

fobi.contrib.apps.mezzanine_integration.helpers module

fobi.contrib.apps.mezzanine_integration.helpers.**get_form_template_choices**()

Gets the form template choices.

It's possible to provide theme templates per theme or just per project.

Return list

fobi.contrib.apps.mezzanine_integration.helpers.**get_success_page_template_choices**()

Get success page template choices.

Return list

fobi.contrib.apps.mezzanine_integration.models module

fobi.contrib.apps.mezzanine_integration.page_processors module

fobi.contrib.apps.mezzanine_integration.settings module

- *WIDGET_FORM_SENT_GET_PARAM* (str): Name of the GET param indicating that form has been successfully sent.

Module contents

Module contents

fobi.contrib.plugins package

Subpackages

fobi.contrib.plugins.form_elements package

Subpackages

fobi.contrib.plugins.form_elements.content package

Subpackages

fobi.contrib.plugins.form_elements.content.content_image package

Submodules

fobi.contrib.plugins.form_elements.content.content_image.apps module

class `fobi.contrib.plugins.form_elements.content.content_image.apps.`**`Config`**(*app_name*, *app_module*)

Bases: `django.apps.config.AppConfig`

Config.

`label` = 'fobi_contrib_plugins_form_elements_content_content_image'

`name` = 'fobi.contrib.plugins.form_elements.content.content_image'

fobi.contrib.plugins.form_elements.content.content_image.conf module

`fobi.contrib.plugins.form_elements.content.content_image.conf.`**`get_setting`**(*setting*, *override=None*)

Get setting.

Get a setting from `fobi.contrib.plugins.form_elements.content.content_image` conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **`setting`** – String with setting name

- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.content.content_image.defaults module

fobi.contrib.plugins.form_elements.content.content_image.fobi_form_elements module

class fobi.contrib.plugins.form_elements.content.content_image.fobi_form_elements.**ContentImag**

Bases: *fobi.base.FormElementPlugin*

Content image plugin.

clone_plugin_data(*entry*)

Clone plugin data.

Clone plugin data, which means we make a copy of the original image.

TODO: Perhaps rely more on data of form_element_entry?

delete_plugin_data()

Delete uploaded file.

form

alias of ContentImageForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

post_processor()

Post process data.

Always the same.

uid = 'content_image'

fobi.contrib.plugins.form_elements.content.content_image.forms module

class fobi.contrib.plugins.form_elements.content.content_image.forms.**ContentImageForm**(*data=Non files=Non auto_id=u pre- fix=None, ini- tial=None er- ror_class= 'django.fo la- bel_suffix= empty_pe field_orde use_requi*

Bases: django.forms.forms.Form, *fobi.base.BasePluginForm*

Form for ContentImagePlugin.

base_fields = OrderedDict([('file', <django.forms.fields.ImageField object at 0x7ff21ecdf290>), ('alt', <django.forms.f

declared_fields = OrderedDict([('file', <django.forms.fields.ImageField object at 0x7ff21ecdf290>), ('alt', <django.fo

media

plugin_data_fields = [('file', ''), ('alt', ''), ('fit_method', 'center'), ('size', '500x500')]

save_plugin_data(*request=None*)
Saving the plugin data and moving the file.

fobi.contrib.plugins.form_elements.content.content_image.helpers module

fobi.contrib.plugins.form_elements.content.content_image.helpers.**handle_uploaded_file**(*image_*
Handle uploaded file.

Parameters **image_file** (*django.core.files.uploadedfile.InMemoryUploadedFile*) –

Return string Path to the image (relative).

fobi.contrib.plugins.form_elements.content.content_image.helpers.**get_crop_filter**(*fit_method*)
Get crop filter.

fobi.contrib.plugins.form_elements.content.content_image.helpers.**delete_file**(*image_file*)
Delete file from disc.

fobi.contrib.plugins.form_elements.content.content_image.helpers.**ensure_unique_filename**(*des*
Makes sure filenames are never overwritten.

Parameters **destination** (*string*) –

Return string

fobi.contrib.plugins.form_elements.content.content_image.helpers.**clone_file**(*source_filename*, *relative_path=True*)
Clone the file.

Parameters **source_filename** (*string*) – Source filename.

Return string Filename of the cloned file.

fobi.contrib.plugins.form_elements.content.content_image.settings module

- FIT_METHOD_CROP_SMART (string)
- FIT_METHOD_CROP_CENTER (string)
- FIT_METHOD_CROP_SCALE (string)
- FIT_METHOD_FIT_WIDTH (string)
- FIT_METHOD_FIT_HEIGHT (string)
- DEFAULT_FIT_METHOD (string)
- FIT_METHODS_CHOICES (tuple)
- FIT_METHODS_CHOICES_WITH_EMPTY_OPTION (list)
- IMAGES_UPLOAD_DIR (string)

Module contents

fobi.contrib.plugins.form_elements.content.content_text package

Submodules

fobi.contrib.plugins.form_elements.content.content_text.apps module

class fobi.contrib.plugins.form_elements.content.content_text.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_content_content_text'

name = 'fobi.contrib.plugins.form_elements.content.content_text'

fobi.contrib.plugins.form_elements.content.content_text.conf module

fobi.contrib.plugins.form_elements.content.content_text.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from fobi.contrib.plugins.form_elements.content.content_text conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.content.content_text.defaults module

fobi.contrib.plugins.form_elements.content.content_text.fobi_form_elements module

class fobi.contrib.plugins.form_elements.content.content_text.fobi_form_elements.**ContentTextP**

Bases: *fobi.base.FormElementPlugin*

Content text plugin.

form

alias of ContentTextForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

post_processor()

Post process data.

Always the same.

uid = 'content_text'

fobi.contrib.plugins.form_elements.content.content_text.forms module

class fobi.contrib.plugins.form_elements.content.content_text.forms.**ContentTextForm**(*data=None*, *files=None*, *auto_id=u'id*, *prefix=None*, *initial=None*, *error_class=<*, *'django.form*, *label_suffix=N*, *empty_permi*, *field_order=*, *use_required*)

Bases: django.forms.forms.Form, *fobi.base.BasePluginForm*

Form for ContentTextPlugin.

base_fields = OrderedDict([('text', <django.forms.fields.CharField object at 0x7ff21ecdf1d0>)])

clean_text()
Clean text value.

declared_fields = OrderedDict([('text', <django.forms.fields.CharField object at 0x7ff21ecdf1d0>)])

media

plugin_data_fields = [('text', '')]

fobi.contrib.plugins.form_elements.content.content_text.settings module

Module contents

fobi.contrib.plugins.form_elements.content.content_video package

Submodules

fobi.contrib.plugins.form_elements.content.content_video.apps module

class fobi.contrib.plugins.form_elements.content.content_video.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_content_content_video'

name = 'fobi.contrib.plugins.form_elements.content.content_video'

fobi.contrib.plugins.form_elements.content.content_video.conf module

fobi.contrib.plugins.form_elements.content.content_video.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from `fobi.contrib.plugins.form_elements.content.content_video` conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.content.content_video.defaults module

fobi.contrib.plugins.form_elements.content.content_video.fobi_form_elements module

class fobi.contrib.plugins.form_elements.content.content_video.fobi_form_elements.**ContentVide**

Bases: *fobi.base.FormElementPlugin*

Content video plugin.

form
alias of ContentVideoForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)
Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

post_processor()
Process plugin data.

Always the same.

uid = 'content_video'

fobi.contrib.plugins.form_elements.content.content_video.forms module

class fobi.contrib.plugins.form_elements.content.content_video.forms.**ContentVideoForm**(*data=Non files=Non auto_id=u pre- fix=None, ini- tial=None er- ror_class= 'django.fo la- bel_suffix= empty_pe field_orde use_requi*

Bases: django.forms.forms.Form, *fobi.base.BasePluginForm*

Form for ContentVideoPlugin.

base_fields = OrderedDict([('title', <django.forms.fields.CharField object at 0x7ff21eccbf10>), ('url', <django.forms.f

declared_fields = OrderedDict([('title', <django.forms.fields.CharField object at 0x7ff21eccbf10>), ('url', <django.fo

media

plugin_data_fields = [('title', ''), ('url', ''), ('size', '500x400')]

fobi.contrib.plugins.form_elements.content.content_video.settings module

Module contents

Module contents

fobi.contrib.plugins.form_elements.fields package

Subpackages

fobi.contrib.plugins.form_elements.fields.boolean package

Submodules

fobi.contrib.plugins.form_elements.fields.boolean.apps module

class fobi.contrib.plugins.form_elements.fields.boolean.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_boolean'

name = 'fobi.contrib.plugins.form_elements.fields.boolean'

fobi.contrib.plugins.form_elements.fields.boolean.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.boolean.fobi_form_elements.**BooleanSelectPlugi**

Bases: *fobi.base.FormFieldPlugin*

Boolean select plugin.

form

alias of BooleanSelectForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'boolean'

fobi.contrib.plugins.form_elements.fields.boolean.forms module

Module contents

fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple package

Submodules

fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.apps module

class fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.apps.**Config**(*app_name*, *app_modul*

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_checkbox_select_multiple'

name = 'fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple'

fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.conf module

fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.conf.**get_setting**(*setting*, *override=Non*

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.defaults module

fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.fobi_form_elements.C

Bases: *fobi.base.FormFieldPlugin*

Checkbox select multiple field plugin.

form

alias of CheckboxSelectMultipleInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)

Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.

- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'checkbox_select_multiple'

fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.forms module

class fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.forms.**CheckboxSelect**

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for CheckboxSelectMultipleInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec66f90>), ('name', <django.for

clean_initial()
Validating the initial value.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec66f90>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('choices', ''), ('help_text', ''), ('initial', ''), ('required', False)]

fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.date package

Submodules

fobi.contrib.plugins.form_elements.fields.date.apps module

class fobi.contrib.plugins.form_elements.fields.date.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_date'

name = 'fobi.contrib.plugins.form_elements.fields.date'

fobi.contrib.plugins.form_elements.fields.date.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.date.fobi_form_elements.**DateInputPlugin**(*user=N*

Bases: *fobi.base.FormFieldPlugin*

Date field plugin.

form

alias of DateInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)

Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'date'

fobi.contrib.plugins.form_elements.fields.date.forms module

class fobi.contrib.plugins.form_elements.fields.date.forms.**DateInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for DateInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ee22f50>), ('name', <django.for

clean_initial()

Clean the initial value.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ee22f50>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('input_formats', ''), ('required', False)

fobi.contrib.plugins.form_elements.fields.date.widgets module

class fobi.contrib.plugins.form_elements.fields.date.widgets.**BaseDatePluginWidget**(*plugin*)

Bases: *fobi.base.FormElementPluginWidget*

Base date form element plugin widget.

plugin_uid = 'date'

Module contents

fobi.contrib.plugins.form_elements.fields.date_drop_down package

Submodules

fobi.contrib.plugins.form_elements.fields.date_drop_down.apps module

class fobi.contrib.plugins.form_elements.fields.date_drop_down.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_date_drop_down'

name = 'fobi.contrib.plugins.form_elements.fields.date_drop_down'

fobi.contrib.plugins.form_elements.fields.date_drop_down.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.date_drop_down.fobi_form_elements.**DateDropDow**

Bases: *fobi.base.FormFieldPlugin*

Date drop down field plugin.

form

alias of DateDropDownInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'date_drop_down'

fobi.contrib.plugins.form_elements.fields.date_drop_down.forms module

class fobi.contrib.plugins.form_elements.fields.date_drop_down.forms.**DateDropDownInputForm**(*da*
file
au
pr
fix
ini
tia
er-
ro
'dj
la-
be
em
fie
us

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for DateDropDownInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec664d0>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec664d0>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('year_min', ''), ('year_max', ''), ('initial', ''), ('inpu

Module contents

fobi.contrib.plugins.form_elements.fields.datetime package

Submodules

fobi.contrib.plugins.form_elements.fields.datetime.apps module

class fobi.contrib.plugins.form_elements.fields.datetime.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_datetime'

name = 'fobi.contrib.plugins.form_elements.fields.datetime'

fobi.contrib.plugins.form_elements.fields.datetime.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.datetime.fobi_form_elements.**DateTimeInputPlug**

Bases: *fobi.base.FormFieldPlugin*

DateTime field plugin.

form

alias of DateTimeInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)
Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of `fobi.models.FormEntry`.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'datetime'

fobi.contrib.plugins.form_elements.fields.datetime.forms module

class `fobi.contrib.plugins.form_elements.fields.datetime.forms.`**DateTimeInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s*, *prefix=None*, *initial=None*, *error_class=<class*, *'django.forms.uti*, *label_suffix=None*, *empty_permitted=*, *field_order=None*, *use_required_attr*)

Bases: `django.forms.forms.Form`, *fobi.base.BaseFormFieldPluginForm*

Form for `DateTimeInputPlugin`.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21edbdfd0>), ('name', <django.for

clean_initial()
Clean the initial value.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21edbdfd0>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('input_formats', ''), ('required', False)

fobi.contrib.plugins.form_elements.fields.datetime.widgets module

class `fobi.contrib.plugins.form_elements.fields.datetime.widgets.`**BaseDateTimePluginWidget**(*plug*
Bases: *fobi.base.FormElementPluginWidget*

Base datetime form element plugin widget.

plugin_uid = 'datetime'

Module contents

fobi.contrib.plugins.form_elements.fields.decimal package

Submodules

fobi.contrib.plugins.form_elements.fields.decimal.apps module

class fobi.contrib.plugins.form_elements.fields.decimal.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_decimal'

name = 'fobi.contrib.plugins.form_elements.fields.decimal'

fobi.contrib.plugins.form_elements.fields.decimal.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.decimal.fobi_form_elements.**DecimalInputPlugin**

Bases: *fobi.base.FormFieldPlugin*

Decimal input plugin.

form

alias of DecimalInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'decimal'

fobi.contrib.plugins.form_elements.fields.decimal.forms module

class fobi.contrib.plugins.form_elements.fields.decimal.forms.**DecimalInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.E*, *label_suffix=None*, *empty_permitted=Fa*, *field_order=None*, *use_required_attribu*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for DecimalInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21edbd710>), ('name', <django.for

clean()

Validating the values.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21edbd710>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('max_digits', ''), ('decimal_places', '')

Module contents

fobi.contrib.plugins.form_elements.fields.email package

Submodules

fobi.contrib.plugins.form_elements.fields.email.apps module

class fobi.contrib.plugins.form_elements.fields.email.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_email'

name = 'fobi.contrib.plugins.form_elements.fields.email'

fobi.contrib.plugins.form_elements.fields.email.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.email.fobi_form_elements.**EmailInputPlugin**(*user*

Bases: *fobi.base.FormFieldPlugin*

Email input plugin.

form

alias of EmailInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'email'

fobi.contrib.plugins.form_elements.fields.email.forms module

class fobi.contrib.plugins.form_elements.fields.email.forms.**EmailInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorL*
label_suffix=None, *empty_permitted=False*, *field_order=None*, *use_required_attribute=N*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for `EmailInputPlugin`.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21edbd0d0>), ('name', <django.fo

clean()

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21edbd0d0>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('max_length', '255'), ('required', False

Module contents

fobi.contrib.plugins.form_elements.fields.file package

Submodules

fobi.contrib.plugins.form_elements.fields.file.apps module

class fobi.contrib.plugins.form_elements.fields.file.apps.**Config**(*app_name*, *app_module*)

Bases: `django.apps.config.AppConfig`

Config.

label = 'fobi_contrib_plugins_form_elements_fields_file'

name = 'fobi.contrib.plugins.form_elements.fields.file'

fobi.contrib.plugins.form_elements.fields.file.conf module

fobi.contrib.plugins.form_elements.fields.file.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from `fobi.contrib.plugins.form_elements.fields.conf` module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.file.defaults module

fobi.contrib.plugins.form_elements.fields.file.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.file.fobi_form_elements.**FileInputPlugin**(*user=N*

Bases: *fobi.base.FormFieldPlugin*

File field plugin.

form

alias of `FileInputForm`

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)

Submit plugin form data/process file upload.

Handling the posted data for file plugin when form is submitted. This method affects the original form and that's why it returns it.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of `fobi.models.FormEntry`.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'file'

fobi.contrib.plugins.form_elements.fields.file.forms module

class `fobi.contrib.plugins.form_elements.fields.file.forms.`**FileInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: `django.forms.forms.Form`, *fobi.base.BaseFormFieldPluginForm*

Form for `FileInputPlugin`.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec50ad0>), ('name', <django.for

clean()

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec50ad0>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('max_length', '255'), ('required', Fals

fobi.contrib.plugins.form_elements.fields.file.settings module

- `FILES_UPLOAD_DIR` (string)

Module contents

fobi.contrib.plugins.form_elements.fields.float package

Submodules

fobi.contrib.plugins.form_elements.fields.float.apps module

class fobi.contrib.plugins.form_elements.fields.float.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_float'

name = 'fobi.contrib.plugins.form_elements.fields.float'

fobi.contrib.plugins.form_elements.fields.float.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.float.fobi_form_elements.**FloatInputPlugin**(*user*

Bases: *fobi.base.FormFieldPlugin*

Float input plugin.

form

alias of FloatInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'float'

fobi.contrib.plugins.form_elements.fields.float.forms module

class fobi.contrib.plugins.form_elements.fields.float.forms.**FloatInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorL*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=N*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for FloatInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec50490>), ('name', <django.for

clean()

Validating the values.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec50490>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('min_value', None), ('max_value', Non

Module contents

fobi.contrib.plugins.form_elements.fields.hidden package

Submodules

fobi.contrib.plugins.form_elements.fields.hidden.apps module

class fobi.contrib.plugins.form_elements.fields.hidden.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_hidden'

name = 'fobi.contrib.plugins.form_elements.fields.hidden'

fobi.contrib.plugins.form_elements.fields.hidden.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.hidden.fobi_form_elements.**HiddenInputPlugin**(*u*

Bases: *fobi.base.FormFieldPlugin*

Hidden field plugin.

form

alias of HiddenInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

is_hidden = True

name = <django.utils.functional.__proxy__ object>

uid = 'hidden'

fobi.contrib.plugins.form_elements.fields.hidden.forms module

class fobi.contrib.plugins.form_elements.fields.hidden.forms.**HiddenInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.Err*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for HiddenInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec44fd0>), ('name', <django.for

clean()

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec44fd0>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('initial', ''), ('max_length', '255'), ('required', False)]

Module contents

fobi.contrib.plugins.form_elements.fields.input package

Submodules

fobi.contrib.plugins.form_elements.fields.input.apps module

class fobi.contrib.plugins.form_elements.fields.input.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_input'

name = 'fobi.contrib.plugins.form_elements.fields.input'

fobi.contrib.plugins.form_elements.fields.input.constants module

fobi.contrib.plugins.form_elements.fields.input.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.input.fobi_form_elements.**InputPlugin**(*user=None*

Bases: *fobi.base.FormFieldPlugin*

Input field plugin.

form

alias of InputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'input'

fobi.contrib.plugins.form_elements.fields.input.forms module

class fobi.contrib.plugins.form_elements.fields.input.forms.**InputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for InputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec44210>), ('name', <django.for

clean()

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec44210>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('max_length', '255'), ('required', Fals

Module contents

fobi.contrib.plugins.form_elements.fields.integer package

Submodules

fobi.contrib.plugins.form_elements.fields.integer.apps module

class fobi.contrib.plugins.form_elements.fields.integer.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_integer'

name = 'fobi.contrib.plugins.form_elements.fields.integer'

fobi.contrib.plugins.form_elements.fields.integer.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.integer.fobi_form_elements.**IntegerInputPlugin**

Bases: *fobi.base.FormFieldPlugin*

Integer input plugin.

form

alias of IntegerInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'integer'

fobi.contrib.plugins.form_elements.fields.integer.forms module

class fobi.contrib.plugins.form_elements.fields.integer.forms.**IntegerInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.E*, *label_suffix=None*, *empty_permitted=Fa*, *field_order=None*, *use_required_attribu*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for IntegerInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecb7b90>), ('name', <django.for

clean()

Validating the values.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecb7b90>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('min_value', None), ('max_value', Non

Module contents

fobi.contrib.plugins.form_elements.fields.ip_address package

Submodules

fobi.contrib.plugins.form_elements.fields.ip_address.apps module

class fobi.contrib.plugins.form_elements.fields.ip_address.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_ip_address'

name = 'fobi.contrib.plugins.form_elements.fields.ip_address'

fobi.contrib.plugins.form_elements.fields.ip_address.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.ip_address.fobi_form_elements.**IPAddressInputP**

Bases: *fobi.base.FormFieldPlugin*

IP address field plugin.

form

alias of IPAddressInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'ip_address'

fobi.contrib.plugins.form_elements.fields.ip_address.forms module

Module contents

fobi.contrib.plugins.form_elements.fields.null_boolean package

Submodules

fobi.contrib.plugins.form_elements.fields.null_boolean.apps module

class fobi.contrib.plugins.form_elements.fields.null_boolean.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_null_boolean'

name = 'fobi.contrib.plugins.form_elements.fields.null_boolean'

fobi.contrib.plugins.form_elements.fields.null_boolean.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.null_boolean.fobi_form_elements.**NullBooleanSe**

Bases: *fobi.base.FormFieldPlugin*

Null boolean select plugin.

form

alias of NullBooleanSelectForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'null_boolean'

fobi.contrib.plugins.form_elements.fields.null_boolean.forms module

Module contents

fobi.contrib.plugins.form_elements.fields.password package

Submodules

fobi.contrib.plugins.form_elements.fields.password.apps module

class fobi.contrib.plugins.form_elements.fields.password.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_password'

name = 'fobi.contrib.plugins.form_elements.fields.password'

fobi.contrib.plugins.form_elements.fields.password.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.password.fobi_form_elements.**PasswordInputPlug**

Bases: *fobi.base.FormFieldPlugin*

Password field plugin.

form

alias of PasswordInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'password'

fobi.contrib.plugins.form_elements.fields.password.forms module

class fobi.contrib.plugins.form_elements.fields.password.forms.**PasswordInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s*, *prefix=None*, *initial=None*, *error_class=<class*, *'django.forms.uti*, *label_suffix=None*, *empty_permitted=*, *field_order=None*, *use_required_attr*)

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for PasswordInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecaa950>), ('name', <django.for

clean()

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecaa950>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('max_length', '255'), ('required', False

Module contents

fobi.contrib.plugins.form_elements.fields.radio package

Submodules

fobi.contrib.plugins.form_elements.fields.radio.apps module

class fobi.contrib.plugins.form_elements.fields.radio.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_radio'

name = 'fobi.contrib.plugins.form_elements.fields.radio'

fobi.contrib.plugins.form_elements.fields.radio.conf module

fobi.contrib.plugins.form_elements.fields.radio.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.radio* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name

- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.radio.defaults module

fobi.contrib.plugins.form_elements.fields.radio.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.radio.fobi_form_elements.**RadioInputPlugin**(*user*

Bases: *fobi.base.FormFieldPlugin*

Radio field plugin.

form
alias of RadioInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)
Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)
Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'radio'

fobi.contrib.plugins.form_elements.fields.radio.forms module

class fobi.contrib.plugins.form_elements.fields.radio.forms.**RadioInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorL*
label_suffix=None, *empty_permitted=False*, *field_order=None*, *use_required_attribute=N*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for RadioInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecaa310>), ('name', <django.for

clean_initial()
Validating the initial value.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecaa310>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('choices', ''), ('help_text', ''), ('initial', ''), ('required', False)]

fobi.contrib.plugins.form_elements.fields.radio.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.range_select package

Submodules

fobi.contrib.plugins.form_elements.fields.range_select.apps module

class fobi.contrib.plugins.form_elements.fields.range_select.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_range_select'

name = 'fobi.contrib.plugins.form_elements.fields.range_select'

fobi.contrib.plugins.form_elements.fields.range_select.conf module

fobi.contrib.plugins.form_elements.fields.range_select.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.range_select* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.range_select.defaults module

fobi.contrib.plugins.form_elements.fields.range_select.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.range_select.fobi_form_elements.**RangeSelectIn**

Bases: *fobi.base.FormFieldPlugin*

Range select input plugin.

form

alias of RangeSelectInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'range_select'

fobi.contrib.plugins.form_elements.fields.range_select.forms module

class fobi.contrib.plugins.form_elements.fields.range_select.forms.**RangeSelectInputForm**(*data=N files=N auto_id pre- fix=No ini- tial=N er- ror_cla 'django la- bel_suf empty_ field_o use_re*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for RangeSelectInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec9fa10>), ('name', <django.form

clean()

Validating the values.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec9fa10>), ('name', <django

media

plugin_data_fields = [('label', ''), ('name', ''), ('min_value', 0), ('max_value', 100), ('step', 1), ('help_text', ''), ('init

fobi.contrib.plugins.form_elements.fields.range_select.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.regex package

Submodules

fobi.contrib.plugins.form_elements.fields.regex.apps module

class fobi.contrib.plugins.form_elements.fields.regex.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_regex'

name = 'fobi.contrib.plugins.form_elements.fields.regex'

fobi.contrib.plugins.form_elements.fields.regex.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.regex.fobi_form_elements.**RegexInputPlugin**(*user*

Bases: *fobi.base.FormFieldPlugin*

Regex field plugin.

form

alias of RegexInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'regex'

fobi.contrib.plugins.form_elements.fields.regex.forms module

class fobi.contrib.plugins.form_elements.fields.regex.forms.**RegexInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorL*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=N*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for RegexInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec9f2d0>), ('name', <django.for

clean()

Validation.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec9f2d0>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('regex', ''), ('max_length', '255'), ('req

Module contents

fobi.contrib.plugins.form_elements.fields.select package

Submodules

fobi.contrib.plugins.form_elements.fields.select.apps module

class fobi.contrib.plugins.form_elements.fields.select.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_select'

name = 'fobi.contrib.plugins.form_elements.fields.select'

fobi.contrib.plugins.form_elements.fields.select.conf module

fobi.contrib.plugins.form_elements.fields.select.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.select* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.select.defaults module

fobi.contrib.plugins.form_elements.fields.select.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.select.fobi_form_elements.**SelectInputPlugin**(*u*

Bases: *fobi.base.FormFieldPlugin*

Select field plugin.

form

alias of SelectInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)

Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'select'

fobi.contrib.plugins.form_elements.fields.select.forms module

class fobi.contrib.plugins.form_elements.fields.select.forms.**SelectInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.Err*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for SelectInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec95c90>), ('name', <django.for

clean_initial()
Validating the initial value.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec95c90>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('choices', ''), ('help_text', ''), ('initial', ''), ('required', False)]

fobi.contrib.plugins.form_elements.fields.select.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.select_model_object package

Submodules

fobi.contrib.plugins.form_elements.fields.select_model_object.apps module

class fobi.contrib.plugins.form_elements.fields.select_model_object.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_select_model_object'

name = 'fobi.contrib.plugins.form_elements.fields.select_model_object'

fobi.contrib.plugins.form_elements.fields.select_model_object.conf module

fobi.contrib.plugins.form_elements.fields.select_model_object.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.select_model_object* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.select_model_object.defaults module

fobi.contrib.plugins.form_elements.fields.select_model_object.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.select_model_object.fobi_form_elements.**Select**

Bases: *fobi.base.FormFieldPlugin*

Select model object field plugin.

form
alias of SelectModelObjectInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)
Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)
Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'select_model_object'

fobi.contrib.plugins.form_elements.fields.select_model_object.forms module

class fobi.contrib.plugins.form_elements.fields.select_model_object.forms.**SelectModelObjectIn**

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for SelectModelObjectPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec95790>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec95790>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('model', ''), ('help_text', ''), ('initial', ''), ('required', False)]

fobi.contrib.plugins.form_elements.fields.select_model_object.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.select_mptt_model_object package

Submodules

fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.apps module

class fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.apps.**Config**(*app_name*, *app_modul*

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_select_mptt_model_object'

name = 'fobi.contrib.plugins.form_elements.fields.select_mptt_model_object'

fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.conf module

fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.conf.**get_setting**(*setting*, *override=Non*

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.select_mptt_model_object* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.defaults module

fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.fobi_form_elements module

fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.forms module

class fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.forms.**SelectMPTTMode**

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for SelectMPTTModelObjectPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21a9b5790>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21a9b5790>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('model', ''), ('help_text', ''), ('initial', ''), ('required', False)]

fobi.contrib.plugins.form_elements.fields.select_mptt_model_object.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.select_multiple package

Submodules

fobi.contrib.plugins.form_elements.fields.select_multiple.apps module

class fobi.contrib.plugins.form_elements.fields.select_multiple.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_select_multiple'

name = 'fobi.contrib.plugins.form_elements.fields.select_multiple'

fobi.contrib.plugins.form_elements.fields.select_multiple.conf module

fobi.contrib.plugins.form_elements.fields.select_multiple.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.select_multiple* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.select_multiple.defaults module

fobi.contrib.plugins.form_elements.fields.select_multiple.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.select_multiple.fobi_form_elements.**SelectMult**

Bases: *fobi.base.FormFieldPlugin*

Select multiple field plugin.

form

alias of SelectMultipleInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)

Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of `fobi.models.FormEntry`.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'select_multiple'

fobi.contrib.plugins.form_elements.fields.select_multiple.forms module

class `fobi.contrib.plugins.form_elements.fields.select_multiple.forms.`**SelectMultipleInputForm**

Bases: `django.forms.forms.Form`, *fobi.base.BaseFormFieldPluginForm*

Form for `SelectMultipleInputPlugin`.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec95190>), ('name', <django.for

clean_initial()
Validating the initial value.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec95190>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('choices', ''), ('help_text', ''), ('initial', ''), ('required', False)]

fobi.contrib.plugins.form_elements.fields.select_multiple.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects package

Submodules

fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.apps module

class `fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.apps.`**Config**(*app*
app

Bases: `django.apps.config.AppConfig`

Config.

label = 'fobi_contrib_plugins_form_elements_fields_select_multiple_model_objects'

name = 'fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects'

fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.conf module

fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.conf.**get_setting**(*se*
ov
ri

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.defaults module

fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.fobi_form_eleme

Bases: *fobi.base.FormFieldPlugin*

Select multiple model objects field plugin.

form
alias of SelectMultipleModelObjectsInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)
Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)
Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'select_multiple_model_objects'

fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.forms module

class fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.forms.**SelectMul**

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for SelectMultipleModelObjectsPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec87590>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec87590>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('model', ''), ('help_text', ''), ('initial', ''), ('required', False)]

fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects package

Submodules

fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.apps module

class fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.apps.**Confi**

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_select_multiple_mptt_model_objects'

name = 'fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects'

fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.conf module

fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.conf.**get_sett**

Get a setting from *fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_ objects* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.defaults module

fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.fobi_form_elements module

fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.forms module

class fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.forms.**Sele**

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for SelectMultipleMPTTModelObjectsPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21b303a90>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21b303a90>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('model', ''), ('help_text', ''), ('initial', ''), ('required', False)]

fobi.contrib.plugins.form_elements.fields.select_multiple_mptt_model_objects.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max package

Submodules

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.apps module

class fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.apps.**Config**(*app_name*, *app_modul*

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_select_multiple_with_max'

name = 'fobi.contrib.plugins.form_elements.fields.select_multiple_with_max'

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.conf module

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.conf.**get_setting**(*setting*, *override=Non*

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.select_multiple_with_max* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.defaults module

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.fields module

class fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.fields.**MultipleChoic**

Bases: django.forms.fields.MultipleChoiceField

Multiple choice with max field.

validate(*value*)
Validate.

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.fobi_form_elements.**S**

Bases: *fobi.base.FormFieldPlugin*

Select multiple with max field plugin.

form
alias of SelectMultipleWithMaxInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)
Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)
Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'select_multiple_with_max'

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.forms module

class fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.forms.**SelectMultiple**

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for SelectMultipleWithMaxInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec87a90>), ('name', <django.for

clean_initial()
Validating the initial value.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec87a90>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('choices', ''), ('help_text', ''), ('initial', ''), ('required', False), ('max

fobi.contrib.plugins.form_elements.fields.select_multiple_with_max.settings module

Module contents

fobi.contrib.plugins.form_elements.fields.slider package

Submodules

fobi.contrib.plugins.form_elements.fields.slider.apps module

class fobi.contrib.plugins.form_elements.fields.slider.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_slider'

name = 'fobi.contrib.plugins.form_elements.fields.slider'

fobi.contrib.plugins.form_elements.fields.slider.conf module

fobi.contrib.plugins.form_elements.fields.slider.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from *fobi.contrib.plugins.form_elements.fields.slider* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name.
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.fields.slider.constants module

fobi.contrib.plugins.form_elements.fields.slider.defaults module

fobi.contrib.plugins.form_elements.fields.slider.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.slider.fobi_form_elements.**SliderInputPlugin**(*u*

Bases: *fobi.base.FormFieldPlugin*

Slider field plugin.

form

alias of SliderInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

html_classes = ['slider']

name = <django.utils.functional.__proxy__ object>

uid = 'slider'

fobi.contrib.plugins.form_elements.fields.slider.forms module

class fobi.contrib.plugins.form_elements.fields.slider.forms.**SliderInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.Err*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for SliderInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec7b6d0>), ('name', <django.for

clean()

Validating the values.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec7b6d0>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('initial', 50), ('min_value', 0), ('max_value', 100), ('step', 1), ('toolti

fobi.contrib.plugins.form_elements.fields.slider.helpers module

fobi.contrib.plugins.form_elements.fields.slider.helpers.**generate_ticks**(*choices*, *empty_labels=False*)

Generate ticks.

Parameters

- **choices** (*iterable*) – Iterable of tuples or lists:
- **empty_labels** (*bool*) –

Return dict

fobi.contrib.plugins.form_elements.fields.slider.settings module

fobi.contrib.plugins.form_elements.fields.slider.widgets module

class fobi.contrib.plugins.form_elements.fields.slider.widgets.**BaseSliderPluginWidget**(*plugin*)

Bases: *fobi.base.FormElementPluginWidget*

Base date form element plugin widget.

html_classes = ['slider']

plugin_uid = 'slider'

Module contents

fobi.contrib.plugins.form_elements.fields.slug package

Submodules

fobi.contrib.plugins.form_elements.fields.slug.apps module

class fobi.contrib.plugins.form_elements.fields.slug.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_slug'

name = 'fobi.contrib.plugins.form_elements.fields.slug'

fobi.contrib.plugins.form_elements.fields.slug.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.slug.fobi_form_elements.**SlugInputPlugin**(*user=N*

Bases: *fobi.base.FormFieldPlugin*

Slug field plugin.

form

alias of SlugInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'slug'

fobi.contrib.plugins.form_elements.fields.slug.forms module

class fobi.contrib.plugins.form_elements.fields.slug.forms.**SlugInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for SlugInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec7b110>), ('name', <django.for

clean()

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ec7b110>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('max_length', '255'), ('required', False

Module contents

fobi.contrib.plugins.form_elements.fields.text package

Submodules

fobi.contrib.plugins.form_elements.fields.text.apps module

class fobi.contrib.plugins.form_elements.fields.text.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_text'

name = 'fobi.contrib.plugins.form_elements.fields.text'

fobi.contrib.plugins.form_elements.fields.text.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.text.fobi_form_elements.**TextInputPlugin**(*user=N*

Bases: *fobi.base.FormFieldPlugin*

Text field plugin.

form

alias of TextInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'text'

fobi.contrib.plugins.form_elements.fields.text.forms module

class fobi.contrib.plugins.form_elements.fields.text.forms.**TextInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for TextInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21eceeb10>), ('name', <django.for

clean()

Validation.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21eceeb10>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('max_length', '255'), ('required', False

Module contents

fobi.contrib.plugins.form_elements.fields.textarea package

Submodules

fobi.contrib.plugins.form_elements.fields.textarea.apps module

class fobi.contrib.plugins.form_elements.fields.textarea.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_textarea'

name = 'fobi.contrib.plugins.form_elements.fields.textarea'

fobi.contrib.plugins.form_elements.fields.textarea.fobi_form_elements module

fobi.contrib.plugins.form_elements.fields.textarea.forms module

class fobi.contrib.plugins.form_elements.fields.textarea.forms.**TextareaForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.Error*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for TextareaPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecee610>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecee610>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('required', False), ('placeholder', '')]

Module contents

fobi.contrib.plugins.form_elements.fields.time package

Submodules

fobi.contrib.plugins.form_elements.fields.time.apps module

class fobi.contrib.plugins.form_elements.fields.time.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_time'

name = 'fobi.contrib.plugins.form_elements.fields.time'

fobi.contrib.plugins.form_elements.fields.time.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.time.fobi_form_elements.**TimeInputPlugin**(*user=N*

Bases: *fobi.base.FormFieldPlugin*

Time field plugin.

form

alias of TimeInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)
Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)
Submit plugin form data/process.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of `fobi.models.FormEntry`.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

uid = 'time'

fobi.contrib.plugins.form_elements.fields.time.forms module

class `fobi.contrib.plugins.form_elements.fields.time.forms.`**TimeInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)
Bases: `django.forms.forms.Form`, *fobi.base.BaseFormFieldPluginForm*

Form for `TimeInputPlugin`.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecee150>), ('name', <django.for

clean_initial()
Clean the initial value.

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecee150>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('input_formats', ''), ('required', False)

Module contents

fobi.contrib.plugins.form_elements.fields.url package

Submodules

fobi.contrib.plugins.form_elements.fields.url.apps module

class fobi.contrib.plugins.form_elements.fields.url.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_fields_url'

name = 'fobi.contrib.plugins.form_elements.fields.url'

fobi.contrib.plugins.form_elements.fields.url.fobi_form_elements module

class fobi.contrib.plugins.form_elements.fields.url.fobi_form_elements.**URLInputPlugin**(*user=Non*

Bases: *fobi.base.FormFieldPlugin*

URL input plugin.

form

alias of URLInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'url'

fobi.contrib.plugins.form_elements.fields.url.forms module

class fobi.contrib.plugins.form_elements.fields.url.forms.**URLInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for URLPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecdfb50>), ('name', <django.for

clean()

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21ecdfb50>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('initial', ''), ('max_length', '255'), ('required', False

Module contents

Module contents

fobi.contrib.plugins.form_elements.security package

Subpackages

fobi.contrib.plugins.form_elements.security.captcha package

Submodules

fobi.contrib.plugins.form_elements.security.captcha.apps module

class fobi.contrib.plugins.form_elements.security.captcha.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_security_captcha'

name = 'fobi.contrib.plugins.form_elements.security.captcha'

fobi.contrib.plugins.form_elements.security.captcha.fobi_form_elements module

class fobi.contrib.plugins.form_elements.security.captcha.fobi_form_elements.**CaptchaInputPlug**

Bases: *fobi.base.FormElementPlugin*

Captcha field plugin.

form

alias of CaptchaInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'captcha'

fobi.contrib.plugins.form_elements.security.captcha.forms module

class fobi.contrib.plugins.form_elements.security.captcha.forms.**CaptchaInputForm**(*data=None*, *files=None*, *auto_id=u'id_%s*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.uti*, *label_suffix=None*, *empty_permitted=*, *field_order=None*, *use_required_attr*)

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for CaptchaInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21b1ac590>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21b1ac590>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('required', True)]

Module contents

fobi.contrib.plugins.form_elements.security.honeypot package

Submodules

fobi.contrib.plugins.form_elements.security.honeypot.apps module

class fobi.contrib.plugins.form_elements.security.honeypot.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_security_honeypot'

name = 'fobi.contrib.plugins.form_elements.security.honeypot'

fobi.contrib.plugins.form_elements.security.honeypot.conf module

fobi.contrib.plugins.form_elements.security.honeypot.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from fobi.contrib.plugins.form_elements.security.honeypot conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_elements.security.honeypot.defaults module

fobi.contrib.plugins.form_elements.security.honeypot.fields module

class fobi.contrib.plugins.form_elements.security.honeypot.fields.**HoneypotField**(*max_length=None*, *min_length=None*, *strip=True*, **args*, ***kwargs*)

Bases: django.forms.fields.CharField

clean(*value*)
Check that honeypot value remained the same.

default_error_messages = {'invalid': <django.utils.functional.__proxy__ object at 0x7ff21edae350>}

widget
alias of HiddenInput

fobi.contrib.plugins.form_elements.security.honeypot.fobi_form_elements module

class fobi.contrib.plugins.form_elements.security.honeypot.fobi_form_elements.**HoneypotInputPl**

Bases: *fobi.base.FormElementPlugin*

Honeypot field plugin.

form
alias of HoneypotInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)
Get form field instances.

group = <django.utils.functional.__proxy__ object>

is_hidden = True

name = <django.utils.functional.__proxy__ object>

uid = 'honeypot'

fobi.contrib.plugins.form_elements.security.honeypot.forms module

class fobi.contrib.plugins.form_elements.security.honeypot.forms.**HoneypotInputForm**(*data=None*, *files=None*, *auto_id=u'id_* *pre-* *fix=None*, *ini-* *tial=None*, *er-* *ror_class=<cl* *'django.forms.* *la-* *bel_suffix=No* *empty_permitt* *field_order=N* *use_required_*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for HoneypotInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21edaef50>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff21edaef50>), ('name', <djang

media

plugin_data_fields = [('label', ''), ('name', ''), ('initial', ''), ('max_length', '255'), ('required', True)]

fobi.contrib.plugins.form_elements.security.honeypot.settings module

- HONEYPOT_VALUE (string)

Module contents

fobi.contrib.plugins.form_elements.security.recaptcha package

Submodules

fobi.contrib.plugins.form_elements.security.recaptcha.apps module

class fobi.contrib.plugins.form_elements.security.recaptcha.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_security_recaptcha'

name = 'fobi.contrib.plugins.form_elements.security.recaptcha'

fobi.contrib.plugins.form_elements.security.recaptcha.fobi_form_elements module

class fobi.contrib.plugins.form_elements.security.recaptcha.fobi_form_elements.**ReCaptchaInput**

Bases: *fobi.base.FormElementPlugin*

ReCaptcha field plugin.

form

alias of ReCaptchaInputForm

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

uid = 'recaptcha'

fobi.contrib.plugins.form_elements.security.recaptcha.forms module

class fobi.contrib.plugins.form_elements.security.recaptcha.forms.**ReCaptchaInputForm**(*data=None*, *files=None*, *auto_id=u'*, *prefix=None*, *initial=None*, *error_class=*, *'django.for*, *label_suffix=*, *empty_perm*, *field_order=*, *use_require*

Bases: django.forms.forms.Form, *fobi.base.BaseFormFieldPluginForm*

Form for ReCaptchaInputPlugin.

base_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff219592590>), ('name', <django.for

declared_fields = OrderedDict([('label', <django.forms.fields.CharField object at 0x7ff219592590>), ('name', <djan

media

plugin_data_fields = [('label', ''), ('name', ''), ('help_text', ''), ('required', True)]

Module contents

Module contents

fobi.contrib.plugins.form_elements.test package

Subpackages

fobi.contrib.plugins.form_elements.test.dummy package

Submodules

fobi.contrib.plugins.form_elements.test.dummy.apps module

class fobi.contrib.plugins.form_elements.test.dummy.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_elements_test_dummy'

name = 'fobi.contrib.plugins.form_elements.test.dummy'

fobi.contrib.plugins.form_elements.test.dummy.fobi_form_elements module

class fobi.contrib.plugins.form_elements.test.dummy.fobi_form_elements.**DummyPlugin**(*user=None*)

Bases: *fobi.base.FormElementPlugin*

Dummy plugin.

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)

Get form field instances.

group = <django.utils.functional.__proxy__ object>

name = <django.utils.functional.__proxy__ object>

post_processor()

Post process data.

Always the same.

uid = 'dummy'

fobi.contrib.plugins.form_elements.test.dummy.widgets module

class fobi.contrib.plugins.form_elements.test.dummy.widgets.**BaseDummyPluginWidget**(*plugin*)

Bases: *fobi.base.FormElementPluginWidget*

Base dummy form element plugin widget.

plugin_uid = 'dummy'

Module contents

Module contents

Module contents

fobi.contrib.plugins.form_handlers package

Subpackages

fobi.contrib.plugins.form_handlers.db_store package

Subpackages

fobi.contrib.plugins.form_handlers.db_store.migrations package

Submodules

fobi.contrib.plugins.form_handlers.db_store.migrations.0001_initial module

class fobi.contrib.plugins.form_handlers.db_store.migrations.0001_initial.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0002_auto_20150912_1744'), (u'auth', u'__first__')]

operations = [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'form_data_headers', <django.db.

fobi.contrib.plugins.form_handlers.db_store.migrations.0002_savedformwizarddataentry module

class fobi.contrib.plugins.form_handlers.db_store.migrations.0002_savedformwizarddataentry.**Mi**

Bases: django.db.migrations.migration.Migration

dependencies = [(u'auth', u'__first__'), (u'fobi', u'0010_formwizardhandler'), (u'fobi_contrib_plugins_form_handlers

operations = [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'form_data_headers', <django.db.

Module contents

fobi.contrib.plugins.form_handlers.db_store.urls package

Submodules

fobi.contrib.plugins.form_handlers.db_store.urls.form_handlers module

fobi.contrib.plugins.form_handlers.db_store.urls.form_wizard_handlers module

Module contents

Submodules

fobi.contrib.plugins.form_handlers.db_store.admin module

class fobi.contrib.plugins.form_handlers.db_store.admin.**SavedFormDataEntryAdmin**(*model*, *admin_site*)

Bases: fobi.contrib.plugins.form_handlers.db_store.admin.BaseSavedFormDataEntryAdmin

Saved form data entry admin.

class **Meta**

Meta class.

app_label = <django.utils.functional.__proxy__ object>

SavedFormDataEntryAdmin.**actions = ['export_data']**

SavedFormDataEntryAdmin.**fieldsets = ((None, {'fields': ('form_entry', 'user')}), (<django.utils.functional.__pro**

SavedFormDataEntryAdmin.**list_display** = ('form_entry', 'user', 'formatted_saved_data', 'created')

SavedFormDataEntryAdmin.**list_filter** = ('form_entry', 'user')

SavedFormDataEntryAdmin.**media**

SavedFormDataEntryAdmin.**only_args** = ['form_entry']

SavedFormDataEntryAdmin.**readonly_fields** = ('created', 'formatted_saved_data')

class fobi.contrib.plugins.form_handlers.db_store.admin.**SavedFormWizardDataEntryAdmin**(*model*, *admin_site*)

Bases: fobi.contrib.plugins.form_handlers.db_store.admin.BaseSavedFormDataEntryAdmin

Saved form wizard data entry admin.

class Meta

Meta class.

app_label = <django.utils.functional.__proxy__ object>

SavedFormWizardDataEntryAdmin.**actions** = ['export_data']

SavedFormWizardDataEntryAdmin.**fieldsets** = ((None, {'fields': ('form_wizard_entry', 'user')}), (<django.utils.

SavedFormWizardDataEntryAdmin.**list_display** = ('form_wizard_entry', 'user', 'formatted_saved_data', 'crea

SavedFormWizardDataEntryAdmin.**list_filter** = ('form_wizard_entry', 'user')

SavedFormWizardDataEntryAdmin.**media**

SavedFormWizardDataEntryAdmin.**only_args** = ['form_wizard_entry']

SavedFormWizardDataEntryAdmin.**readonly_fields** = ('created', 'formatted_saved_data')

fobi.contrib.plugins.form_handlers.db_store.apps module

class fobi.contrib.plugins.form_handlers.db_store.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_handlers_db_store'

name = 'fobi.contrib.plugins.form_handlers.db_store'

fobi.contrib.plugins.form_handlers.db_store.conf module

fobi.contrib.plugins.form_handlers.db_store.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from fobi.contrib.plugins.form_handlers.db_store conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_handlers.db_store.defaults module

fobi.contrib.plugins.form_handlers.db_store.fobi_form_handlers module

class fobi.contrib.plugins.form_handlers.db_store.fobi_form_handlers.**DBStoreHandlerPlugin**(*user*

Bases: *fobi.base.FormHandlerPlugin*

DB store form handler plugin.

Can be used only once per form.

allow_multiple = False

custom_actions(*form_entry*, *request=None*)
Custom actions.

Adding a link to view the saved form entries.

Return iterable

name = <django.utils.functional.__proxy__ object>

run(*form_entry*, *request*, *form*, *form_element_entries=None*)
Run.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –
- **form_element_entries** (*iterable*) – Iterable of fobi.models.FormElementEntry objects.

uid = 'db_store'

class fobi.contrib.plugins.form_handlers.db_store.fobi_form_handlers.**DBStoreWizardHandlerPlug**

Bases: *fobi.base.FormWizardHandlerPlugin*

DB store form wizard handler plugin.

Can be used only once per form.

allow_multiple = False

custom_actions(*form_wizard_entry*, *request=None*)
Custom actions.

Adding a link to view the saved form entries.

Return iterable

name = <django.utils.functional.__proxy__ object>

run(*form_wizard_entry*, *request*, *form_list*, *form_wizard*, *form_element_entries=None*)
Run.

Parameters

- **form_wizard_entry** (fobi.models.FormWizardEntry) – Instance of *fobi.models.FormWizardEntry*.
- **request** (*django.http.HttpRequest*) –
- **form_list** (*list*) – List of django.forms.Form instances.

- **form_wizard** (fobi.wizard.views.dynamic.DynamicWizardView) – Instance of *fobi.wizard.views.dynamic.DynamicWizardView*.
- **form_element_entries** (*iterable*) – Iterable of fobi.models.FormElementEntry objects.

uid = 'db_store'

fobi.contrib.plugins.form_handlers.db_store.helpers module

class fobi.contrib.plugins.form_handlers.db_store.helpers.**DataExporter**(*queryset*, *only_args*)

Bases: object

Exporting the data.

export_to_csv()
Export data to CSV.

export_to_xls()
Export data to XLS.

graceful_export()
Export data into XLS/CSV depending on what is available.

fobi.contrib.plugins.form_handlers.db_store.models module

class fobi.contrib.plugins.form_handlers.db_store.models.**AbstractSavedFormDataEntry**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

Abstract saved form data entry.

class Meta
Bases: object

Meta class.

abstract = False

AbstractSavedFormDataEntry.**created**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

AbstractSavedFormDataEntry.**form_data_headers**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

AbstractSavedFormDataEntry.**formatted_saved_data**()
Shows the formatted saved data records.

Return string

AbstractSavedFormDataEntry.**get_next_by_created**(**moreargs*, ***morekwargs*)

AbstractSavedFormDataEntry.**get_previous_by_created**(**moreargs*, ***morekwargs*)

AbstractSavedFormDataEntry.**saved_data**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

AbstractSavedFormDataEntry.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

AbstractSavedFormDataEntry.**user_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class fobi.contrib.plugins.form_handlers.db_store.models.**SavedFormDataEntry**(**args*, ***kwargs*)
Bases: *fobi.contrib.plugins.form_handlers.db_store.models.AbstractSavedFormDataEntry*

Saved form data.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception SavedFormDataEntry.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

SavedFormDataEntry.**form_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

SavedFormDataEntry.**form_entry_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

SavedFormDataEntry.**get_next_by_created**(**moreargs*, ***morekwargs*)

SavedFormDataEntry.**get_previous_by_created**(**moreargs*, ***morekwargs*)

SavedFormDataEntry.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

SavedFormDataEntry.**objects = <django.db.models.manager.Manager object>**

SavedFormDataEntry.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

class fobi.contrib.plugins.form_handlers.db_store.models.**SavedFormWizardDataEntry**(**args*, ***kwargs*)
Bases: *fobi.contrib.plugins.form_handlers.db_store.models.AbstractSavedFormDataEntry*

Saved form data.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception SavedFormWizardDataEntry.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

SavedFormWizardDataEntry.**form_wizard_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

SavedFormWizardDataEntry.**form_wizard_entry_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

SavedFormWizardDataEntry.**get_next_by_created**(**moreargs*, ***morekwargs*)

SavedFormWizardDataEntry.**get_previous_by_created**(**moreargs*, ***morekwargs*)

SavedFormWizardDataEntry.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

SavedFormWizardDataEntry.**objects = <django.db.models.manager.Manager object>**

SavedFormWizardDataEntry.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

fobi.contrib.plugins.form_handlers.db_store.settings module

- CSV_DELIMITER (string)
- CSV_QUOTECHAR (string)

fobi.contrib.plugins.form_handlers.db_store.views module

fobi.contrib.plugins.form_handlers.db_store.views.**view_saved_form_data_entries**(*request*, **args*, ***kwargs*)

View saved form data entries.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_id** (*int*) – Form ID.
- **theme** (*fobi.base.BaseTheme*) – Subclass of fobi.base.BaseTheme.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.contrib.plugins.form_handlers.db_store.views.**export_saved_form_data_entries**(*request*, **args*, ***kwargs*)

Export saved form data entries.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_id** (*int*) – Form ID.
- **theme** (*fobi.base.BaseTheme*) – Subclass of fobi.base.BaseTheme.

Return django.http.HttpResponse

fobi.contrib.plugins.form_handlers.db_store.views.**view_saved_form_wizard_data_entries**(*reques* **args*, ***kwa*

View saved form wizard data entries.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_wizard_entry_id** (*int*) – Form ID.
- **theme** (*fobi.base.BaseTheme*) – Subclass of fobi.base.BaseTheme.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.contrib.plugins.form_handlers.db_store.views.**export_saved_form_wizard_data_entries**(*req* **ar* ***k*

Export saved form wizard data entries.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_wizard_entry_id** (*int*) – Form ID.
- **theme** (*fobi.base.BaseTheme*) – Subclass of fobi.base.BaseTheme.

Return django.http.HttpResponse

fobi.contrib.plugins.form_handlers.db_store.widgets module

class fobi.contrib.plugins.form_handlers.db_store.widgets.**BaseDbStorePluginWidget**(*plugin*)

Bases: *fobi.base.FormHandlerPluginWidget*

Base dummy form element plugin widget.

export_entries_icon_class = 'glyphicon glyphicon-export'

plugin_uid = 'db_store'

view_entries_icon_class = 'glyphicon glyphicon-list'

view_saved_form_data_entries_template_name = 'db_store/view_saved_form_data_entries.html'

Module contents

fobi.contrib.plugins.form_handlers.http_repost package

Submodules

fobi.contrib.plugins.form_handlers.http_repost.apps module

class fobi.contrib.plugins.form_handlers.http_repost.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_handlers_http_repost'

name = 'fobi.contrib.plugins.form_handlers.http_repost'

fobi.contrib.plugins.form_handlers.http_repost.fobi_form_handlers module

class fobi.contrib.plugins.form_handlers.http_repost.fobi_form_handlers.**HTTPRepostHandlerPlug**

Bases: *fobi.base.FormHandlerPlugin*

HTTP repost handler plugin.

Makes a HTTP repost to a given endpoint. Should be executed before db_store plugin.

form

alias of HTTPRepostForm

name = <django.utils.functional.__proxy__ object>

plugin_data_repr()

Human readable representation of plugin data.

Return string

run(*form_entry*, *request*, *form*, *form_element_entries=None*)

Run.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –
- **form_element_entries** (*iterable*) – Iterable of fobi.models.FormElementEntry objects.

uid = 'http_repost'

class fobi.contrib.plugins.form_handlers.http_repost.fobi_form_handlers.**HTTPRepostWizardHandl**

Bases: *fobi.base.FormWizardHandlerPlugin*

HTTP repost wizard handler plugin.

Makes a HTTP repost to a given endpoint. Should be executed before db_store plugin.

form

alias of HTTPRepostForm

name = <django.utils.functional.__proxy__ object>

plugin_data_repr()

Human readable representation of plugin data.

Return string

run (*form_wizard_entry*, *request*, *form_list*, *form_wizard*, *form_element_entries=None*)
Run.

Parameters

- **form_wizard_entry** (fobi.models.FormWizardEntry) – Instance of *fobi.models.FormWizardEntry*.
- **request** (*django.http.HttpRequest*) –
- **form_list** (*list*) – List of `django.forms.Form` instances.
- **form_wizard** (fobi.wizard.views.dynamic.DynamicWizardView) – Instance of *fobi.wizard.views.dynamic.DynamicWizardView*.
- **form_element_entries** (*iterable*) – Iterable of `fobi.models.FormElementEntry` objects.

uid = 'http_repost'

fobi.contrib.plugins.form_handlers.http_repost.forms module

class `fobi.contrib.plugins.form_handlers.http_repost.forms.`**HTTPRepostForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorLis*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=No*

Bases: `django.forms.forms.Form`, *fobi.base.BasePluginForm*

Form for `HTTPRepostPlugin`.

base_fields = OrderedDict([('endpoint_url', <django.forms.fields.URLField object at 0x7ff21eccbd10>)])

declared_fields = OrderedDict([('endpoint_url', <django.forms.fields.URLField object at 0x7ff21eccbd10>)])

media

plugin_data_fields = [('endpoint_url', '')]

Module contents

fobi.contrib.plugins.form_handlers.mail package

Submodules

fobi.contrib.plugins.form_handlers.mail.apps module

class fobi.contrib.plugins.form_handlers.mail.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_handlers_mail'

name = 'fobi.contrib.plugins.form_handlers.mail'

fobi.contrib.plugins.form_handlers.mail.conf module

fobi.contrib.plugins.form_handlers.mail.conf.**get_setting**(*setting*, *override=None*)

Get setting.

Get a setting from fobi.contrib.plugins.form_handlers.mail conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.contrib.plugins.form_handlers.mail.defaults module

fobi.contrib.plugins.form_handlers.mail.fields module

class fobi.contrib.plugins.form_handlers.mail.fields.**MultiEmailField**(*required=True*, *widget=None*, *label=None*, *initial=None*, *help_text=u''*, *error_messages=None*, *show_hidden_initial=False*, *validators=[]*, *localize=False*, *disabled=False*, *label_suffix=None*)

Bases: django.forms.fields.Field

MultiEmailField.

code = 'invalid'

message = <django.utils.functional.__proxy__ object>

to_python(*value*)

Normalize data to a list of strings.

validate(*value*)
Check if value consists only of valid emails.

widget
alias of `MultiEmailWidget`

fobi.contrib.plugins.form_handlers.mail.fobi_form_handlers module

class `fobi.contrib.plugins.form_handlers.mail.fobi_form_handlers.`**MailHandlerPlugin**(*user=None*)
Bases: *fobi.base.FormHandlerPlugin*

Mail handler plugin.

Sends emails to the person specified. Should be executed before `db_store` and `http_repost` plugins.

form
alias of `MailForm`

name = <django.utils.functional.__proxy__ object>

plugin_data_repr()
Human readable representation of plugin data.

Return string

run(*form_entry*, *request*, *form*, *form_element_entries=None*)
Run.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of `fobi.models.FormEntry`.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –
- **form_element_entries** (*iterable*) – Iterable of `fobi.models.FormElementEntry` objects.

uid = 'mail'

class `fobi.contrib.plugins.form_handlers.mail.fobi_form_handlers.`**MailWizardHandlerPlugin**(*user=*
Bases: *fobi.base.FormWizardHandlerPlugin*

Mail wizard handler plugin.

Sends emails to the person specified. Should be executed before `db_store` and `http_repost` plugins.

form
alias of `MailForm`

name = <django.utils.functional.__proxy__ object>

plugin_data_repr()
Human readable representation of plugin data.

Return string

run(*form_wizard_entry*, *request*, *form_list*, *form_wizard*, *form_element_entries=None*)
Run.

Parameters

- **form_wizard_entry** (fobi.models.FormWizardEntry) – Instance of *fobi.models.FormWizardEntry*.
- **request** (*django.http.HttpRequest*) –

- **form_list** (*list*) – List of `django.forms.Form` instances.
- **form_wizard** (fobi.wizard.views.dynamic.DynamicWizardView) – Instance of *fobi.wizard.views.dynamic.DynamicWizardView*.
- **form_element_entries** (*iterable*) – Iterable of `fobi.models.FormElementEntry` objects.

uid = 'mail'

fobi.contrib.plugins.form_handlers.mail.forms module

class `fobi.contrib.plugins.form_handlers.mail.forms.`**MailForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: `django.forms.forms.Form`, *fobi.base.BasePluginForm*

Form for `BooleanSelectPlugin`.

base_fields = OrderedDict([('from_name', <django.forms.fields.CharField object at 0x7ff21eccb810>), ('from_email',

declared_fields = OrderedDict([('from_name', <django.forms.fields.CharField object at 0x7ff21eccb810>), ('from_e

media

plugin_data_fields = [('from_name', ''), ('from_email', ''), ('to_name', ''), ('to_email', ''), ('subject', ''), ('body', ''

fobi.contrib.plugins.form_handlers.mail.helpers module

`fobi.contrib.plugins.form_handlers.mail.helpers.`**send_mail**(*subject*, *message*, *from_email*, *recipient_list*, *fail_silently=False*, *auth_user=None*, *auth_password=None*, *connection=None*, *html_message=None*, *attachments=None*)

Send email.

Easy wrapper for sending a single message to a recipient list. All members of the recipient list will see the other recipients in the 'To' field.

If auth_user is None, the EMAIL_HOST_USER setting is used. If auth_password is None, the EMAIL_HOST_PASSWORD setting is used.

Note: The API for this method is frozen. New code wanting to extend the functionality should use the EmailMessage class directly.

fobi.contrib.plugins.form_handlers.mail.settings module

fobi.contrib.plugins.form_handlers.mail.widgets module

class fobi.contrib.plugins.form_handlers.mail.widgets.**MultiEmailWidget**(*attrs=None*)

Bases: django.forms.widgets.Textarea

Multi email widget.

is_hidden = False

media

prep_value(*value*)

Prepare value before effectively render widget

render(*name*, *value*, *attrs=None*)

Render.

Module contents

Module contents

fobi.contrib.plugins.form_importers package

Subpackages

fobi.contrib.plugins.form_importers.mailchimp_importer package

Submodules

fobi.contrib.plugins.form_importers.mailchimp_importer.apps module

class fobi.contrib.plugins.form_importers.mailchimp_importer.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_plugins_form_importers_mailchimp_importer'

name = 'fobi.contrib.plugins.form_importers.mailchimp_importer'

fobi.contrib.plugins.form_importers.mailchimp_importer.fobi_form_importers module

class fobi.contrib.plugins.form_importers.mailchimp_importer.fobi_form_importers.**MailChimpImp**

Bases: *fobi.form_importers.BaseFormImporter*

MailChimp data importer.

extract_field_properties(*field_data*)

Extract field properties.

Handle choices differently as we know what the mailchimp format is.

field_properties_mapping = {'initial': 'default', 'name': 'tag', 'required': 'req', 'choices': 'choices', 'help_text': '

field_type_prop_name = 'field_type'

fields_mapping = {'url': 'url', 'radio': 'radio', 'zip': 'text', 'dropdown': 'select', 'date': 'date', 'text': 'text', 'address

name = <django.utils.functional.__proxy__ object>

position_prop_name = 'order'

templates = ['mailchimp_importer/0.html', 'mailchimp_importer/1.html']

uid = 'mailchimp'

wizard
alias of MailchimpImporterWizardView

fobi.contrib.plugins.form_importers.mailchimp_importer.forms module

class fobi.contrib.plugins.form_importers.mailchimp_importer.forms.**MailchimpAPIKeyForm**(*data=No files=No auto_id= prefix=Non initial=No error_clas 'django_ label_suffi empty_p field_ord use_requ*

Bases: django.forms.forms.Form

MailchimpAPIKeyForm.

First form the the wizard. Here users are supposed to provide the API key of their Mailchimp account.

base_fields = OrderedDict([('api_key', <django.forms.fields.CharField object at 0x7ff21ecc5190>)])

declared_fields = OrderedDict([('api_key', <django.forms.fields.CharField object at 0x7ff21ecc5190>)])

media

class fobi.contrib.plugins.form_importers.mailchimp_importer.forms.**MailchimpListIDForm**(**args*, ***kwarg*

Bases: django.forms.forms.Form

MailchimpListIDForm.

Second form of the wizard. Here users are supposed to choose the form they want to import.

base_fields = OrderedDict([('list_id', <django.forms.fields.ChoiceField object at 0x7ff21ed23e10>)])

declared_fields = OrderedDict([('list_id', <django.forms.fields.ChoiceField object at 0x7ff21ed23e10>)])

media

fobi.contrib.plugins.form_importers.mailchimp_importer.views module

class fobi.contrib.plugins.form_importers.mailchimp_importer.views.**MailchimpImporterWizardVie**

Bases: *fobi.wizard.views.views.SessionWizardView*

MailchimpImporterWizardView.

done(*form_list*, ***kwargs*)

form_list = [<class 'fobi.contrib.plugins.form_importers.mailchimp_importer.forms.MailchimpAPIKeyForm'>, <class

get_form_kwargs(*step*)
Get form kwargs.

Module contents

Module contents

Module contents

fobi.contrib.themes package

Subpackages

fobi.contrib.themes.bootstrap3 package

Subpackages

fobi.contrib.themes.bootstrap3.widgets package

Subpackages

fobi.contrib.themes.bootstrap3.widgets.form_elements package

Subpackages

fobi.contrib.themes.bootstrap3.widgets.form_elements.date_bootstrap3_widget package

Submodules

fobi.contrib.themes.bootstrap3.widgets.form_elements.date_bootstrap3_widget.apps module
class fobi.contrib.themes.bootstrap3.widgets.form_elements.date_bootstrap3_widget.apps.**Config**

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_bootstrap3_widgets_form_elements_date_bootstrap3_widget'

name = 'fobi.contrib.themes.bootstrap3.widgets.form_elements.date_bootstrap3_widget'

fobi.contrib.themes.bootstrap3.widgets.form_elements.date_bootstrap3_widget.fobi_form_elements module

class fobi.contrib.themes.bootstrap3.widgets.form_elements.date_bootstrap3_widget.fobi_form_e

Bases: *fobi.contrib.plugins.form_elements.fields.date.widgets.BaseDatePluginWidget*

Date plugin widget for Bootstrap 3.

media_css = ['bootstrap3/css/bootstrap-datetimepicker.min.css']

media_js = ['js/moment-with-locales.js', 'bootstrap3/js/bootstrap-datetimepicker.min.js', 'bootstrap3/js/fobi.plugin.date

theme_uid = 'bootstrap3'

Module contents

fobi.contrib.themes.bootstrap3.widgets.form_elements.datetime_bootstrap3_widget package

Submodules

fobi.contrib.themes.bootstrap3.widgets.form_elements.datetime_bootstrap3_widget.apps module

class fobi.contrib.themes.bootstrap3.widgets.form_elements.datetime_bootstrap3_widget.apps.**Co**

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_bootstrap3_widgets_form_elements_datetime_bootstrap3_widget'

name = 'fobi.contrib.themes.bootstrap3.widgets.form_elements.datetime_bootstrap3_widget'

fobi.contrib.themes.bootstrap3.widgets.form_elements.datetime_bootstrap3_widget.fobi_form_elements module

class fobi.contrib.themes.bootstrap3.widgets.form_elements.datetime_bootstrap3_widget.fobi_fo

Bases: *fobi.contrib.plugins.form_elements.fields.datetime.widgets.BaseDateTimePluginWidg*

DateTime plugin widget for Bootstrap 3.

media_css = ['bootstrap3/css/bootstrap-datetimepicker.min.css']

media_js = ['js/moment-with-locales.js', 'bootstrap3/js/bootstrap-datetimepicker.min.js', 'bootstrap3/js/fobi.plugin.date

theme_uid = 'bootstrap3'

Module contents

fobi.contrib.themes.bootstrap3.widgets.form_elements.dummy_bootstrap3_widget package

Submodules

fobi.contrib.themes.bootstrap3.widgets.form_elements.dummy_bootstrap3_widget.apps module

class fobi.contrib.themes.bootstrap3.widgets.form_elements.dummy_bootstrap3_widget.apps.**Confi**

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_bootstrap3_widgets_form_elements_dummy_bootstrap3_widget'

name = 'fobi.contrib.themes.bootstrap3.widgets.form_elements.dummy_bootstrap3_widget'

fobi.contrib.themes.bootstrap3.widgets.form_elements.dummy_bootstrap3_widget.fobi_form_elements module

class fobi.contrib.themes.bootstrap3.widgets.form_elements.dummy_bootstrap3_widget.fobi_form_

Bases: *fobi.contrib.plugins.form_elements.test.dummy.widgets.BaseDummyPluginWidget*

Dummy plugin widget for Boootstrap 3.

media_css = []

media_js = []

theme_uid = 'bootstrap3'

Module contents

fobi.contrib.themes.bootstrap3.widgets.form_elements.slider_bootstrap3_widget package

Submodules

fobi.contrib.themes.bootstrap3.widgets.form_elements.slider_bootstrap3_widget.apps module

class fobi.contrib.themes.bootstrap3.widgets.form_elements.slider_bootstrap3_widget.apps.**Conf**

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_bootstrap3_widgets_form_elements_slider_bootstrap3_widget'

name = 'fobi.contrib.themes.bootstrap3.widgets.form_elements.slider_bootstrap3_widget'

fobi.contrib.themes.bootstrap3.widgets.form_elements.slider_bootstrap3_widget.fobi_form_elements module

class fobi.contrib.themes.bootstrap3.widgets.form_elements.slider_bootstrap3_widget.fobi_form

Bases: *fobi.contrib.plugins.form_elements.fields.slider.widgets.BaseSliderPluginWidget*

Slider plugin widget for Bootstrap 3.

media_css = ['bootstrap3/css/bootstrap-slider.min.css', 'bootstrap3/css/fobi.plugin.slider-bootstrap3-widget.css']

media_js = ['bootstrap3/js/bootstrap-slider.min.js', 'bootstrap3/js/fobi.plugin.slider-bootstrap3-widget.js']

theme_uid = 'bootstrap3'

Module contents

Module contents

Module contents

Submodules

fobi.contrib.themes.bootstrap3.apps module

class fobi.contrib.themes.bootstrap3.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_bootstrap3'

name = 'fobi.contrib.themes.bootstrap3'

fobi.contrib.themes.bootstrap3.fobi_themes module

class fobi.contrib.themes.bootstrap3.fobi_themes.**Bootstrap3Theme**(*user=None*)

Bases: fobi.base.BaseTheme

Bootstrap3 theme.

add_form_element_entry_ajax_template = 'bootstrap3/add_form_element_entry_ajax.html'

add_form_element_entry_template = 'bootstrap3/add_form_element_entry.html'

add_form_handler_entry_ajax_template = 'bootstrap3/add_form_handler_entry_ajax.html'

add_form_handler_entry_template = 'bootstrap3/add_form_handler_entry.html'

add_form_wizard_handler_entry_ajax_template = 'bootstrap3/add_form_wizard_handler_entry_ajax.html'

add_form_wizard_handler_entry_template = 'bootstrap3/add_form_wizard_handler_entry.html'

base_template = 'bootstrap3/base.html'

create_form_entry_ajax_template = 'bootstrap3/create_form_entry_ajax.html'

create_form_entry_template = 'bootstrap3/create_form_entry.html'

create_form_wizard_entry_ajax_template = 'bootstrap3/create_form_wizard_entry_ajax.html'

create_form_wizard_entry_template = 'bootstrap3/create_form_wizard_entry.html'

dashboard_template = 'bootstrap3/dashboard.html'

edit_form_element_entry_ajax_template = 'bootstrap3/edit_form_element_entry_ajax.html'

edit_form_element_entry_template = 'bootstrap3/edit_form_element_entry.html'

edit_form_entry_ajax_template = 'bootstrap3/edit_form_entry_ajax.html'

edit_form_entry_template = 'bootstrap3/edit_form_entry.html'

edit_form_handler_entry_ajax_template = 'bootstrap3/edit_form_handler_entry_ajax.html'

edit_form_handler_entry_template = 'bootstrap3/edit_form_handler_entry.html'

edit_form_wizard_entry_ajax_template = 'bootstrap3/edit_form_wizard_entry_ajax.html'

edit_form_wizard_entry_template = 'bootstrap3/edit_form_wizard_entry.html'

edit_form_wizard_handler_entry_ajax_template = 'bootstrap3/edit_form_wizard_handler_entry_ajax.html'

edit_form_wizard_handler_entry_template = 'bootstrap3/edit_form_wizard_handler_entry.html'

embed_form_entry_submitted_ajax_template = 'bootstrap3/embed_form_entry_submitted_ajax.html'

form_ajax = 'bootstrap3/snippets/form_ajax.html'

form_delete_form_entry_option_class = 'glyphicon glyphicon-remove'

form_edit_form_entry_option_class = 'glyphicon glyphicon-edit'

form_element_checkbox_html_class = 'checkbox'

form_element_html_class = 'form-control'

form_entry_submitted_ajax_template = 'bootstrap3/form_entry_submitted_ajax.html'

form_entry_submitted_template = 'bootstrap3/form_entry_submitted.html'

form_importer_ajax_template = 'bootstrap3/form_importer_ajax.html'

form_importer_template = 'bootstrap3/form_importer.html'

form_list_container_class = 'list-inline'

form_non_field_and_hidden_errors_snippet_template = 'bootstrap3/snippets/form_non_field_and_hidden

form_properties_snippet_template_name = 'bootstrap3/snippets/form_properties_snippet.html'

form_snippet_template_name = 'bootstrap3/snippets/form_snippet.html'

form_view_form_entry_option_class = 'glyphicon glyphicon-list'

form_wizard_ajax = 'bootstrap3/snippets/form_wizard_ajax.html'

form_wizard_properties_snippet_template_name = 'bootstrap3/snippets/form_wizard_properties_snippet.ht

form_wizard_snippet_template_name = 'bootstrap3/snippets/form_wizard_snippet.html'

form_wizard_template = 'bootstrap3/snippets/form_wizard.html'

form_wizards_dashboard_template = 'bootstrap3/form_wizards_dashboard.html'

forms_list_template = 'bootstrap3/forms_list.html'

master_base_template = 'bootstrap3/_base.html'

media_css = ('bootstrap3/css/bootstrap.css', 'bootstrap3/css/bootstrap3_fobi_extras.css', 'css/fobi.core.css')

media_js = ('js/jquery-1.10.2.min.js', 'jquery-ui/js/jquery-ui-1.10.4.custom.min.js', 'bootstrap3/js/bootstrap.min.js', 'js/

messages_snippet_template_name = 'bootstrap3/snippets/messages_snippet.html'

name = <django.utils.functional.__proxy__ object>

uid = 'bootstrap3'

view_embed_form_entry_ajax_template = 'bootstrap3/view_embed_form_entry_ajax.html'

view_form_entry_ajax_template = 'bootstrap3/view_form_entry_ajax.html'

view_form_entry_template = 'bootstrap3/view_form_entry.html'

view_form_wizard_entry_ajax_template = 'bootstrap3/view_form_wizard_entry_ajax.html'

view_form_wizard_entry_template = 'bootstrap3/view_form_wizard_entry.html'

Module contents

fobi.contrib.themes.djangocms_admin_style_theme package

Subpackages

fobi.contrib.themes.djangocms_admin_style_theme.widgets package

Subpackages

fobi.contrib.themes.djangocms_admin_style_theme.widgets.form_handlers package

Subpackages

fobi.contrib.themes.djangocms_admin_style_theme.widgets.form_handlers.db_store package

Submodules

fobi.contrib.themes.djangocms_admin_style_theme.widgets.form_handlers.db_store.apps module

class fobi.contrib.themes.djangocms_admin_style_theme.widgets.form_handlers.db_store.apps.**Cor**

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_djangocms_admin_style_theme_widgets_form_handlers_db_store'

name = 'fobi.contrib.themes.djangocms_admin_style_theme.widgets.form_handlers.db_store'

fobi.contrib.themes.djangocms_admin_style_theme.widgets.form_handlers.db_store.fobi_form_elements module

class fobi.contrib.themes.djangocms_admin_style_theme.widgets.form_handlers.db_store.fobi_for

Bases: *fobi.contrib.plugins.form_handlers.db_store.widgets.BaseDbStorePluginWidget*

DbStore plugin widget for djangocms_admin_style_theme theme.

export_entries_icon_class = ''

theme_uid = 'djangocms_admin_style_theme'

view_entries_icon_class = ''

Module contents

Module contents

Module contents

Submodules

fobi.contrib.themes.djangocms_admin_style_theme.apps module

class fobi.contrib.themes.djangocms_admin_style_theme.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_djangocms_admin_style_theme'

name = 'fobi.contrib.themes.djangocms_admin_style_theme'

fobi.contrib.themes.djangocms_admin_style_theme.fobi_themes module

class fobi.contrib.themes.djangocms_admin_style_theme.fobi_themes.**DjangoCMSAdminStyleTheme**(*us*

Bases: fobi.base.BaseTheme

A theme that has a native djangocms-admin-style style.

add_form_element_entry_ajax_template = 'djangocms_admin_style_theme/add_form_element_entry_ajax.htm

add_form_element_entry_template = 'djangocms_admin_style_theme/add_form_element_entry.html'

add_form_handler_entry_ajax_template = 'djangocms_admin_style_theme/add_form_handler_entry_ajax.htm

add_form_handler_entry_template = 'djangocms_admin_style_theme/add_form_handler_entry.html'

base_edit_template = 'djangocms_admin_style_theme/base_edit.html'

base_template = 'djangocms_admin_style_theme/base.html'

base_view_template = 'djangocms_admin_style_theme/base_view.html'

create_form_entry_ajax_template = 'djangocms_admin_style_theme/create_form_entry_ajax.html'

create_form_entry_template = 'djangocms_admin_style_theme/create_form_entry.html'

dashboard_template = 'djangocms_admin_style_theme/dashboard.html'

edit_form_element_entry_ajax_template = 'djangocms_admin_style_theme/edit_form_element_entry_ajax.ht

edit_form_element_entry_template = 'djangocms_admin_style_theme/edit_form_element_entry.html'

edit_form_entry_ajax_template = 'djangocms_admin_style_theme/edit_form_entry_ajax.html'

classmethod edit_form_entry_edit_option_html()

For adding the edit link to edit form entry view.

Return str

classmethod edit_form_entry_help_text_extra()

For adding the edit link to edit form entry view.

Return str

edit_form_entry_template = 'djangocms_admin_style_theme/edit_form_entry.html'

edit_form_handler_entry_ajax_template = 'djangocms_admin_style_theme/edit_form_handler_entry_ajax.ht

edit_form_handler_entry_template = 'djangocms_admin_style_theme/edit_form_handler_entry.html'

form_ajax = 'djangocms_admin_style_theme/snippets/form_ajax.html'

form_delete_form_entry_option_class = 'deletelink'

form_edit_ajax = 'djangocms_admin_style_theme/snippets/form_edit_ajax.html'

form_edit_form_entry_option_class = 'edit'

form_edit_snippet_template_name = 'djangocms_admin_style_theme/snippets/form_edit_snippet.html'

form_element_checkbox_html_class = 'checkbox'

form_element_html_class = 'vTextField'

form_entry_submitted_ajax_template = 'djangocms_admin_style_theme/form_entry_submitted_ajax.html'

form_entry_submitted_template = 'djangocms_admin_style_theme/form_entry_submitted.html'

form_list_container_class = 'list-inline'

form_properties_snippet_template_name = 'djangocms_admin_style_theme/snippets/form_properties_snippet

form_radio_element_html_class = 'radiolist'

form_snippet_template_name = 'djangocms_admin_style_theme/snippets/form_snippet.html'

form_view_form_entry_option_class = 'viewlink'

form_view_snippet_template_name = 'djangocms_admin_style_theme/snippets/form_view_snippet.html'

forms_list_template = 'djangocms_admin_style_theme/forms_list.html'

import_form_entry_ajax_template = 'djangocms_admin_style_theme/import_form_entry_ajax.html'

import_form_entry_template = 'djangocms_admin_style_theme/import_form_entry.html'

master_base_template = 'djangocms_admin_style_theme/_base.html'

media_css = ('djangocms_admin_style_theme/css/fobi.djangocms_admin_style_theme.css', 'jquery-ui/css/smoothness/j

media_js = ('js/jquery-1.10.2.min.js', 'jquery-ui/js/jquery-ui-1.10.4.custom.min.js', 'js/jquery.slugify.js', 'js/fobi.core.js'

messages_snippet_template_name = 'djangocms_admin_style_theme/snippets/messages_snippet.html'

name = <django.utils.functional.__proxy__ object>

uid = 'djangocms_admin_style_theme'

view_form_entry_ajax_template = 'djangocms_admin_style_theme/view_form_entry_ajax.html'

view_form_entry_template = 'djangocms_admin_style_theme/view_form_entry.html'

Module contents

fobi.contrib.themes.foundation5 package

Subpackages

fobi.contrib.themes.foundation5.widgets package

Subpackages

fobi.contrib.themes.foundation5.widgets.form_elements package

Subpackages

fobi.contrib.themes.foundation5.widgets.form_elements.date_foundation5_widget package

Submodules

fobi.contrib.themes.foundation5.widgets.form_elements.date_foundation5_widget.apps module

class fobi.contrib.themes.foundation5.widgets.form_elements.date_foundation5_widget.apps.**Conf**

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_foundation5_widgets_form_elements_date_foundation5_widget'

name = 'fobi.contrib.themes.foundation5.widgets.form_elements.date_foundation5_widget'

fobi.contrib.themes.foundation5.widgets.form_elements.date_foundation5_widget.fobi_form_elements module

class fobi.contrib.themes.foundation5.widgets.form_elements.date_foundation5_widget.fobi_form

Bases: *fobi.contrib.plugins.form_elements.fields.date.widgets.BaseDatePluginWidget*

Date plugin widget for Foundation 5.

media_css = ['foundation5/css/foundation-datepicker.css']

media_js = ['js/moment-with-locales.js', 'foundation5/js/foundation-datepicker.js', 'foundation5/js/fobi.plugin.date-foun

theme_uid = 'foundation5'

Module contents

fobi.contrib.themes.foundation5.widgets.form_elements.datetime_foundation5_widget package

Submodules

fobi.contrib.themes.foundation5.widgets.form_elements.datetime_foundation5_widget.apps module

class fobi.contrib.themes.foundation5.widgets.form_elements.datetime_foundation5_widget.apps.

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_foundation5_widgets_form_elements_datetime_foundation5_widget'

name = 'fobi.contrib.themes.foundation5.widgets.form_elements.datetime_foundation5_widget'

fobi.contrib.themes.foundation5.widgets.form_elements.datetime_foundation5_widget.fobi_form_elements module

class fobi.contrib.themes.foundation5.widgets.form_elements.datetime_foundation5_widget.fobi_

Bases: *fobi.contrib.plugins.form_elements.fields.datetime.widgets.BaseDateTimePluginWidg*

DateTime plugin widget for Foundation 5.

media_css = ['foundation5/css/foundation-datetimepicker.css']

media_js = ['js/moment-with-locales.js', 'foundation5/js/foundation-datetimepicker.js', 'foundation5/js/fobi.plugin.datet

theme_uid = 'foundation5'

Module contents

fobi.contrib.themes.foundation5.widgets.form_elements.dummy_foundation5_widget package

Submodules

fobi.contrib.themes.foundation5.widgets.form_elements.dummy_foundation5_widget.apps module

class fobi.contrib.themes.foundation5.widgets.form_elements.dummy_foundation5_widget.apps.**Cor**

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_foundation5_widgets_form_elements_dummy_foundation5_widget'

name = 'fobi.contrib.themes.foundation5.widgets.form_elements.dummy_foundation5_widget'

fobi.contrib.themes.foundation5.widgets.form_elements.dummy_foundation5_widget.fobi_form_elements module

class fobi.contrib.themes.foundation5.widgets.form_elements.dummy_foundation5_widget.fobi_for

Bases: *fobi.contrib.plugins.form_elements.test.dummy.widgets.BaseDummyPluginWidget*

Dummy plugin widget for Foundation 5.

media_css = []

media_js = []

theme_uid = 'foundation5'

Module contents

Module contents

fobi.contrib.themes.foundation5.widgets.form_handlers package

Subpackages

fobi.contrib.themes.foundation5.widgets.form_handlers.db_store_foundation5_widget package

Submodules

fobi.contrib.themes.foundation5.widgets.form_handlers.db_store_foundation5_widget.apps module

class fobi.contrib.themes.foundation5.widgets.form_handlers.db_store_foundation5_widget.apps.

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_foundation5_widgets_form_handlers_db_store_foundation5_widget'

name = 'fobi.contrib.themes.foundation5.widgets.form_handlers.db_store_foundation5_widget'

fobi.contrib.themes.foundation5.widgets.form_handlers.db_store_foundation5_widget.fobi_form_elements module

class fobi.contrib.themes.foundation5.widgets.form_handlers.db_store_foundation5_widget.fobi_

Bases: *fobi.contrib.plugins.form_handlers.db_store.widgets.BaseDbStorePluginWidget*

DbStore plugin widget for Foundation 5.

export_entries_icon_class = 'fi-page-export'

theme_uid = 'foundation5'

view_entries_icon_class = 'fi-list'

view_saved_form_data_entries_template_name = 'db_store_foundation5_widget/view_saved_form_data_entri

Module contents

Module contents

Module contents

Submodules

fobi.contrib.themes.foundation5.apps module

class fobi.contrib.themes.foundation5.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_foundation5'

name = 'fobi.contrib.themes.foundation5'

fobi.contrib.themes.foundation5.fobi_themes module

class fobi.contrib.themes.foundation5.fobi_themes.**Foundation5Theme**(*user=None*)

Bases: fobi.base.BaseTheme

Foundation5 theme.

Based on the "Workspace" example of the Foundation 5. Click here for more.

add_form_element_entry_ajax_template = 'foundation5/add_form_element_entry_ajax.html'

add_form_element_entry_template = 'foundation5/add_form_element_entry.html'

add_form_handler_entry_ajax_template = 'foundation5/add_form_handler_entry_ajax.html'

add_form_handler_entry_template = 'foundation5/add_form_handler_entry.html'

base_template = 'foundation5/base.html'

create_form_entry_ajax_template = 'foundation5/create_form_entry_ajax.html'

create_form_entry_template = 'foundation5/create_form_entry.html'

dashboard_template = 'foundation5/dashboard.html'

edit_form_element_entry_ajax_template = 'foundation5/edit_form_element_entry_ajax.html'

edit_form_element_entry_template = 'foundation5/edit_form_element_entry.html'

edit_form_entry_ajax_template = 'foundation5/edit_form_entry_ajax.html'

edit_form_entry_template = 'foundation5/edit_form_entry.html'

edit_form_handler_entry_ajax_template = 'foundation5/edit_form_handler_entry_ajax.html'

edit_form_handler_entry_template = 'foundation5/edit_form_handler_entry.html'

form_ajax = 'foundation5/snippets/form_ajax.html'

form_delete_form_entry_option_class = 'fi-page-delete'

form_edit_form_entry_option_class = 'fi-page-edit'

form_element_checkbox_html_class = 'checkbox'

form_element_html_class = 'form-control'

form_entry_submitted_ajax_template = 'foundation5/form_entry_submitted_ajax.html'

form_entry_submitted_template = 'foundation5/form_entry_submitted.html'

form_list_container_class = 'inline-list'

form_non_field_and_hidden_errors_snippet_template = 'foundation5/snippets/form_non_field_and_hidde

form_properties_snippet_template_name = 'foundation5/snippets/form_properties_snippet.html'

form_snippet_template_name = 'foundation5/snippets/form_snippet.html'

form_view_form_entry_option_class = 'fi-list'

forms_list_template = 'foundation5/forms_list.html'

import_form_entry_ajax_template = 'foundation5/import_form_entry_ajax.html'

import_form_entry_template = 'foundation5/import_form_entry.html'

master_base_template = 'foundation5/_base.html'

media_css = ('foundation5/css/foundation.min.css', 'foundation5/css/foundation_fobi_extras.css', 'foundation5/icons/3/i

media_js = ('foundation5/js/vendor/modernizr.js', 'foundation5/js/vendor/jquery.js', 'jquery-ui/js/jquery-ui-1.10.4.custo

messages_snippet_template_name = 'foundation5/snippets/messages_snippet.html'

name = <django.utils.functional.__proxy__ object>

uid = 'foundation5'

view_form_entry_ajax_template = 'foundation5/view_form_entry_ajax.html'

view_form_entry_template = 'foundation5/view_form_entry.html'

Module contents

fobi.contrib.themes.simple package

Subpackages

fobi.contrib.themes.simple.widgets package

Subpackages

fobi.contrib.themes.simple.widgets.form_handlers package

Subpackages

fobi.contrib.themes.simple.widgets.form_handlers.db_store package

Submodules

fobi.contrib.themes.simple.widgets.form_handlers.db_store.apps module

class fobi.contrib.themes.simple.widgets.form_handlers.db_store.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_simple_widgets_form_handlers_db_store'

name = 'fobi.contrib.themes.simple.widgets.form_handlers.db_store'

fobi.contrib.themes.simple.widgets.form_handlers.db_store.fobi_form_elements module

class fobi.contrib.themes.simple.widgets.form_handlers.db_store.fobi_form_elements.**DbStorePlu**

Bases: *fobi.contrib.plugins.form_handlers.db_store.widgets.BaseDbStorePluginWidget*

DbStore plugin widget for Simple theme.

export_entries_icon_class = ''

theme_uid = 'simple'

view_entries_icon_class = ''

Module contents

Module contents

Module contents

Submodules

fobi.contrib.themes.simple.apps module

class fobi.contrib.themes.simple.apps.**Config**(*app_name*, *app_module*)

Bases: django.apps.config.AppConfig

Config.

label = 'fobi_contrib_themes_simple'

name = 'fobi.contrib.themes.simple'

fobi.contrib.themes.simple.fobi_themes module

class fobi.contrib.themes.simple.fobi_themes.**SimpleTheme**(*user=None*)

Bases: fobi.base.BaseTheme

Simple theme that has a native Django style.

add_form_element_entry_ajax_template = 'simple/add_form_element_entry_ajax.html'

add_form_element_entry_template = 'simple/add_form_element_entry.html'

add_form_handler_entry_ajax_template = 'simple/add_form_handler_entry_ajax.html'

add_form_handler_entry_template = 'simple/add_form_handler_entry.html'

base_edit_template = 'simple/base_edit.html'

base_template = 'simple/base.html'

base_view_template = 'simple/base_view.html'

create_form_entry_ajax_template = 'simple/create_form_entry_ajax.html'

create_form_entry_template = 'simple/create_form_entry.html'

dashboard_template = 'simple/dashboard.html'

edit_form_element_entry_ajax_template = 'simple/edit_form_element_entry_ajax.html'

edit_form_element_entry_template = 'simple/edit_form_element_entry.html'

edit_form_entry_ajax_template = 'simple/edit_form_entry_ajax.html'

edit_form_entry_template = 'simple/edit_form_entry.html'

edit_form_handler_entry_ajax_template = 'simple/edit_form_handler_entry_ajax.html'

edit_form_handler_entry_template = 'simple/edit_form_handler_entry.html'

form_ajax = 'simple/snippets/form_ajax.html'

form_delete_form_entry_option_class = 'glyphicon glyphicon-remove'

form_edit_ajax = 'simple/snippets/form_edit_ajax.html'

form_edit_form_entry_option_class = 'glyphicon glyphicon-edit'

form_edit_snippet_template_name = 'simple/snippets/form_edit_snippet.html'

form_element_checkbox_html_class = 'checkbox'

form_element_html_class = 'vTextField'

form_entry_submitted_ajax_template = 'simple/form_entry_submitted_ajax.html'

form_entry_submitted_template = 'simple/form_entry_submitted.html'

form_list_container_class = 'list-inline'

form_properties_snippet_template_name = 'simple/snippets/form_properties_snippet.html'

form_radio_element_html_class = 'radiolist'

form_snippet_template_name = 'simple/snippets/form_snippet.html'

form_view_form_entry_option_class = 'glyphicon glyphicon-list'

form_view_snippet_template_name = 'simple/snippets/form_view_snippet.html'

forms_list_template = 'simple/forms_list.html'

import_form_entry_ajax_template = 'simple/import_form_entry_ajax.html'

import_form_entry_template = 'simple/import_form_entry.html'

master_base_template = 'simple/_base.html'

media_css = ('simple/css/fobi.simple.css', 'jquery-ui/css/django-admin-theme/jquery-ui-1.10.4.custom.min.css')

media_js = ('js/jquery-1.10.2.min.js', 'jquery-ui/js/jquery-ui-1.10.4.custom.min.js', 'js/jquery.slugify.js', 'js/fobi.core.js'

messages_snippet_template_name = 'simple/snippets/messages_snippet.html'

name = <django.utils.functional.__proxy__ object>

uid = 'simple'

view_form_entry_ajax_template = 'simple/view_form_entry_ajax.html'

view_form_entry_template = 'simple/view_form_entry.html'

Module contents

Module contents

Module contents

fobi.integration package

Submodules

fobi.integration.helpers module

fobi.integration.helpers.**get_template_choices**(*source*, *choices*, *theme_specific_choices_key*)

Get the template choices.

It's possible to provide theme templates per theme or just per project.

Parameters

- **source** (*str*) – Example value 'feincms_integration'.
- **or list choices** (*tuple*) –
- **theme_specific_choices_key** (*str*) –

Return list

fobi.integration.processors module

class fobi.integration.processors.**IntegrationProcessor**

Bases: object

Generic integration processor.

Parameters

- **form_sent_get_param** (*str*) –

- **can_redirect** (*bool*) – If set to True, if not authenticated an attempt to redirect user to a login page would be made. Otherwise, a message about authentication would be generated instead (in place of the form). Some content management systems, like Django-CMS, aren't able to redirect on plugin level. For those systems, the value of `can_redirect` should be set to False.
- **login_required_template_name** (*str*) – Template to be used for rendering the login required message. This is only important when `login_required_redirect` is set to False.

can_redirect = True

form_sent_get_param = 'sent'

get_context_data(*request*, *instance*, ***kwargs*)

get_form_template_name(*request*, *instance*)

get_login_required_template_name(*request*, *instance*)

get_success_page_template_name(*request*, *instance*)

integration_check(*instance*)
Integration check.

Performs a simple check to identify whether the model instance has been implemented according to the expectations.

login_required_template_name = 'fobi/integration/login_required.html'

Module contents

fobi.management package

Subpackages

fobi.management.commands package

Submodules

fobi.management.commands.fobi_find_broken_entries module

class `fobi.management.commands.fobi_find_broken_entries.`**Command**(*stdout=None*, *stderr=None*, *no_color=False*)

Bases: `django.core.management.base.BaseCommand`

Find the broken plugin records in the database:

- `fobi.models.FormElementEntry`
- `fobi.models.FormHandlerEntry`

handle(**args*, ***options*)
Handle.

fobi.management.commands.fobi_migrate_03_to_04 module

class fobi.management.commands.fobi_migrate_03_to_04.**Command**(*stdout=None*, *stderr=None*, *no_color=False*)

Bases: django.core.management.base.BaseCommand

Database related changes necessary to upgrade fobi==0.3.* to fobi==0.4.

The full list of changes is listed below:

- Change the "birthday" occurrences to "date_drop_down".

handle(**args*, ***options*)

Handle.

fobi.management.commands.fobi_sync_plugins module

class fobi.management.commands.fobi_sync_plugins.**Command**(*stdout=None*, *stderr=None*, *no_color=False*)

Bases: django.core.management.base.BaseCommand

Adds the missing plugins to database.

This command shall be ran every time a developer adds a new plugin. The following plugins are affected:

- fobi.models.FormElementPlugin
- fobi.models.FormHandlerPlugin

handle(**args*, ***options*)

Handle.

fobi.management.commands.fobi_update_plugin_data module

class fobi.management.commands.fobi_update_plugin_data.**Command**(*stdout=None*, *stderr=None*, *no_color=False*)

Bases: django.core.management.base.BaseCommand

Updates the plugin data for all entries of all users.

Rules for update are specified in the plugin itself.

This command shall be ran if significant changes have been made to the system for which the data shall be updated.

handle(**args*, ***options*)

Handle.

Module contents

Module contents

fobi.migrations package

Submodules

fobi.migrations.0001_initial module

class fobi.migrations.0001_initial.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'auth', u'__first__'), (u'auth', u'0006_require_contenttypes_0002')]

operations = [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'plugin_uid', <django.db.models.fi

fobi.migrations.0002_auto_20150912_1744 module

class fobi.migrations.0002_auto_20150912_1744.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0001_initial')]

operations = [<AddField field=<django.db.models.fields.DateTimeField>, name=u'created', model_name=u'formentry

fobi.migrations.0003_auto_20160517_1005 module

class fobi.migrations.0003_auto_20160517_1005.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0002_auto_20150912_1744')]

operations = [<AlterField field=<django.db.models.fields.CharField>, name=u'plugin_uid', model_name=u'formeleme

fobi.migrations.0004_auto_20160906_1513 module

class fobi.migrations.0004_auto_20160906_1513.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0003_auto_20160517_1005')]

operations = [<AlterField field=<django.db.models.fields.CharField>, name=u'plugin_uid', model_name=u'formeleme

fobi.migrations.0005_auto_20160908_1457 module

class fobi.migrations.0005_auto_20160908_1457.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0004_auto_20160906_1513')]

operations = [<AlterField field=<django.db.models.fields.CharField>, name=u'plugin_uid', model_name=u'formeleme

fobi.migrations.0006_auto_20160911_1549 module

class fobi.migrations.0006_auto_20160911_1549.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0005_auto_20160908_1457')]

operations = [<AlterField field=<django.db.models.fields.CharField>, name=u'plugin_uid', model_name=u'formhand

fobi.migrations.0007_auto_20160926_1652 module

class fobi.migrations.0007_auto_20160926_1652.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0006_auto_20160911_1549')]

operations = [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'position', <django.db.models.field

fobi.migrations.0008_formwizardhandlerentry module

class fobi.migrations.0008_formwizardhandlerentry.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0007_auto_20160926_1652')]

operations = [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'plugin_data', <django.db.models.

fobi.migrations.0009_formwizardentry_wizard_type module

class fobi.migrations.0009_formwizardentry_wizard_type.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0008_formwizardhandlerentry')]

operations = [<AddField field=<django.db.models.fields.CharField>, name=u'wizard_type', model_name=u'formwiza

fobi.migrations.0010_formwizardhandler module

class fobi.migrations.0010_formwizardhandler.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'auth', u'0007_alter_validators_add_error_messages'), (u'auth', u'__first__'), (u'fobi', u'0009_form

operations = [<CreateModel fields=[(u'id', <django.db.models.fields.AutoField>), (u'plugin_uid', <django.db.models.fi

fobi.migrations.0011_formentry_title module

class fobi.migrations.0011_formentry_title.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0010_formwizardhandler')]

operations = [<AddField field=<django.db.models.fields.CharField>, name=u'title', model_name=u'formentry'>]

fobi.migrations.0012_auto_20161109_1550 module

class fobi.migrations.0012_auto_20161109_1550.**Migration**(*name*, *app_label*)
Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0011_formentry_title')]

operations = [<AddField field=<django.db.models.fields.CharField>, name=u'title', model_name=u'formwizardentry'

fobi.migrations.0013_formwizardentry_show_all_navigation_buttons module

class fobi.migrations.0013_formwizardentry_show_all_navigation_buttons.**Migration**(*name*, *app_label*)

Bases: django.db.migrations.migration.Migration

dependencies = [(u'fobi', u'0012_auto_20161109_1550')]

operations = [<AddField field=<django.db.models.fields.BooleanField>, name=u'show_all_navigation_buttons', model

Module contents

fobi.south_migrations package

Submodules

fobi.south_migrations.0001_initial module

fobi.south_migrations.0002_auto__add_field_formentry_created__add_field_formentry_updated module

fobi.south_migrations.0003_auto__add_formwizardformentry__add_unique_formwizardformentry_form_wiz module

Module contents

fobi.templatetags package

Submodules

fobi.templatetags.fobi_tags module

fobi.templatetags.fobi_tags.**get_fobi_form_handler_plugin_custom_actions**(*parser*, *token*)

Get the form handler plugin custom actions.

Note, that plugin shall be a instance of fobi.models.FormHandlerEntry.

Syntax

{% get_fobi_form_handler_plugin_custom_actions [plugin] [form_entry] as [context_var_name] %}

Example

{% get_fobi_form_handler_plugin_custom_actions plugin form_entry as form_handler_plugin_custom_actions %}

fobi.templatetags.fobi_tags.**get_fobi_form_wizard_handler_plugin_custom_actions**(*parser*, *token*)

Get the form wizard handler plugin custom actions.

Note, that plugin shall be a instance of fobi.models.FormWizardHandlerEntry.

Syntax

{% get_fobi_form_wizard_handler_plugin_custom_actions [plugin] [form_wizard_entry] as [context_var_name] %}

Example

{% get_fobi_form_wizard_handler_plugin_custom_actions

plugin form_wizard_entry as form_wizard_handler_plugin_custom_actions %}

fobi.templatetags.fobi_tags.**get_fobi_plugin**(*parser*, *token*)
Get the plugin.

Note, that `entry` shall be a instance of `fobi.models.FormElementEntry` or `fobi.models.FormHandlerEntry`.

Syntax {% get_fobi_plugin entry as [context_var_name] %}

Example {% get_fobi_plugin entry as plugin %}

{% get_fobi_plugin entry as plugin %} {{ plugin.render }}

fobi.templatetags.fobi_tags.**get_form_field_type**(*parser*, *token*)
Get form field type.

Syntax:

```
{% get_form_field_type [field] as [context_var_name] %}
```

Example:

```
{% get_form_field_type form.field as form_field_type %}
{% if form_field_type.is_checkbox %}
    ...
{% endif %}
```

fobi.templatetags.fobi_tags.**get_form_hidden_fields_errors**(*parser*, *token*)
Get form hidden fields errors.

Syntax {% get_form_hidden_fields_errors [form] as [context_var_name] %}

Example {% get_form_hidden_fields_errors form as form_hidden_fields_errors %} {{ form_hidden_fields_errors.as_ul }}

fobi.templatetags.fobi_tags.**has_edit_form_entry_permissions**(*parser*, *token*)
Checks the permissions

Syntax {% has_edit_form_entry_permissions as [var_name] %}

Example {% has_edit_form_entry_permissions %}

or

{% has_edit_form_entry_permissions as has_permissions %}

fobi.templatetags.fobi_tags.**render_auth_link**(*context*)
Render auth link.

fobi.templatetags.fobi_tags.**render_fobi_forms_list**(*context*, *queryset*, **args*, ***kwargs*)
Render the list of fobi forms.

Syntax {% render_fobi_forms_list [queryset] [show_edit_link] [show_delete_link] [show_export_link] %}

Example {% render_fobi_forms_list queryset show_edit_link=True show_delete_link=False show_export_link=False %}

fobi.templatetags.future_compat module

fobi.templatetags.future_compat.**firstof**(*parser*, *token*, *escape=False*)
Outputs the first variable passed that is not False.

Outputs the first variable passed that is not False, without escaping.

Outputs nothing if all the passed variables are False.

Sample usage:

```
{% firstof var1 var2 var3 %}
```

This is equivalent to:

```
{% if var1 %}
    {{ var1|safe }}
{% elif var2 %}
    {{ var2|safe }}
{% elif var3 %}
    {{ var3|safe }}
{% endif %}
```

but obviously much cleaner!

You can also use a literal string as a fallback value in case all passed variables are False:

```
{% firstof var1 var2 var3 "fallback value" %}
```

If you want to escape the output, use a filter tag:

```
{% filter force_escape %}
    {% firstof var1 var2 var3 "fallback value" %}
{% endfilter %}
```

Module contents

fobi.tests package

Submodules

fobi.tests.base module

fobi.tests.base.**print_info**(*func*)
Prints some useful info.

fobi.tests.base.**skip**(*func*)
Simply skips the test.

fobi.tests.base.**is_fobi_setup_completed**()
Is fobi setup completed?

fobi.tests.base.**mark_fobi_setup_as_completed**()
Mark fobi setup as completed.

fobi.tests.constants module

fobi.tests.data module

fobi.tests.helpers module

fobi.tests.helpers.**get_or_create_admin_user**()
Create a user for testing the fobi.

TODO: At the moment an admin account is being tested. Automated tests with diverse accounts are to be implemented.

fobi.tests.helpers.**get_or_create_admin_user**()
Create a user for testing the fobi.

TODO: At the moment an admin account is being tested. Automated tests with diverse accounts are to be implemented.

fobi.tests.helpers.**create_form_with_entries**(*user=None*, *create_entries_if_form_exist=True*)
Create test form with entries.

Fills the form with pre-defined plugins.

Parameters

- **user** (*django.contrib.auth.models.User*) –
- **create_entries_if_form_exist** (*bool*) – If set to True, entries are being created even if form already exists (a database record).

Return fobi.models.FormEntry Instance of `fobi.models.FormEntry` with a number of form elements and handlers filled in.

fobi.tests.helpers.**db_clean_up**()
Clean up the database.

Clean up the database by removing all form element and form handler entries.

fobi.tests.helpers.**phantom_js_clean_up**()
Clean up Phantom JS.

Kills all phantomjs instances, disregard of their origin.

fobi.tests.test_browser_build_dynamic_forms module

class fobi.tests.test_browser_build_dynamic_forms.**BaseFobiBrowserBuldDynamicFormsTest**(*methodNa*
Bases: `django.test.testcases.LiveServerTestCase`

Browser tests django-fobi bulding forms functionality.

Backed up by selenium. This test is based on the bootstrap3 theme.

LIVE_SERVER_URL = None

cleans_up_after_itself = True

e = AttributeError("'Settings' object has no attribute 'LIVE_SERVER_URL'",)

classmethod setUpClass()
Set up class.

tearDown()
Tear down.

classmethod tearDownClass()
Tear down class.

test_1001_open_dashboard(**args*, ***kwargs*)
Inner.

test_2001_add_form(**args*, ***kwargs*)
Inner.

test_2002_edit_form(**args*, ***kwargs*)
Inner.

test_2003_delete_form(**args*, ***kwargs*)
Inner.

test_2004_submit_form(**args*, ***kwargs*)
Inner.

test_3001_add_form_elements(**args*, ***kwargs*)
Inner.

test_3002_remove_form_elements(**args*, ***kwargs*)
Inner.

test_3003_edit_form_elements(**args*, ***kwargs*)
Inner.

test_4001_add_form_handlers(**args*, ***kwargs*)
Inner.

test_4002_remove_form_handlers(**args*, ***kwargs*)
Inner.

test_4003_edit_form_handlers(**args*, ***kwargs*)
Inner.

fobi.tests.test_core module

class fobi.tests.test_core.**FobiCoreTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of django-fobi core functionality.

setUp()
Set up.

test_01_get_registered_form_element_plugins(**args*, ***kwargs*)
Inner.

test_02_get_registered_form_handler_plugins(**args*, ***kwargs*)
Inner.

test_03_get_registered_form_callbacks(**args*, ***kwargs*)
Inner.

test_04_get_registered_themes(**args*, ***kwargs*)
Inner.

test_05_action_url(**args*, ***kwargs*)
Inner.

fobi.tests.test_dynamic_forms module

class fobi.tests.test_dynamic_forms.**FobiDynamicFormsTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of django-fob dynamic forms functionality.

setUp()
Set up.

test_01_assemble_form_class_and_render_form(**args*, ***kwargs*)
Inner.

fobi.tests.test_form_importers_mailchimp module

class fobi.tests.test_form_importers_mailchimp.**FormImpotersMailchimpTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of form importers mailchimp functionality.

setUp()
Set up.

test_01_test_mailchimp_impoter(**args*, ***kwargs*)
Inner.

fobi.tests.test_sortable_dict module

class fobi.tests.test_sortable_dict.**FobiDataStructuresTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of django-fobi data_structures module functionality.

setUp()
Set up.

test_01_sortable_dict_move_before_key(**args*, ***kwargs*)
Inner.

test_02_sortable_dict_move_after_key(**args*, ***kwargs*)
Inner.

Module contents

fobi.urls package

Submodules

fobi.urls.edit module

fobi.urls.view module

Module contents

fobi.wizard package

Subpackages

fobi.wizard.views package

Submodules

fobi.wizard.views.dynamic module

class fobi.wizard.views.dynamic.**DynamicWizardView**(***kwargs*)

Bases: django.views.generic.base.TemplateView

The WizardView is used to create multi-page forms.

Handles all the storage and validation stuff. The wizard is based on Django's generic class based views.

classmethod as_view(**args*, ***kwargs*)

As view.

This method is used within urls.py to create unique wizardview instances for every request. We need to override this method because we add some kwargs which are needed to make the wizardview usable.

compute_form_list(*form_list=None*, **args*, ***kwargs*)

Compute the forms list.

condition_dict = None

dispatch(*request*, **args*, ***kwargs*)

Dispatch.

This method gets called by the routing engine. The first argument is *request* which contains a *HttpRequest* instance. The request is stored in *self.request* for later use. The storage instance is stored in *self.storage*.

After processing the request using the *dispatch* method, the response gets updated by the storage engine (for example add cookies).

done(*form_list*, ***kwargs*)

Done.

This method must be overridden by a subclass to process to form data after processing all steps.

get(*request*, **args*, ***kwargs*)

GET requests.

This method handles GET requests.

If a GET request reaches this point, the wizard assumes that the user just starts at the first step or wants to restart the process. The data of the wizard will be resetted before rendering the first step

get_all_cleaned_data()

Get all cleaned data.

Returns a merged dictionary of all step cleaned_data dictionaries. If a step contains a *FormSet*, the key will be prefixed with 'formset-' and contain a list of the formset cleaned_data dictionaries.

get_cleaned_data_for_step(*step*)

Get clean data for step.

Returns the cleaned data for a given *step*. Before returning the cleaned data, the stored values are revalidated through the form. If the data doesn't validate, None will be returned.

get_context_data(*form*, ***kwargs*)
Get context data.

Returns the template context for a step. You can overwrite this method to add more data for all or some steps. This method returns a dictionary containing the rendered form step. Available template context variables are:

- all extra data stored in the storage backend
- *wizard* - a dictionary representation of the wizard instance

Example:

```
class MyWizard(WizardView):
    def get_context_data(self, form, **kwargs):
        context = super(MyWizard, self).get_context_data(form=form,
                                                         **kwargs)
        if self.steps.current == 'my_step_name':
            context.update({'another_var': True})
        return context
```

get_form(*step=None*, *data=None*, *files=None*)
Get the form.

Constructs the form for a given *step*. If no *step* is defined, the current step will be determined automatically.

The form will be initialized using the *data* argument to prefill the new form. If needed, instance or queryset (for *ModelForm* or *ModelFormSet*) will be added too.

get_form_initial(*step*)
Get form initial

Returns a dictionary which will be passed to the form for *step* as *initial*. If no initial data was provided while initializing the form wizard, an empty dictionary will be returned.

get_form_instance(*step*)
Get form instance.

Returns an object which will be passed to the form for *step* as *instance*. If no instance object was provided while initializing the form wizard, None will be returned.

get_form_kwargs(*step=None*)
Get form kwargs.

Returns the keyword arguments for instantiating the form (or formset) on the given step.

get_form_list()
Get form list.

This method returns a form_list based on the initial form list but checks if there is a condition method/value in the condition_list. If an entry exists in the condition list, it will call/read the value and respect the result. (True means add the form, False means ignore the form)

The form_list is always generated on the fly because condition methods could use data from other (maybe previous forms).

get_form_prefix(*step=None*, *form=None*)
Get form prefix.

Returns the prefix which will be used when calling the actual form for the given step. *step* contains the step-name, *form* the form which will be called with the returned prefix.

If no step is given, the form_prefix will determine the current step automatically.

get_form_step_data(*form*)
Get form step data.

Is used to return the raw form data. You may use this method to manipulate the data.

get_form_step_files(*form*)
Get form step files.

Is used to return the raw form files. You may use this method to manipulate the data.

get_initial_wizard_data(**args*, ***kwargs*)
This should be implemented in your subclass.

You are supposed to return a dict with the dynamic properties, such as *form_list* or *template_name*.

classmethod get_initkwargs(*form_list=None*, *initial_dict=None*, *instance_dict=None*, *condition_dict=None*, **args*, ***kwargs*)
Create a dict with all needed parameters.

For the form wizard instances.

- *form_list* - is a list of forms. The list entries can be single form classes or tuples of (*step_name*, *form_class*). If you pass a list of forms, the wizardview will convert the class list to (*zero_based_counter*, *form_class*). This is needed to access the form for a specific step.
- *initial_dict* - contains a dictionary of initial data dictionaries. The key should be equal to the *step_name* in the *form_list* (or the str of the zero based counter - if no step_names added in the *form_list*)
- *instance_dict* - contains a dictionary whose values are model instances if the step is based on a `ModelForm` and querysets if the step is based on a `ModelFormSet`. The key should be equal to the *step_name* in the *form_list*. Same rules as for *initial_dict* apply.
- *condition_dict* - contains a dictionary of boolean values or callables. If the value of for a specific *step_name* is callable it will be called with the wizardview instance as the only argument. If the return value is true, the step's form will be used.

get_next_step(*step=None*)
Get next step.

Returns the next step after the given *step*. If no more steps are available, None will be returned. If the *step* argument is None, the current step will be determined automatically.

get_prefix(*request*, **args*, ***kwargs*)
Get prefix.

get_prev_step(*step=None*)
Get previous step.

Returns the previous step before the given *step*. If there are no steps available, None will be returned. If the *step* argument is None, the current step will be determined automatically.

get_step_index(*step=None*)
Get step index.

Returns the index for the given *step* name. If no step is given, the current step will be used to get the index.

initial_dict = None

instance_dict = None

post(**args*, ***kwargs*)
POST requests.

This method handles POST requests.

The wizard will render either the current step (if form validation wasn't successful), the next step (if the current step was stored successful) or the done view (if no more steps are available)

process_step(*form*)
Process the step.

This method is used to post-process the form data. By default, it returns the raw *form.data* dictionary.

process_step_files(*form*)
Process step files.

This method is used to post-process the form files. By default, it returns the raw *form.files* dictionary.

render(*form=None*, ***kwargs*)
Render.

Returns a `HttpResponse` containing all needed context data.

render_done(*form*, ***kwargs*)
Render done.

This method gets called when all forms passed. The method should also re-validate all steps to prevent manipulation. If any form fails to validate, *render_revalidation_failure* should get called. If everything is fine call *done*.

render_goto_step(*goto_step*, ***kwargs*)
Render goto step.

This method gets called when the current step has to be changed. *goto_step* contains the requested step to go to.

render_next_step(*form*, ***kwargs*)
Render next step.

This method gets called when the next step/form should be rendered. *form* contains the last/current form.

render_revalidation_failure(*step*, *form*, ***kwargs*)
Render revalidation failure.

Gets called when a form doesn't validate when rendering the done view. By default, it changes the current step to failing forms step and renders the form.

storage_name = None

template_name = 'formtools/wizard/wizard_form.html'

class fobi.wizard.views.dynamic.**DynamicSessionWizardView**(***kwargs*)
Bases: *fobi.wizard.views.dynamic.DynamicWizardView*

A WizardView with pre-configured SessionStorage backend.

storage_name = 'formtools.wizard.storage.session.SessionStorage'

class fobi.wizard.views.dynamic.**DynamicCookieWizardView**(***kwargs*)
Bases: *fobi.wizard.views.dynamic.DynamicWizardView*

A WizardView with pre-configured CookieStorage backend.

storage_name = 'formtools.wizard.storage.cookie.CookieStorage'

class fobi.wizard.views.dynamic.**DynamicNamedUrlWizardView**(***kwargs*)
Bases: *fobi.wizard.views.dynamic.DynamicWizardView*

A WizardView with URL named steps support.

done_step_name = None

get (**args*, ***kwargs*)
GET request.

This renders the form or, if needed, does the http redirects.

get_context_data (*form*, ***kwargs*)
Get context data.

NamedUrlWizardView provides the url_name of this wizard in the context dict *wizard*.

classmethod get_initkwargs (**args*, ***kwargs*)
Get init kwargs.

We require a url_name to reverse URLs later. Additionally users can pass a done_step_name to change the URL name of the "done" view.

get_step_url (*step*)
Get step URL.

post (**args*, ***kwargs*)
POST request.

Do a redirect if user presses the prev. step button. The rest of this is super'd from WizardView.

render_done (*form*, ***kwargs*)
Render done.

When rendering the done view, we have to redirect first (if the URL name doesn't fit).

render_goto_step (*goto_step*, ***kwargs*)
Render goto step.

This method gets called when the current step has to be changed. *goto_step* contains the requested step to go to.

render_next_step (*form*, ***kwargs*)
Render next step.

When using the NamedUrlWizardView, we have to redirect to update the browser's URL to match the shown step.

render_revalidation_failure (*failed_step*, *form*, ***kwargs*)
Render revalidation failure.

When a step fails, we have to redirect the user to the first failing step.

url_name = None

class fobi.wizard.views.dynamic.**DynamicNamedUrlSessionWizardView** (***kwargs*)
Bases: *fobi.wizard.views.dynamic.DynamicNamedUrlWizardView*

A NamedUrlWizardView with pre-configured SessionStorage backend.

storage_name = 'formtools.wizard.storage.session.SessionStorage'

class fobi.wizard.views.dynamic.**DynamicNamedUrlCookieWizardView** (***kwargs*)
Bases: *fobi.wizard.views.dynamic.DynamicNamedUrlWizardView*

A NamedUrlFormWizard with pre-configured CookieStorageBackend.

storage_name = 'formtools.wizard.storage.cookie.CookieStorage'

fobi.wizard.views.views module

class fobi.wizard.views.views.**WizardView**(***kwargs*)
Bases: formtools.wizard.views.WizardView, fobi.wizard.views.views.PatchGetMixin

Patched version of the original WizardView.

get(*request*, **args*, ***kwargs*)
GET requests.

This method handles GET requests.

If a GET request reaches this point, the wizard assumes that the user just starts at the first step or wants to restart the process. The data of the wizard will be resetted before rendering the first step.

class fobi.wizard.views.views.**SessionWizardView**(***kwargs*)
Bases: formtools.wizard.views.SessionWizardView, fobi.wizard.views.views.PatchGetMixin

A WizardView with pre-configured SessionStorage backend.

get(*request*, **args*, ***kwargs*)
GET requests.

This method handles GET requests.

If a GET request reaches this point, the wizard assumes that the user just starts at the first step or wants to restart the process. The data of the wizard will be resetted before rendering the first step.

class fobi.wizard.views.views.**CookieWizardView**(***kwargs*)
Bases: formtools.wizard.views.CookieWizardView, fobi.wizard.views.views.PatchGetMixin

A WizardView with pre-configured CookieStorage backend.

get(*request*, **args*, ***kwargs*)
GET requests.

This method handles GET requests.

If a GET request reaches this point, the wizard assumes that the user just starts at the first step or wants to restart the process. The data of the wizard will be resetted before rendering the first step.

Module contents

Module contents

Submodules

fobi.admin module

fobi.admin.**base_bulk_change_plugins**(*PluginForm*, *named_url*, *modeladmin*, *request*, *queryset*)
Bulk change of plugins action additional view.

class fobi.admin.**BasePluginModelAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Base plugin admin.

class Meta
Bases: object

Meta.

app_label = <django.utils.functional.__proxy__ object>

BasePluginModelAdmin.**bulk_change_plugins**(**args*, ***kwargs*)
Bulk change plugins.

This is where the data is actually processed.

BasePluginModelAdmin.**fieldsets = ((None, {'fields': ('plugin_uid', 'users', 'groups')}),)**

BasePluginModelAdmin.**filter_horizontal = ('users', 'groups')**

BasePluginModelAdmin.**get_queryset**(*request*)
Internal method used in get_queryset or queryset methods.

BasePluginModelAdmin.**has_add_permission**(*request*)
Has add permissions.

We don't want to allow to add form elements/handlers manually. It should happen using the management command `fobi_sync_plugins` instead.

BasePluginModelAdmin.**list_display = ('plugin_uid_admin', 'users_list', 'groups_list')**

BasePluginModelAdmin.**media**

BasePluginModelAdmin.**readonly_fields = ('plugin_uid', 'plugin_uid_admin')**

fobi.admin.**bulk_change_form_element_plugins**(*modeladmin*, *request*, *queryset*)
Bulk change FormElement plugins.

fobi.admin.**bulk_change_form_handler_plugins**(*modeladmin*, *request*, *queryset*)
Bulk change FormHandler plugins.

fobi.admin.**bulk_change_form_wizard_handler_plugins**(*modeladmin*, *request*, *queryset*)
Bulk change FormWizardHandler plugins.

class fobi.admin.**FormElementAdmin**(*model*, *admin_site*)
Bases: *fobi.admin.BasePluginModelAdmin*

FormElement admin.

actions = [<function bulk_change_form_element_plugins at 0x7ff21ec2b1b8>]

get_urls()
Get URLs.

media

class fobi.admin.**FormElementEntryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

FormElementEntry admin.

class Meta
Bases: object

Meta.

app_label = <django.utils.functional.__proxy__ object>

FormElementEntryAdmin.**fieldsets = ((<django.utils.functional.__proxy__ object at 0x7ff21ec310d0>, {'fields': ('**

FormElementEntryAdmin.**get_queryset**(*request*)
Internal method used in get_queryset or queryset methods.

FormElementEntryAdmin.**list_display = ('plugin_uid', 'plugin_uid_code', 'plugin_data', 'position', 'form_entry**

FormElementEntryAdmin.**list_editable = ('position',)**

FormElementEntryAdmin.**list_filter = ('form_entry', 'plugin_uid')**

FormElementEntryAdmin.**media**

FormElementEntryAdmin.**readonly_fields = ('plugin_uid_code',)**

class fobi.admin.**FormElementEntryInlineAdmin**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

FormElementEntry inline admin.

extra = 0

fields = ('form_entry', 'plugin_uid', 'plugin_data', 'position')

form
alias of FormElementEntryForm

media

model
alias of FormElementEntry

class fobi.admin.**FormEntryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

FormEntry admin.

class Meta
Bases: object

Meta.

app_label = <django.utils.functional.__proxy__ object>

FormEntryAdmin.**fieldsets = ((<django.utils.functional.__proxy__ object at 0x7ff21ef58a10>, {'fields': ('name', 'is_**

FormEntryAdmin.**inlines = [<class 'fobi.admin.FormElementEntryInlineAdmin'>, <class 'fobi.admin.FormHandler**

FormEntryAdmin.**list_display = ('name', 'slug', 'user', 'is_public', 'created', 'updated', 'is_cloneable')**

FormEntryAdmin.**list_editable = ('is_public', 'is_cloneable')**

FormEntryAdmin.**list_filter = ('is_public', 'is_cloneable')**

FormEntryAdmin.**media**

FormEntryAdmin.**radio_fields = {'user': 2}**

FormEntryAdmin.**readonly_fields = ('slug',)**

class fobi.admin.**FormFieldsetEntryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

FormEieldsetEntry admin.

class Meta
Bases: object

Meta.

app_label = <django.utils.functional.__proxy__ object>

FormFieldsetEntryAdmin.**fieldsets = ((None, {'fields': ('form_entry', 'name', 'is_repeatable')}),)**

FormFieldsetEntryAdmin.**list_display = ('form_entry', 'name', 'is_repeatable')**

FormFieldsetEntryAdmin.**list_editable = ('is_repeatable',)**

FormFieldsetEntryAdmin.**list_filter = ('is_repeatable',)**

FormFieldsetEntryAdmin.**media**

class fobi.admin.**FormHandlerAdmin**(*model*, *admin_site*)
Bases: *fobi.admin.BasePluginModelAdmin*

FormHandler admin.

actions = [<function bulk_change_form_handler_plugins at 0x7ff21ec2b230>]

get_urls()
Get URLs.

media

class fobi.admin.**FormHandlerEntryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

FormHandlerEntry admin.

class Meta
Bases: object

Meta.

app_label = <django.utils.functional.__proxy__ object>

FormHandlerEntryAdmin.**fieldsets = ((<django.utils.functional.__proxy__ object at 0x7ff21ec31310>, {'fields': ('**

FormHandlerEntryAdmin.**get_queryset**(*request*)
Internal method used in get_queryset or queryset methods.

FormHandlerEntryAdmin.**list_display = ('plugin_uid', 'plugin_uid_code', 'plugin_data', 'form_entry')**

FormHandlerEntryAdmin.**list_filter = ('form_entry', 'plugin_uid')**

FormHandlerEntryAdmin.**media**

FormHandlerEntryAdmin.**readonly_fields = ('plugin_uid_code',)**

class fobi.admin.**FormHandlerEntryInlineAdmin**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

FormHandlerEntry inline admin.

extra = 0

fields = ('form_entry', 'plugin_uid', 'plugin_data')

form
alias of FormHandlerEntryForm

media

model
alias of FormHandlerEntry

class fobi.admin.**FormWizardEntryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

FormWizardEntry admin.

class Meta
Bases: object

Meta.

app_label = <django.utils.functional.__proxy__ object>

FormWizardEntryAdmin.**fieldsets** = ((<django.utils.functional.__proxy__ object at 0x7ff21ef58610>, {'fields': ('na

FormWizardEntryAdmin.**inlines** = [<class 'fobi.admin.FormWizardFormEntryInlineAdmin'>, <class 'fobi.admin.

FormWizardEntryAdmin.**list_display** = ('name', 'slug', 'user', 'is_public', 'created', 'updated', 'is_cloneable')

FormWizardEntryAdmin.**list_editable** = ('is_public', 'is_cloneable')

FormWizardEntryAdmin.**list_filter** = ('is_public', 'is_cloneable')

FormWizardEntryAdmin.**media**

FormWizardEntryAdmin.**radio_fields** = {'user': 2}

FormWizardEntryAdmin.**readonly_fields** = ('slug',)

class fobi.admin.**FormWizardFormEntryInlineAdmin**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

FormWizardFormEntry inline admin.

extra = 0

fields = ('form_entry', 'position')

media

model
alias of FormWizardFormEntry

class fobi.admin.**FormWizardHandlerAdmin**(*model*, *admin_site*)
Bases: *fobi.admin.BasePluginModelAdmin*

FormHandler admin.

actions = [<function bulk_change_form_wizard_handler_plugins at 0x7ff21ec2b2a8>]

get_urls()
Get URLs.

media

class fobi.admin.**FormWizardHandlerEntryInlineAdmin**(*parent_model*, *admin_site*)
Bases: django.contrib.admin.options.TabularInline

FormWizardHandlerEntry inline admin.

extra = 0

fields = ('plugin_uid', 'plugin_data')

form
alias of FormWizardHandlerEntryForm

media

model
alias of FormWizardHandlerEntry

fobi.app module

fobi.app.**app_name**(*path*, *reduce_depth_by=1*)
Return another path by reducing the depth by one.

Parameters

- **path** (*str*) – Absolute app path (from project root).

- **reduce_depth_by** (*int*) –

Return str

fobi.app.**app_config**(*path*, *config_app_path='apps.Config'*)
App config.

Parameters

- **path** (*str*) – Absolute app path (from project root).
- **config_app_path** (*str*) – Relative config path (from app root)

Return str

fobi.apps module

class fobi.apps.**Config**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

Config.

label = 'fobi'

name = 'fobi'

fobi.base module

Base module. All *uids* are supposed to be pythonic function names (see PEP http://www.python.org/dev/peps/pep-0008/#function-names).

fobi.base.**assemble_form_field_widget_class**(*base_class*, *plugin*)
Assemble form field widget class.

Finish this or remove.

#TODO

class fobi.base.**BaseDataStorage**
Bases: object

Base storage data.

class fobi.base.**BaseFormFieldPluginForm**
Bases: *fobi.base.BasePluginForm*

Base form for form field plugins.

help_text = <django.forms.fields.CharField object>

label = <django.forms.fields.CharField object>

name = <django.forms.fields.CharField object>

plugin_data_fields = [('name', ''), ('label', ''), ('help_text', ''), ('required', False)]

required = <django.forms.fields.BooleanField object>

validate_plugin_data(*form_element_entries*, *request=None*)
Validate plugin data.

Parameters

- **form_element_entries** (*iterable*) – Iterable of `fobi.models.FormElementEntry`.
- **request** (*django.http.HttpRequest*) –

Return bool

class `fobi.base.`**BasePlugin**(*user=None*)

Bases: `object`

Base plugin.

Base form field from which every form field should inherit.

Properties

- ***uid*** **(string): Plugin uid (obligatory). Example value: 'dummy',** 'wysiwyg', 'news'.
- ***name*** **(string): Plugin name (obligatory). Example value:** 'Dummy plugin', 'WYSIWYG', 'Latest news'.
- ***description*** **(string): Plugin decription (optional). Example** value: 'Dummy plugin used just for testing'.
- ***help_text*** **(string): Plugin help text (optional). This text would** be shown in `fobi.views.add_form_plugin_entry` and `fobi.views.edit_form_plugin_entry` views.
- ***form*****: Plugin form (optional). A subclass of `django.forms.Form`.** Should be given in case plugin is configurable.
- ***add_form_template*** **(str) (optional): Add form template (optional).** If given, overrides the

  fobi.views.add_form_handler_entry default template.
- ***edit_form_template*** **(string): Edit form template (optional). If** given, overrides the *fobi.views.edit_form_handler_entry* default template.
- *html_classes* (list): List of extra HTML classes for the plugin.
- ***group*** **(string): Plugin are grouped under the specified group.** Override in your plugin if necessary.

add_form_template = None

clone_plugin_data(*entry*)

Clone plugin data.

Used when copying entries. If any objects or files are created by plugin, they should be cloned.

Parameters **fobi.models.AbstractPluginEntry** – Instance of `fobi.models.AbstractPluginEntry`.

Return string JSON dumped string of the cloned plugin data. The returned value would be inserted as is into the *fobi.models.AbstractPluginEntry.plugin_data* field.

delete_plugin_data()

Delete plugin data (internal method).

Used in `fobi.views.delete_form_entry` and `fobi.views.delete_form_handler_entry`. Fired automatically, when `fobi.models.FormEntry` object is about to be deleted. Make use of it if your plugin creates database records or files that are not monitored externally but by fobi only.

description = None

edit_form_template = None

form = None

get_cloned_plugin_data(*update={}*)
Get cloned plugin data.

Get the cloned plugin data and returns it in a JSON dumped format.

Parameters **update** (*dict*) –

Return string JSON dumped string of the cloned plugin data.

Example

In the get_cloned_plugin_data method of your plugin, do as follows:

```
>>> def clone_plugin_data(self, dashboard_entry):
>>>     cloned_image = clone_file(self.data.image, relative_path=True)
>>>     return self.get_cloned_plugin_data(
>>>         update={'image': cloned_image}
>>>     )
```

get_form()
Get the plugin form class.

Override this method in your subclassed `fobi.base.BasePlugin` class when you need your plugin setup to vary depending on the placeholder, workspace, user or request given. By default returns the value of the `form` attribute defined in your plugin.

Return django.forms.Form|django.forms.ModelForm Subclass of `django.forms.Form` or `django.forms.ModelForm`.

get_initialised_create_form(*data=None*, *files=None*, *initial_data=None*)
Get initialized create form.

Used `fobi.views.add_form_element_entry` and `fobi.views.add_form_handler_entry` view to gets initialised form for object to be created.

get_initialised_create_form_or_404(*data=None*, *files=None*)
Get initialized create form or page 404.

Same as `get_initialised_create_form` but raises `django.http.Http404` on errors.

get_initialised_edit_form(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
Get initialized edit form.

Used in `fobi.views.edit_form_element_entry` and `fobi.views.edit_form_handler_entry` views.

get_initialised_edit_form_or_404(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*)
Get initialized edit form or page 404.

Same as `get_initialised_edit_form` but raises `django.http.Http404` on errors.

get_instance()
Get instance.

get_plugin_form_data()
Get plugin form data.

Fed as `initial` argument to the plugin form when initialising the instance for adding or editing the plugin. Override in your plugin class if you need customisations.

get_updated_plugin_data(*update={}*)
Get updated plugin data.

Returns it in a JSON dumped format.

Parameters **update** (*dict*) –

Return string JSON dumped string of the cloned plugin data.

get_widget(*request=None*, *as_instance=False*)
Get the plugin widget.

Parameters

- **request** (*django.http.HttpRequest*) –
- **as_instance** (*bool*) –

Return mixed Subclass of *fobi.base.BasePluginWidget* or instance of subclassed *fobi.base.BasePluginWidget* object.

group = <django.utils.functional.__proxy__ object>

help_text = None

html_class
HTML class.

A massive work on positioning the plugin and having it to be displayed in a given width is done here. We should be getting the plugin widget for the plugin given and based on its' properties (static!) as well as on plugin position (which we have from model), we can show the plugin with the exact class.

html_classes = []

html_id
HTML id.

load_plugin_data(*plugin_data*)
Load plugin data.

Load the plugin data saved in `fobi.models.FormElementEntry` or `fobi.models.FormHandlerEntry`. Plugin data is saved in JSON string.

Parameters **plugin_data** (*string*) – JSON string with plugin data.

media_css = []

media_js = []

name = None

plugin_data_repr()
Plugin data repr.

Human readable representation of plugin data. A very basic way would be just:

```
>>> return self.data.__dict__
```

Return string

post_processor()
Post-processor (self).

Redefine in your subclassed plugin when necessary.

Post process plugin data here (before rendering). This method is being called after the data has been loaded into the plugin.

Note, that request (django.http.HttpRequest) is available (self.request).

pre_processor()
Pre-processor (callback).

Redefine in your subclassed plugin when necessary.

Pre process plugin data (before rendering). This method is being called before the data has been loaded into the plugin.

Note, that request (django.http.HttpRequest) is available (self.request).

process(*plugin_data=None*, *fetch_related_data=False*)
Process.

Init plugin with data.

process_plugin_data(*fetch_related_data=False*)
Processes plugin data.

render(*request=None*)
Renders the plugin HTML.

Parameters **request** (*django.http.HttpRequest*) –

Return string

storage = None

uid = None

update_plugin_data(*entry*)
Update plugin data.

Used in `fobi.management.commands.fobi_update_plugin_data`.

Some plugins would contain data fetched from various sources (models, remote data). Since form entries are by definition loaded extremely much, you are advised to store as much data as possible in `plugin_data` field of `fobi.models.FormElementEntry` or `fobi.models.FormHandlerEntry`. Some externally fetched data becomes invalid after some time and needs updating. For that purpose, in case if your plugin needs that, redefine this method in your plugin. If you need your data to be periodically updated, add a cron-job which would run `fobi_update_plugin_data` management command (see `fobi.management.commands.fobi_update_plugin_data` module).

Parameters **or fobi.models.FormHandlerEntry** (fobi.models.FormElementEntry) – Instance of `fobi.models.FormeHandlerEntry`.

Return dict Should return a dictionary containing data of fields to be updated.

widget = None

class `fobi.base.`**BasePluginForm**
Bases: `object`

Not a form actually; defined for magic only.

Property iterable plugin_data_fields Fields to get when calling the `get_plugin_data` method. These field will be JSON serialized. All other fields, even if they are part of the form, won't be. Make sure all fields are serializable. If some of them aren't, override the `save_plugin_data` method and make them serializable there. See *fobi.contrib.plugins.form_elements.fields.select.forms* as a good example.

Example

```
>>> plugin_data_fields = (
>>>     ('name', ''),
>>>     ('active': False)
>>> )
```

get_plugin_data(*request=None*, *json_format=True*)
Get plugin data.

Data that would be saved in the `plugin_data` field of the `fobi.models.FormElementEntry` or ‘‘fobi.models.FormHandlerEntry‘.‘ subclassed model.

Parameters **request** (*django.http.HttpRequest*) –

plugin_data_fields = None

save_plugin_data(*request=None*)
Save plugin data.

Dummy, but necessary.

validate_plugin_data(*form_element_entries*, *request=None*)
Validate plugin data.

Parameters

- **form_element_entries** (*iterable*) – Iterable of `fobi.models.FormElementEntry`.
- **request** (*django.http.HttpRequest*) –

Return bool

class `fobi.base.`**BaseRegistry**
Bases: `object`

Base registry.

Registry of plugins. It's essential, that class registered has the `uid` property.

If `fail_on_missing_plugin` is set to True, an appropriate exception (`plugin_not_found_exception_cls`) is raised in cases if plugin could't be found in the registry.

Property mixed type

Property bool fail_on_missing_plugin

Property fobi.exceptions.DoesNotExist plugin_not_found_exception_cls

Property str plugin_not_found_error_message

fail_on_missing_plugin = False

get(*uid*, *default=None*)
Get the given entry from the registry.

Parameters

- **uid** (*string*) –

- **default** (*mixed*) –

:return mixed.

items()
Shortcut to self._registry.items().

plugin_not_found_error_message = "Can't find plugin with uid '{0}' in '{1}' registry."

plugin_not_found_exception_cls
alias of DoesNotExist

register(*cls*, *force=False*)
Registers the plugin in the registry.

Parameters

- **cls** (*mixed*) –
- **force** (*bool*) –

registry
Shortcut to self._registry.

type = None

unregister(*cls*)
Un-register.

class fobi.base.**ClassProperty**
Bases: property

ClassProperty.

fobi.base.**classproperty**
alias of *ClassProperty*

fobi.base.**collect_plugin_media**(*form_element_entries*, *request=None*)
Collect the plugin media for form element entries given.

Parameters

- **form_element_entries** (*iterable*) – Iterable of fobi.models.FormElementEntry instances.
- **request** (*django.http.HttpRequest*) –

Return dict Returns a dict containing the 'js' and 'css' keys. Correspondent values of those keys are lists containing paths to the CSS and JS media files.

fobi.base.**ensure_autodiscover**()
Ensure that plugins are auto-discovered.

The form callbacks registry is intentionally left out, since they will be auto-discovered in any case if other modules are discovered.

fobi.base.**fire_form_callbacks**(*form_entry*, *request*, *form*, *stage=None*)
Fire form callbacks.

Parameters

- **form_entry** (fobi.models.FormEntry) –
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

- **stage** (*string*) –

Return django.forms.Form form

class fobi.base.**FormCallback**
Bases: object

Base form callback.

callback(*form_entry*, *request*, *form*)
Callback.

Custom callback code should be implemented here.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of fobi.models.FormEntry.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –

stage = None

class fobi.base.**FormCallbackRegistry**
Bases: object

Registry of callbacks.

Holds callbacks for stages listed in the fobi.constants.CALLBACK_STAGES.

get_callbacks(*stage=None*)
Get callbacks for the stage given.

Parameters **stage** (*string*) –

Return list

register(*cls*)
Registers the plugin in the registry.

Parameters **cls** (*mixed*) –

uidfy(*cls*)
Makes a UID string from the class given.

Parameters **cls** (*mixed*) –

Return string

class fobi.base.**FormElementPlugin**(*user=None*)
Bases: *fobi.base.BasePlugin*

Base form element plugin.

Property fobi.base.FormElementPluginDataStorage storage

Property bool has_value If set to False, ignored (removed) from the POST when processing the form.

get_form_field_instances(*request=None*, *form_entry=None*, *form_element_entries=None*, ***kwargs*)
Get the instances of form fields, that plugin contains.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry** (fobi.models.FormEntry) –

- **form_element_entries** (*django.db.models.QuerySet*) – Queryset of *fobi.models.FormElementEntry* instances.

Return list List of Django form field instances.

Example

```
>>> from django.forms.fields import CharField, IntegerField, TextField
>>> [CharField(max_length=100), IntegerField(), TextField()]
```

get_origin_kwargs_update_func_results(*kwargs_update_func*, *form_element_entry*, *origin*, *extra={}*, *widget_cls=None*)

Get origin kwargs update func results.

If `kwargs_update_func` is given, is callable and returns results without failures, return the result. Otherwise - return None.

get_origin_return_func_results(*return_func*, *form_element_entry*, *origin*)

Get origin return func results.

If `return_func` is given, is callable and returns results without failures, return the result. Otherwise - return None.

has_value = False

is_hidden = False

storage

alias of *FormElementPluginDataStorage*

submit_plugin_form_data(*form_entry*, *request*, *form*, *form_element_entries=None*, ***kwargs*)

Submit plugin form data.

Called on form submission (when user actually posts the data to assembled form).

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of `fobi.models.FormEntry`.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –
- **form_element_entries** (*iterable*) –

class fobi.base.**FormElementPluginDataStorage**

Bases: *fobi.base.BaseDataStorage*

Storage for *FormElementPlugin* data.

class fobi.base.**FormElementPluginRegistry**

Bases: *fobi.base.BaseRegistry*

Form element plugins registry.

fail_on_missing_plugin = True

plugin_not_found_exception_cls

alias of `FormElementPluginDoesNotExist`

type = (<class 'fobi.base.FormElementPlugin'>, <class 'fobi.base.FormFieldPlugin'>)

class fobi.base.**FormElementPluginWidget**(*plugin*)

Bases: `fobi.base.BasePluginWidget`

Form element plugin widget.

storage
alias of `FormElementPluginWidgetDataStorage`

class `fobi.base.`**FormElementPluginWidgetRegistry**
Bases: `fobi.base.BasePluginWidgetRegistry`

Registry of form element plugins.

type
alias of *FormElementPluginWidget*

class `fobi.base.`**FormFieldPlugin**(*user=None*)
Bases: *fobi.base.FormElementPlugin*

Form field plugin.

has_value = True

class `fobi.base.`**FormHandlerPlugin**(*user=None*)
Bases: *fobi.base.BasePlugin*

Form handler plugin.

Property fobi.base.FormHandlerPluginDataStorage storage

Property bool allow_multiple If set to True, plugin can be used multiple times within (per form). Otherwise - just once.

allow_multiple = True

custom_actions(*form_entry*, *request=None*)
Custom actions.

Override this method in your form handler if you want to specify custom actions. Note, that expected return value of this method is an iterable with a triple, where the first item is the URL of the action and the second item is the action title and the third item is the icon class of the action.

Example

```
>>> return (
>>>     ('/add-to-favorites/',
>>>      'Add to favourites',
>>>      'glyphicon glyphicon-favourties'),
>>> )
```

get_custom_actions(*form_entry*, *request=None*)
Internal method to for obtaining the `get_custom_actions`.

run(*form_entry*, *request*, *form*, *form_element_entries=None*)
Run.

Custom code should be implemented here.

Parameters

- **form_entry** (fobi.models.FormEntry) – Instance of `fobi.models.FormEntry`.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –
- **form_element_entries** (*iterable*) – Iterable of `fobi.models.FormElementEntry` objects.

Return mixed May be a tuple (bool, mixed) or None

storage
alias of *FormHandlerPluginDataStorage*

class fobi.base.**FormHandlerPluginDataStorage**
Bases: *fobi.base.BaseDataStorage*

Storage for *FormHandlerPlugin* data.

class fobi.base.**FormHandlerPluginRegistry**
Bases: *fobi.base.BaseRegistry*

Form handler plugins registry.

fail_on_missing_plugin = True

plugin_not_found_exception_cls
alias of FormHandlerPluginDoesNotExist

type
alias of *FormHandlerPlugin*

class fobi.base.**FormHandlerPluginWidget**(*plugin*)
Bases: fobi.base.BasePluginWidget

Form handler plugin widget.

storage
alias of FormHandlerPluginWidgetDataStorage

class fobi.base.**FormHandlerPluginWidgetRegistry**
Bases: fobi.base.BasePluginWidgetRegistry

Registry of form handler plugins.

type
alias of *FormHandlerPluginWidget*

class fobi.base.**FormWizardHandlerPlugin**(*user=None*)
Bases: *fobi.base.BasePlugin*

Form wizard handler plugin.

Property fobi.base.FormWizardHandlerPluginDataStorage storage

Property bool allow_multiple If set to True, plugin can be used multiple times within (per form). Otherwise - just once.

DONE

allow_multiple = True

custom_actions(*form_wizard_entry*, *request=None*)
Custom actions.

Override this method in your form handler if you want to specify custom actions. Note, that expected return value of this method is an iterable with a triple, where the first item is the URL of the action and the second item is the action title and the third item is the icon class of the action.

Example

```
>>>  return (
>>>      ('/add-to-favorites/',
>>>       'Add to favourites',
>>>       'glyphicon glyphicon-favourties'),
>>>  )
```

get_custom_actions (*form_wizard_entry*, *request=None*)
Internal method to for obtaining the `get_custom_actions`.

run (*form_wizard_entry*, *request*, *form_list*, *form_wizard*, *form_element_entries=None*)
Run.

Custom code should be implemented here.

Parameters

- **form_wizard_entry** (fobi.models.FormEntry) – Instance of `fobi.models.FormEntry`.
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –
- **form_element_entries** (*iterable*) – Iterable of `fobi.models.FormElementEntry` objects.

Return mixed May be a tuple (bool, mixed) or None

storage
alias of *FormWizardHandlerPluginDataStorage*

class fobi.base.**FormWizardHandlerPluginDataStorage**
Bases: *fobi.base.BaseDataStorage*

Storage for *FormWizardHandlerPlugin* handler data.

class fobi.base.**FormWizardHandlerPluginRegistry**
Bases: *fobi.base.BaseRegistry*

Form wizard handler plugins registry.

fail_on_missing_plugin = True

plugin_not_found_exception_cls
alias of `FormWizardHandlerPluginDoesNotExist`

type
alias of *FormWizardHandlerPlugin*

class fobi.base.**FormWizardHandlerPluginWidget** (*plugin*)
Bases: `fobi.base.BasePluginWidget`

Form wizard handler plugin widget.

storage
alias of `FormWizardHandlerPluginWidgetDataStorage`

class fobi.base.**FormWizardHandlerPluginWidgetRegistry**
Bases: `fobi.base.BasePluginWidgetRegistry`

Registry of form wizard handler plugins.

type
alias of *FormWizardHandlerPluginWidget*

fobi.base.**get_form_element_plugin_widget** (*plugin_uid*, *request=None*, *as_instance=False*, *theme=None*)
Get the form element plugin widget for the `plugin_uid` given.

Parameters

- **plugin_uid** (*str*) – UID of the plugin to get the widget for.

- **request** (*django.http.HttpRequest*) –
- **as_instance** (*bool*) –
- **theme** (*fobi.base.BaseTheme*) – Subclass of.

Return BasePluginWidget Subclass of.

fobi.base.**get_form_handler_plugin_widget**(*plugin_uid*, *request=None*, *as_instance=False*, *theme=None*)

Get the form handler plugin widget for the `plugin_uid` given.

Parameters

- **plugin_uid** (*str*) – UID of the plugin to get the widget for.
- **request** (*django.http.HttpRequest*) –
- **as_instance** (*bool*) –
- **theme** (*fobi.base.BaseTheme*) – Subclass of.

Return BasePluginWidget Subclass of.

fobi.base.**get_form_wizard_handler_plugin_widget**(*plugin_uid*, *request=None*, *as_instance=False*, *theme=None*)

Get the form wizard handler plugin widget for the `plugin_uid` given.

Parameters

- **plugin_uid** (*str*) – UID of the plugin to get the widget for.
- **request** (*django.http.HttpRequest*) –
- **as_instance** (*bool*) –
- **theme** (*fobi.base.BaseTheme*) – Subclass of.

Return BasePluginWidget Subclass of.

fobi.base.**get_plugin_widget**(*registry*, *plugin_uid*, *request=None*, *as_instance=False*, *theme=None*)

Get the plugin widget for the `plugin_uid` given.

Looks up in the `registry` provided.

Parameters

- **registry** (*fobi.base.BasePluginWidgetRegistry*) – Subclass of.
- **plugin_uid** (*str*) – UID of the plugin to get the widget for.
- **request** (*django.http.HttpRequest*) –
- **as_instance** (*bool*) –
- **theme** (*fobi.base.BaseTheme*) – Subclass of.

Return BasePluginWidget Subclass of.

fobi.base.**get_processed_form_data**(*form*, *form_element_entries*)

Gets processed form data.

Simply fires both `fobi.base.get_cleaned_data` and `fobi.base.get_field_name_to_label_map` functions and returns the result.

Parameters

- **form** (*django.forms.Form*) –

- **form_element_entries** (*iterable*) – Iterable of form element entries.

Return tuple

fobi.base.**get_processed_form_wizard_data**(*form_wizard*, *form_list*, *form_element_entries*)
Get processed form wizard data.

fobi.base.**get_registered_form_callbacks**(*stage=None*)
Get registered form callbacks for the stage given.

fobi.base.**get_registered_form_element_plugin_uids**(*flattern=True*)
Get registered form element plugin uids.

Gets a list of registered plugins in a form if tuple (plugin name, plugin description). If not yet auto-discovered, auto-discovers them.

Return list

fobi.base.**get_registered_form_element_plugins**()
Get registered form element plugins.

Gets a list of registered plugins in a form if tuple (plugin name, plugin description). If not yet autodiscovered, autodiscovers them.

Return list

fobi.base.**get_registered_form_handler_plugin_uids**(*flattern=True*)
Get registered form handler plugin uids.

Gets a list of UIDs of registered form handler plugins. If not yet auto-discovered, auto-discovers them.

Return list

fobi.base.**get_registered_form_handler_plugins**(*as_instances=False*)
Get registered form handler plugins.

Gets a list of registered plugins in a form of tuple (plugin name, plugin description). If not yet auto-discovered, auto-discovers them.

Return list

fobi.base.**get_registered_form_wizard_handler_plugin_uids**(*flattern=True*)
Get registered form handler plugin uids.

Gets a list of UIDs of registered form wizard handler plugins. If not yet auto-discovered, auto-discovers them.

Return list

fobi.base.**get_registered_form_wizard_handler_plugins**(*as_instances=False*)
Get registered form handler wizard plugins.

Gets a list of registered plugins in a form of tuple (plugin name, plugin description). If not yet auto-discovered, auto-discovers them.

Return list

fobi.base.**get_registered_plugin_uids**(*registry*, *flattern=True*, *sort_items=True*)
Get a list of registered plugin uids as a list .

If not yet auto-discovered, auto-discovers them.

The *sort_items* is applied only if *flattern* is True.

Parameters

- **registry** –

- **flattern** (*bool*) –
- **sort_items** (*bool*) –

Return list

fobi.base.**get_registered_plugins**(*registry*, *as_instances=False*, *sort_items=True*)
Get registered plugins.

Get a list of registered plugins in a form if tuple (plugin name, plugin description). If not yet auto-discovered, auto-discovers them.

Parameters

- **registry** –
- **as_instances** (*bool*) –
- **sort_items** (*bool*) –

Return list

fobi.base.**get_registered_theme_uids**(*flattern=True*)
Get registered theme uids.

Gets a list of registered themes in a form of tuple (plugin name, plugin description). If not yet auto-discovered, auto-discovers them.

Return list

fobi.base.**get_registered_themes**()
Get registered themes.

Gets a list of registered themes in form of tuple (plugin name, plugin description). If not yet auto-discovered, auto-discovers them.

Return list

fobi.base.**run_form_handlers**(*form_entry*, *request*, *form*, *form_element_entries=None*)
Run form handlers.

Parameters

- **form_entry** (fobi.models.FormEntry) –
- **request** (*django.http.HttpRequest*) –
- **form** (*django.forms.Form*) –
- **form_element_entries** (*iterable*) –

Return tuple List of success responses, list of error responses

fobi.base.**run_form_wizard_handlers**(*form_wizard_entry*, *request*, *form_list*, *form_wizard*, *form_element_entries=None*)
Run form wizard handlers.

Parameters

- **form_wizard_entry** (fobi.models.FormWizardEntry) –
- **request** (*django.http.HttpRequest*) –
- **form_list** (*list*) – List of `django.forms.Form` objects.
- **form_wizard** (fobi.wizard.views.dynamic.DynamicWizardView) – The form wizard view object.

- **form_element_entries** (*iterable*) – Iterable of fobi.base.FormElementEntry objects.

Return tuple List of success responses, list of error responses

fobi.base.**validate_form_element_plugin_uid**(*plugin_uid*)
Validate the form element plugin uid.

Parameters **plugin_uid** (*string*) –

Return bool

fobi.base.**validate_form_handler_plugin_uid**(*plugin_uid*)
Validate the plugin uid.

Parameters **plugin_uid** (*string*) –

Return bool

fobi.base.**validate_form_wizard_handler_plugin_uid**(*plugin_uid*)
Validate the plugin uid.

Parameters **plugin_uid** (*string*) –

Return bool

fobi.base.**validate_theme_uid**(*plugin_uid*)
Validate the theme uid.

Parameters **plugin_uid** (*string*) –

Return bool

fobi.compat module

class fobi.compat.**User**(**args*, ***kwargs*)
Bases: django.contrib.auth.models.AbstractUser

Users within the Django authentication system are represented by this model.

Username, password and email are required. Other fields are optional.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception User.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

User.**formelement_set**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

User.**formentry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`User.`**`formhandler_set`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`User.`**`formwizardentry_set`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`User.`**`formwizardhandler_set`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`User.`**`get_next_by_date_joined`**(**moreargs*, ***morekwargs*)

`User.`**`get_previous_by_date_joined`**(**moreargs*, ***morekwargs*)

`User.`**`groups`**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

User.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

User.**logentry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

User.**registrationprofile**
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`place.restaurant` is a `ReverseOneToOneDescriptor` instance.

User.**savedformdataentry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

User.**savedformwizarddataentry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

User.**user_permissions**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

fobi.conf module

fobi.conf.**get_setting**(*setting*, *override=None*)
Get setting.

Get a setting from *fobi* conf module, falling back to the default.

If override is not None, it will be used instead of the setting.

Parameters

- **setting** – String with setting name
- **override** – Value to use when no setting is available. Defaults to None.

Returns Setting value.

fobi.constants module

fobi.context_processors module

fobi.context_processors.**dynamic_values**(*request*)
Dynamic values exposed to public forms.

fobi.context_processors.**form_importers**(*request*)
Form importers.

fobi.context_processors.**theme**(*request*)
Get active theme.

Parameters **request** (*django.http.HttpRequest*) –

Return fobi.base.BaseTheme Instance of fobi.base.BaseTheme.

fobi.data_structures module

class fobi.data_structures.**SortableDict**(*data=None*)
Bases: dict

SortableDict.

A dictionary that keeps its keys in the order in which they're inserted. Very similar to (and partly based on) SortedDict of the Django, but has several additional methods implemented, such as: insert_before_key and insert_after_key.

clear()
Clear.

copy()
Returns a copy of this object.

insert(*index*, *key*, *value*)
Inserts the key, value pair before the item with the given index.

insert_after_key(*target_key*, *key*, *value*, *fail_silently=True*)
Insert the {key: value} after the target_key.

Parameters

- **target_key** (*immutable*) –

- **key** (*immutable*) –
- **value** (*mutable*) –
- **fail_silently** (boolean) –
- **offset** (*int*) –

Return bool

insert_before_key (*target_key*, *key*, *value*, *fail_silently=True*, *offset=0*)
Insert the {key: value} before the target_key.

Parameters

- **target_key** (*immutable*) –
- **key** (*immutable*) –
- **value** (*mutable*) –
- **fail_silently** (boolean) –
- **offset** (*int*) –

Return bool

items ()

iteritems ()
Iter items (internal method).

iterkeys ()

itervalues ()
Iter values (internal method).

keys ()

move_after_key (*source_key*, *target_key*, *fail_silently=True*)
Move the {key: value} after the given source_key.

Parameters

- **source_key** (*immutable*) –
- **target_key** (*immutable*) –
- **fail_silently** (boolean) –

Return bool

move_before_key (*source_key*, *target_key*, *fail_silently=True*, *offset=0*)
Move the {key: value} before the given source_key.

Parameters

- **source_key** (*immutable*) –
- **target_key** (*immutable*) –
- **fail_silently** (boolean) –
- **offset** (*int*) –

Return bool

pop (*k*, **args*)
Pop.

popitem()
Pop item.

setdefault(*key*, *default*)
Set default.

update(*dict_*)
Update.

value_for_index(*index*)
Returns the value of the item at the given zero-based index.

values()

fobi.decorators module

fobi.decorators.**permissions_required**(*perms*, *satisfy='all'*, *login_url=None*, *raise_exception=False*)
Check for the permissions given based on the strategy chosen.

Parameters

- **perms** (*iterable*) –
- **satisfy** (*string*) – Allowed values are "all" and "any".
- **login_url** (*string*) –
- **raise_exception** (*bool*) – If set to True, the `PermissionDenied` exception is raised on failures.

Return bool

Example

```
>>> @login_required
>>> @permissions_required(satisfy='any', perms=[
>>>     'fobi.add_formentry',
>>>     'fobi.change_formentry',
>>>     'fobi.delete_formentry',
>>>     'fobi.add_formelemententry',
>>>     'fobi.change_formelemententry',
>>>     'fobi.delete_formelemententry',
>>> ])
>>> def edit_dashboard(request):
>>>     # your code
```

fobi.decorators.**all_permissions_required**(*perms*, *login_url=None*, *raise_exception=False*)
Check for the permissions given based on SATISFY_ALL strategy chosen.

Example

```
>>> @login_required
>>> @all_permissions_required([
>>>     'fobi.add_formentry',
>>>     'fobi.change_formentry',
>>>     'fobi.delete_formentry',
>>>     'fobi.add_formelemententry',
>>>     'fobi.change_formelemententry',
>>>     'fobi.delete_formelemententry',
>>> ])
```

```
>>> def edit_dashboard(request):
>>>     # your code
```

fobi.decorators.**any_permission_required**(*perms*, *login_url=None*, *raise_exception=False*)
Check for the permissions given based on SATISFY_ANY strategy chosen.

Example

```
>>> @login_required
>>> @any_permission_required([
>>>     'fobi.add_formentry',
>>>     'fobi.change_formentry',
>>>     'fobi.delete_formentry',
>>>     'fobi.add_formelemententry',
>>>     'fobi.change_formelemententry',
>>>     'fobi.delete_formelemententry',
>>> ])
>>> def edit_dashboard(request):
>>>     # your code
```

fobi.defaults module

fobi.discover module

fobi.discover.**autodiscover**()
Auto-discovers files that should be found by fobi.

fobi.dynamic module

fobi.dynamic.**assemble_form_class**(*form_entry*, *base_class=<class 'django.forms.forms.BaseForm'>*, *request=None*, *origin=None*, *origin_kwargs_update_func=None*, *origin_return_func=None*, *form_element_entries=None*, *get_form_field_instances_kwargs={}*)
Assemble a form class by given entry.

Parameters

- **form_entry** –
- **base_class** –
- **request** (*django.http.HttpRequest*) –
- **origin** (*string*) –
- **origin_kwargs_update_func** (*callable*) –
- **origin_return_func** (*callable*) –
- **form_element_entries** (*iterable*) – If given, used instead of `form_entry.formelemententry_set.all` (no additional database hit).
- **get_form_field_instances_kwargs** (*dict*) – To be passed as **kwargs to the **:method:'get_form_field_instances_kwargs'**.

fobi.dynamic.**assemble_form_wizard_class**(*form_wizard_entry*, *base_class=<class 'formtools.wizard.views.SessionWizardView'>*, *request=None*, *origin=None*, *origin_kwargs_update_func=None*, *origin_return_func=None*, *form_wizard_form_entries=None*, *template_name=None*)

Assemble form wizard class.

fobi.exceptions module

exception fobi.exceptions.**BaseException**
Bases: exceptions.Exception

Base exception.

exception fobi.exceptions.**ImproperlyConfigured**
Bases: *fobi.exceptions.BaseException*

Improperly configured.

Exception raised when developer didn't configure/write the code properly.

exception fobi.exceptions.**InvalidRegistryItemType**
Bases: exceptions.ValueError, *fobi.exceptions.BaseException*

Invalid registry item type.

Raised when an attempt is made to register an item in the registry which does not have a proper type.

exception fobi.exceptions.**DoesNotExist**
Bases: *fobi.exceptions.BaseException*

Raised when something does not exist.

exception fobi.exceptions.**ThemeDoesNotExist**
Bases: *fobi.exceptions.DoesNotExist*

Raised when no theme with given uid can be found.

exception fobi.exceptions.**PluginDoesNotExist**
Bases: *fobi.exceptions.DoesNotExist*

Raised when no plugin with given uid can be found.

exception fobi.exceptions.**FormElementPluginDoesNotExist**
Bases: *fobi.exceptions.PluginDoesNotExist*

Raised when no form element plugin with given uid can be found.

exception fobi.exceptions.**FormHandlerPluginDoesNotExist**
Bases: *fobi.exceptions.PluginDoesNotExist*

Raised when no form handler plugin with given uid can be found.

exception fobi.exceptions.**FormWizardHandlerPluginDoesNotExist**
Bases: *fobi.exceptions.PluginDoesNotExist*

FormWizardHandlerPlugin does not exist.

Raised when no form wizard handler plugin with given uid can be found.

exception fobi.exceptions.**NoDefaultThemeSet**
Bases: *fobi.exceptions.ImproperlyConfigured*

Raised when no active theme is chosen.

exception fobi.exceptions.**FormPluginError**
Bases: *fobi.exceptions.BaseException*

Base error for form elements and handlers.

exception fobi.exceptions.**FormElementPluginError**
Bases: *fobi.exceptions.FormPluginError*

Raised when form element plugin error occurs.

exception fobi.exceptions.**FormHandlerPluginError**
Bases: *fobi.exceptions.FormPluginError*

Raised when form handler plugin error occurs.

exception fobi.exceptions.**FormCallbackError**
Bases: *fobi.exceptions.FormPluginError*

Raised when form callback error occurs.

fobi.form_importers module

class fobi.form_importers.**BaseFormImporter**(*form_entry_cls*, *form_element_entry_cls*, *form_properties=None*, *form_data=None*)
Bases: object

Base importer.

description = None

extract_field_properties(*field_data*)
Extract field properties.

field_properties_mapping = None

field_type_prop_name = None

fields_mapping = None

get_form_data()
Get form data.

get_template_names()
Get template names.

get_wizard(*request*, **args*, ***kwargs*)
Get wizard.

import_data(*form_properties*, *form_data*)
Import data.

name = None

position_prop_name = None

templates = None

uid = None

wizard = None

class fobi.form_importers.**FormImporterPluginRegistry**
Bases: *fobi.base.BaseRegistry*

Form importer plugins registry.

type
alias of *BaseFormImporter*

fobi.form_importers.**ensure_autodiscover**()
Ensure that form importer plugins are auto-discovered.

fobi.form_importers.**get_form_importer_plugin_uids**()
Get form importer plugin uids.

fobi.form_importers.**get_form_impoter_plugin_urls**()
Gets the form importer plugin URLs as a list of tuples.

fobi.form_utils module

class fobi.form_utils.**ErrorDict**
Bases: django.forms.utils.ErrorDict

A better ErrorDict.

as_text()
As text.

class fobi.form_utils.**ErrorList**(*initlist=None*, *error_class=None*)
Bases: django.forms.utils.ErrorList

A better ErrorList.

as_text()
As text.

fobi.forms module

class fobi.forms.**BulkChangeFormElementPluginsForm**(**args*, ***kwargs*)
Bases: fobi.forms.BaseBulkChangePluginsForm

Bulk change form element plugins form.

class Meta
Bases: object

Meta class.

fields = ['groups', 'groups_action', 'users', 'users_action']

model
alias of FormElement

BulkChangeFormElementPluginsForm.**base_fields = OrderedDict([('groups', <django.forms.models.ModelMu**

BulkChangeFormElementPluginsForm.**declared_fields = OrderedDict([('selected_plugins', <django.forms.fie**

BulkChangeFormElementPluginsForm.**media**

class fobi.forms.**BulkChangeFormHandlerPluginsForm**(**args*, ***kwargs*)
Bases: fobi.forms.BaseBulkChangePluginsForm

Bulk change form handler plugins form.

class Meta
Bases: object

Meta class.

fields = ['groups', 'groups_action', 'users', 'users_action']

model
alias of FormHandler

BulkChangeFormHandlerPluginsForm.**base_fields = OrderedDict([('groups', <django.forms.models.ModelMu**

BulkChangeFormHandlerPluginsForm.**declared_fields = OrderedDict([('selected_plugins', <django.forms.fie**

BulkChangeFormHandlerPluginsForm.**media**

class fobi.forms.**BulkChangeFormWizardHandlerPluginsForm**(**args*, ***kwargs*)
Bases: fobi.forms.BaseBulkChangePluginsForm

Bulk change form wizard handler plugins form.

class Meta
Bases: object

Meta class.

fields = ['groups', 'groups_action', 'users', 'users_action']

model
alias of FormWizardHandler

BulkChangeFormWizardHandlerPluginsForm.**base_fields = OrderedDict([('groups', <django.forms.models.M**

BulkChangeFormWizardHandlerPluginsForm.**declared_fields = OrderedDict([('selected_plugins', <django.**

BulkChangeFormWizardHandlerPluginsForm.**media**

fobi.forms.**FormElementEntryFormSet**
alias of FormElementEntryFormFormSet

class fobi.forms.**FormEntryForm**(**args*, ***kwargs*)
Bases: django.forms.models.ModelForm

Form for fobi.models.FormEntry model.

class Meta
Bases: object

Meta class.

fields = ('name', 'title', 'is_public', 'success_page_title', 'success_page_message', 'action')

model
alias of FormEntry

FormEntryForm.**base_fields = OrderedDict([('name', <django.forms.fields.CharField object at 0x7ff21ef630d0>), (**

FormEntryForm.**clean_action**()
Validate the action (URL).

Checks if URL exists.

FormEntryForm.**declared_fields = OrderedDict()**

FormEntryForm.**media**

class fobi.forms.**FormFieldsetEntryForm**(**args*, ***kwargs*)
Bases: django.forms.models.ModelForm

Form for fobi.models.FormFieldsetEntry model.

class Meta
Bases: object

Meta class.

fields = ('name',)

model
alias of FormFieldsetEntry

FormFieldsetEntryForm.**base_fields = OrderedDict([('name', <django.forms.fields.CharField object at 0x7ff21**

FormFieldsetEntryForm.**declared_fields = OrderedDict()**

FormFieldsetEntryForm.**media**

class fobi.forms.**FormHandlerEntryForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*)
Bases: django.forms.models.ModelForm

FormHandlerEntry form.

class Meta
Bases: object

Meta class.

fields = ('form_entry', 'plugin_data', 'plugin_uid')

model
alias of FormHandlerEntry

FormHandlerEntryForm.**base_fields = OrderedDict([('form_entry', <django.forms.models.ModelChoiceField ob**

FormHandlerEntryForm.**declared_fields = OrderedDict([('plugin_uid', <django.forms.fields.ChoiceField object**

FormHandlerEntryForm.**media**

class fobi.forms.**FormHandlerForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*)
Bases: django.forms.models.ModelForm

FormHandler form.

class Meta
Bases: object

Meta class.

fields = ('users', 'groups')

model
alias of FormHandler

FormHandlerForm.**base_fields = OrderedDict([('users', <django.forms.models.ModelMultipleChoiceField object a**

FormHandlerForm.**declared_fields = OrderedDict()**

FormHandlerForm.**media**

class fobi.forms.**FormWizardEntryForm**(**args*, ***kwargs*)
Bases: django.forms.models.ModelForm

Form for fobi.models.FormWizardEntry model.

class Meta
Bases: object

Meta class.

fields = ('name', 'title', 'is_public', 'success_page_title', 'success_page_message', 'show_all_navigation_buttons')

model
alias of FormWizardEntry

FormWizardEntryForm.**base_fields = OrderedDict([('name', <django.forms.fields.CharField object at 0x7ff21ec0**

FormWizardEntryForm.**declared_fields = OrderedDict()**

FormWizardEntryForm.**media**

class fobi.forms.**FormWizardFormEntryForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*)
Bases: django.forms.models.ModelForm

FormWizardFormEntryForm form.

class Meta
Bases: object

Meta class.

fields = ('form_wizard_entry', 'form_entry')

model
alias of FormWizardFormEntry

FormWizardFormEntryForm.**base_fields = OrderedDict([('form_wizard_entry', <django.forms.models.ModelCh**

FormWizardFormEntryForm.**declared_fields = OrderedDict()**

FormWizardFormEntryForm.**media**

fobi.forms.**FormWizardFormEntryFormSet**
alias of FormWizardFormEntryFormFormSet

class fobi.forms.**FormWizardHandlerEntryForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*)
Bases: django.forms.models.ModelForm

FormWizardHandlerEntry form.

class Meta
Bases: object

Meta class.

fields = ('form_wizard_entry', 'plugin_data', 'plugin_uid')

model
alias of FormWizardHandlerEntry

FormWizardHandlerEntryForm.**base_fields = OrderedDict([('form_wizard_entry', <django.forms.models.Mode**

FormWizardHandlerEntryForm.**declared_fields = OrderedDict([('plugin_uid', <django.forms.fields.ChoiceFie**

FormWizardHandlerEntryForm.**media**

class fobi.forms.**ImportFormEntryForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: django.forms.forms.Form

Import form entry form.

base_fields = OrderedDict([('file', <django.forms.fields.FileField object at 0x7ff21ec23b50>)])

declared_fields = OrderedDict([('file', <django.forms.fields.FileField object at 0x7ff21ec23b50>)])

media

class fobi.forms.**ImportFormWizardEntryForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*)

Bases: *fobi.forms.ImportFormEntryForm*

Import form entry wizard form.

base_fields = OrderedDict([('file', <django.forms.fields.FileField object at 0x7ff21ec23b50>)])

declared_fields = OrderedDict([('file', <django.forms.fields.FileField object at 0x7ff21ec23b50>)])

media

fobi.helpers module

Helpers module. This module can be safely imported from any fobi (sub)module, since it never imports from any of the fobi (sub)modules (except for the *fobi.constants* and *fobi.exceptions* modules).

fobi.helpers.**admin_change_url**(*app_label*, *module_name*, *object_id*, *extra_path=''*, *url_title=None*)

Gets an admin change URL for the object given.

Parameters

- **app_label** (*str*) –
- **module_name** (*str*) –
- **object_id** (*int*) –
- **extra_path** (*str*) –
- **url_title** (*str*) – If given, an HTML a tag is returned with *url_title* as the tag title. If left to None just the URL string is returned.

Return str

fobi.helpers.**clean_dict**(*source*, *keys=[]*, *values=[]*)
Removes given keys and values from dictionary.

Parameters

- **source** (*dict*) –
- **keys** (*iterable*) –
- **values** (*iterable*) –

Return dict

fobi.helpers.**clone_file**(*upload_dir*, *source_filename*, *relative_path=True*)
Clones the file.

Parameters **source_filename** (*string*) – Source filename.

Return string Filename of the cloned file.

fobi.helpers.**combine_dicts**(*headers*, *data*)
Combine dicts.

Takes two dictionaries, assuming one contains a mapping keys to titles and another keys to data. Joins as string and returns a result dict.

fobi.helpers.**delete_file**(*image_file*)
Delete file from disc.

fobi.helpers.**do_slugify**(*val*)
Slugify.

fobi.helpers.**ensure_unique_filename**(*destination*)
Makes sure filenames are never overwritten.

Parameters **destination** (*string*) –

Return string

fobi.helpers.**flatatt_inverse_quotes**(*attrs*)
Convert a dictionary of attributes to a single string.

The returned string will contain a leading space followed by key="value", XML-style pairs. In the case of a boolean value, the key will appear without a value. It is assumed that the keys do not need to be XML-escaped. If the passed dictionary is empty, then return an empty string.

The result is passed through 'mark_safe' (by way of 'format_html_join').

fobi.helpers.**get_app_label_and_model_name**(*path*)
Gets app_label and model_name from the path given.

Parameters **path** (*str*) – Dotted path to the model (without ".model", as stored in the Django *ContentType* model.

Return tuple app_label, model_name

fobi.helpers.**get_form_element_entries_for_form_wizard_entry**(*form_wizard_entry*)
Get form element entries for the form wizard entry.

fobi.helpers.**get_model_name_for_object**(*obj*)
Get model name for object.

Django version agnostic.

fobi.helpers.**get_registered_models**(*ignore=[]*)
Gets registered models as list.

Parameters **ignore** (*iterable*) – Ignore the following content types (should be in `app_label.model` format (example `auth.User`).

Return list

`fobi.helpers.`**get_select_field_choices**(*raw_choices_data*, *key_type=None*, *value_type=None*, *fail_silently=True*)

Get select field choices.

Used in `radio`, `select` and other choice based fields.

Parameters

- **raw_choices_data** (*str*) –
- **key_type** (type) –
- **value_type** (type) –
- **fail_silently** (*bool*) –

Return list

`fobi.helpers.`**get_wizard_form_field_value_from_post**(*request*, *wizard_view_name*, *form_key*, *field_name*, *fail_silently=True*)

Get wizard form field value from POST.

This is what we could have:

```
>>> request.POST
>>> {
>>>     'csrfmiddlewaretoken': ['kEprTL218a8HNcC02QefNNnF'],
>>>     'slider-form-test_slider': ['14'],
>>>     'form_wizard_view-current_step': ['slider-form'],
>>>     'slider-form-test_email': ['user@example.com']
>>> }
```

Note, that we know nothing about the types here, type conversion should be done manually. The values returned are strings always.

Parameters

- **request** (*django.http.HttpRequest*) –
- **wizard_view_name** (*str*) –
- **form_key** (*str*) – Typically, this would be the step name (form slug).
- **field_name** (*str*) – Field name.
- **fail_silently** (*bool*) – If set to True, no errors raised.

Return str Since everything in session is stored as string.

`fobi.helpers.`**get_wizard_form_field_value_from_request**(*request*, *wizard_view_name*, *form_key*, *field_name*, *fail_silently=True*, *session_priority=False*)

Get wizard form field value from request.

Note, that we know nothing about the types here, type conversion should be done manually. The values returned are strings always.

Parameters

- **request** (*django.http.HttpRequest*) –

- **wizard_view_name** (*str*) –
- **form_key** (*str*) – Typically, this would be the step name (form slug).
- **field_name** (*str*) – Field name.
- **fail_silently** (*bool*) – If set to True, no errors raised.
- **session_priority** (*bool*) – If set to True, first try to read from session.

Return str Since everything in session is stored as string.

fobi.helpers.**get_wizard_form_field_value_from_session**(*request*, *wizard_view_name*, *form_key*, *field_name*, *fail_silently=True*)

Get wizard form field value from session.

This is what we could have:

```
>>> request.session['wizard_form_wizard_view']['step_data']
>>> {
>>>     'slider-form': {
>>>         'csrfmiddlewaretoken': ['DhINThGTgQ50e2lDnGG4nYrG0a'],
>>>         'slider-form-test_slider': ['14'],
>>>         'form_wizard_view-current_step': ['slider-form'],
>>>         'slider-form-test_email': ['user@example.com']
>>>     }
>>> }
```

Note, that we know nothing about the types here, type conversion should be done manually. The values returned are strings always.

Parameters

- **request** (*django.http.HttpRequest*) –
- **wizard_view_name** (*str*) –
- **form_key** (*str*) – Typically, this would be the step name (form slug).
- **field_name** (*str*) – Field name.
- **fail_silently** (*bool*) – If set to True, no errors raised.

Return str Since everything in session is stored as string.

fobi.helpers.**handle_uploaded_file**(*upload_dir*, *image_file*)

Handle uploaded files.

Parameters **image_file** (*django.core.files.uploadedfile.InMemoryUploadedFile*) –

Return string Path to the image (relative).

fobi.helpers.**iterable_to_dict**(*items*, *key_attr_name*)

Converts iterable of certain objects to dict.

Parameters

- **items** (*iterable*) –
- **key_attr_name** (*string*) – Attribute to use as a dictionary key.

Return dict

class fobi.helpers.**JSONDataExporter**(*data*, *filename*)

Bases: object

Exporting the data into JSON.

export()
Export.

export_to_json()
Export data to JSON.

fobi.helpers.**lists_overlap**(*sub*, *main*)
Check whether lists overlap.

fobi.helpers.**map_field_name_to_label**(*form*)
Takes a form and creates label to field name map.

Parameters **form** (*django.forms.Form*) – Instance of `django.forms.Form`.

Return dict

fobi.helpers.**safe_text**(*text*)
Safe text (encode).

Return str

class fobi.helpers.**StrippedRequest**(*request*)
Bases: `object`

Stripped request object.

META
Request meta stripped down.

A standard Python dictionary containing all available HTTP headers. Available headers depend on the client and server, but here are some examples:

•HTTP_ACCEPT_ENCODING: Acceptable encodings for the response.

•HTTP_ACCEPT_LANGUAGE: Acceptable languages for the response.

•HTTP_HOST: The HTTP Host header sent by the client.

•HTTP_REFERER: The referring page, if any.

•HTTP_USER_AGENT: The clients user-agent string.

•QUERY_STRING: The query string, as a single (unparsed) string.

•REMOTE_ADDR: The IP address of the client.

get_full_path()
Returns the path, plus an appended query string, if applicable.

is_ajax()
Is ajax?

Returns True if the request was made via an XMLHttpRequest, by checking the HTTP_X_REQUESTED_WITH header for the string 'XMLHttpRequest'.

is_secure()
Is secure.

Returns True if the request is secure; that is, if it was made with HTTPS.

path
Path.

A string representing the full path to the requested page, not including the scheme or domain.

class fobi.helpers.**StrippedUser**(*user*)
Bases: object

Stripped user object.

email
Email.

get_full_name()
Get full name.

get_short_name()
Get short name.

get_username()
Get username.

is_anonymous()
Is anonymous.

fobi.helpers.**two_dicts_to_string**(*headers*, *data*, *html_element='p'*)
Two dicts to string.

Takes two dictionaries, assuming one contains a mapping keys to titles and another keys to data. Joins as string and returns wrapped into HTML "p" tag.

fobi.helpers.**uniquify_sequence**(*sequence*)
Uniqify sequence.

Makes sure items in the given sequence are unique, having the original order preserved.

Parameters **sequence** (*iterable*) –

Return list

fobi.helpers.**update_plugin_data**(*entry*, *request=None*)
Update plugin data.

Update plugin data of a given entry.

fobi.helpers.**validate_initial_for_choices**(*plugin_form*, *field_name_choices='choices'*, *field_name_initial='initial'*)
Validate init for choices. Validates the initial value for the choices given.

Parameters

- **plugin_form** (fobi.base.BaseFormFieldPluginForm) –
- **field_name_choices** (*str*) –
- **field_name_initial** (*str*) –

Return str

fobi.helpers.**validate_initial_for_multiple_choices**(*plugin_form*, *field_name_choices='choices'*, *field_name_initial='initial'*)
Validates the initial value for the multiple choices given.

Parameters

- **plugin_form** (fobi.base.BaseFormFieldPluginForm) –
- **field_name_choices** (*str*) –
- **field_name_initial** (*str*) –

Return str

fobi.helpers.**validate_submit_value_as**(*value*)
Validates the *SUBMIT_AS_VALUE*.

Parameters **value** (*str*) –

fobi.models module

class fobi.models.**AbstractPluginModel**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Abstract plugin model.

Used when fobi.settings.RESTRICT_PLUGIN_ACCESS is set to True.

Properties

- *plugin_uid* (str): Plugin UID.
- *users* (django.contrib.auth.models.User): White list of the users allowed to use the plugin.
- *groups* (django.contrib.auth.models.Group): White list of the user groups allowed to use the plugin.

class Meta
Bases: object

Meta class.

abstract = False

AbstractPluginModel.**get_registered_plugins**()
Get registered plugins.

AbstractPluginModel.**groups**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

AbstractPluginModel.**groups_list**()
Groups list.

Flat list (comma separated string) of groups allowed to use the plugin. Used in Django admin.

Return string

AbstractPluginModel.**plugin_uid_admin**()
Plugin uid admin.

Mainly used in admin.

AbstractPluginModel.**plugin_uid_code**()
Plugin uid code.

Mainly used in admin.

AbstractPluginModel.**users**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

AbstractPluginModel.**users_list**()
Users list.

Flat list (comma separated string) of users allowed to use the plugin. Used in Django admin.

Return string

class fobi.models.**FormElement**(**args*, ***kwargs*)
Bases: *fobi.models.AbstractPluginModel*

Form element.

Form field plugin. Used when fobi.settings.RESTRICT_PLUGIN_ACCESS is set to True.

Properties

- *plugin_uid* (str): Plugin UID.
- *users* (django.contrib.auth.models.User): White list of the users allowed to use the form element plugin.
- *groups* (django.contrib.auth.models.Group): White list of the user groups allowed to use the form element plugin.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception FormElement.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

FormElement.**get_registered_plugins**()
Add choices.

FormElement.**groups**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

FormElement.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormElement.**objects = <django.db.models.manager.Manager object>**

FormElement.**plugin_uid**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormElement.**users**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class fobi.models.**FormHandler**(**args*, ***kwargs*)
Bases: *fobi.models.AbstractPluginModel*

Form handler plugin. Used when `fobi.settings.RESTRICT_PLUGIN_ACCESS` is set to True.

Properties

- *plugin_uid* (str): Plugin UID.
- *users* (django.contrib.auth.models.User): White list of the users allowed to use the form handler plugin.
- *groups* (django.contrib.auth.models.Group): White list of the user groups allowed to use the form handler plugin.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception FormHandler.**MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

FormHandler.**get_registered_plugins**()
Add choices.

FormHandler.**groups**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

FormHandler.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormHandler.**objects = <django.db.models.manager.Manager object>**

FormHandler.**plugin_uid**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormHandler.**users**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

class fobi.models.**FormWizardHandler**(**args*, ***kwargs*)
Bases: *fobi.models.AbstractPluginModel*

Form wizard handler plugin. Used when fobi.settings.RESTRICT_PLUGIN_ACCESS is set to True.

Properties

- *plugin_uid* (str): Plugin UID.
- *users* (django.contrib.auth.models.User): White list of the users allowed to use the form handler plugin.
- *groups* (django.contrib.auth.models.Group): White list of the user groups allowed to use the form handler plugin.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception FormWizardHandler.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

FormWizardHandler.**get_registered_plugins**()
Add choices.

FormWizardHandler.**groups**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

pizza.toppings and topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

FormWizardHandler.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormWizardHandler.**objects = <django.db.models.manager.Manager object>**

FormWizardHandler.**plugin_uid**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormWizardHandler.**users**
Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class `fobi.models.`**`BaseAbstractPluginEntry`**(**args*, ***kwargs*)
Bases: `django.db.models.base.Model`

Base for AbstractPluginEntry.

Properties

- *plugin_data* (str): JSON formatted string with plugin data.

class `Meta`
Bases: `object`

Meta class.

`abstract` = False

`BaseAbstractPluginEntry.`**`entry_user`**
Get user from the parent container.

`BaseAbstractPluginEntry.`**`get_plugin`**(*fetch_related_data=False*, *request=None*)
Get plugin.

Gets the plugin class (by `plugin_uid` property), makes an instance of it, serves the data stored in `plugin_data` field (if available). Once all is done, plugin is ready to be rendered.

Parameters **`fetch_related_data`** (*bool*) – When set to True, plugin is told to re-fetch all related data (stored in models or other sources).

Return fobi.base.BasePlugin Subclass of `fobi.base.BasePlugin`.

`BaseAbstractPluginEntry.`**`get_registered_plugins`**()
Get registered plugins.

`BaseAbstractPluginEntry.`**`get_registry`**()
Get registry.

`BaseAbstractPluginEntry.`**`plugin_data`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`BaseAbstractPluginEntry.`**`plugin_name`**()
Plugin name.

`BaseAbstractPluginEntry.`**`plugin_uid_code`**()
Plugin uid code.

Mainly used in admin.

class `fobi.models.`**`AbstractPluginEntry`**(**args*, ***kwargs*)
Bases: *`fobi.models.BaseAbstractPluginEntry`*

Abstract plugin entry.

Properties

•*form_entry* (fobi.models.FormEntry): Form to which the field plugin belongs to.

•*plugin_uid* (str): Plugin UID.

•*plugin_data* (str): JSON formatted string with plugin data.

class Meta
Bases: object

Meta class.

abstract = False

AbstractPluginEntry.**entry_user**
Get user.

AbstractPluginEntry.**form_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

AbstractPluginEntry.**form_entry_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class fobi.models.**FormWizardEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Form wizard entry.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception FormWizardEntry.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

FormWizardEntry.**created**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormWizardEntry.**formwizardformentry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

FormWizardEntry.**formwizardhandlerentry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`FormWizardEntry.`**`get_absolute_url`**`()`
Get absolute URL.

Absolute URL, which goes to the form-wizard view view.

Return string

`FormWizardEntry.`**`get_wizard_type_display`**`(*moreargs, **morekwargs)`

`FormWizardEntry.`**`id`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardEntry.`**`is_cloneable`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardEntry.`**`is_public`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardEntry.`**`name`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardEntry.`**`objects = <django.db.models.manager.Manager object>`**

`FormWizardEntry.`**`savedformwizarddataentry_set`**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`FormWizardEntry.`**`show_all_navigation_buttons`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardEntry.`**`slug`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardEntry.`**`success_page_message`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardEntry.`**`success_page_title`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardEntry.`**`title`**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormWizardEntry.**updated**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormWizardEntry.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

FormWizardEntry.**user_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormWizardEntry.**wizard_type**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class fobi.models.**FormEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Form entry.

Properties

- *user* (django.contrib.auth.models.User: User owning the plugin.
- *name* (str): Form name.
- *title* (str): Form title - used in templates.
- *slug* (str): Form slug.
- *description* (str): Form description.
- *is_public* (bool): If set to True, is visible to public.
- *is_cloneable* (bool): If set to True, is cloneable.
- *position* (int): Ordering position in the wizard.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception FormEntry.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

FormEntry.**action**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**created**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**formelemententry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`FormEntry.`**`formfieldsetentry_set`**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`FormEntry.`**`formhandlerentry_set`**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`FormEntry.`**`formwizardformentry_set`**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`FormEntry.`**`get_absolute_url`**`()`

Get absolute URL.

Absolute URL, which goes to the form-entry view view page.

Return string

`FormEntry.`**`id`**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormEntry.`**`is_cloneable`**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**is_public**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**name**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**objects = <django.db.models.manager.Manager object>**

FormEntry.**savedformdataentry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

FormEntry.**slug**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**success_page_message**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**success_page_title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**title**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**updated**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormEntry.**user**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

FormEntry.**user_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class fobi.models.**FormElementEntry**(**args*, ***kwargs*)
Bases: *fobi.models.AbstractPluginEntry*

Form field entry.

Properties

•*form* (fobi.models.FormEntry): Form to which the field plugin belongs to.

•*plugin_uid* (str): Plugin UID.

•*plugin_data* (str): JSON formatted string with plugin data.

•*form_fieldset_entry*: Fieldset.

•*position* (int): Entry position.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `FormElementEntry.`**MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

`FormElementEntry.`**form_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`FormElementEntry.`**form_fieldset_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`FormElementEntry.`**form_fieldset_entry_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormElementEntry.`**get_registered_plugins**`()`
Gets registered plugins.

`FormElementEntry.`**get_registry**`()`
Get registry.

`FormElementEntry.`**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormElementEntry.`**objects = <django.db.models.manager.Manager object>**

`FormElementEntry.`**plugin_uid**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormElementEntry.`**position**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class `fobi.models.`**FormFieldsetEntry**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

Form fieldset entry.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception FormFieldsetEntry.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

FormFieldsetEntry.**form_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

FormFieldsetEntry.**form_entry_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormFieldsetEntry.**formelemententry_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

FormFieldsetEntry.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormFieldsetEntry.**is_repeatable**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormFieldsetEntry.**name**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormFieldsetEntry.**objects = <django.db.models.manager.Manager object>**

class fobi.models.**FormHandlerEntry**(**args*, ***kwargs*)
Bases: *fobi.models.AbstractPluginEntry*

Form handler entry.

Properties

- *form_entry* (fobi.models.FormEntry): Form to which the handler plugin belongs to.
- *plugin_uid* (str): Plugin UID.
- *plugin_data* (str): JSON formatted string with plugin data.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception FormHandlerEntry.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

FormHandlerEntry.**form_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

FormHandlerEntry.**get_registered_plugins**()
Gets registered plugins.

FormHandlerEntry.**get_registry**()
Get registry.

FormHandlerEntry.**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

FormHandlerEntry.**objects = <django.db.models.manager.Manager object>**

FormHandlerEntry.**plugin_uid**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class fobi.models.**AbstractFormWizardPluginEntry**(**args*, ***kwargs*)
Bases: *fobi.models.BaseAbstractPluginEntry*

Abstract form wizard plugin entry.

Properties

- *form_entry* (fobi.models.FormWizardEntry): FormWizard to which the plugin belongs to.
- *plugin_uid* (str): Plugin UID.
- *plugin_data* (str): JSON formatted string with plugin data.

class Meta
Bases: object

Meta class.

abstract = False

AbstractFormWizardPluginEntry.**entry_user**
Get user.

AbstractFormWizardPluginEntry.**form_wizard_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

AbstractFormWizardPluginEntry.**form_wizard_entry_id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class fobi.models.**FormWizardHandlerEntry**(**args*, ***kwargs*)
Bases: *fobi.models.AbstractFormWizardPluginEntry*

Form wizard handler entry.

Properties

- *form_wizard_entry* (fobi.models.FormWizardEntry): FormWizard to which the handler plugin belongs to.
- *plugin_uid* (str): Plugin UID.
- *plugin_data* (str): JSON formatted string with plugin data.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception FormWizardHandlerEntry.MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

`FormWizardHandlerEntry.`**form_wizard_entry**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`FormWizardHandlerEntry.`**get_registered_plugins**`()`
Gets registered plugins.

`FormWizardHandlerEntry.`**get_registry**`()`
Get registry.

`FormWizardHandlerEntry.`**id**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

`FormWizardHandlerEntry.`**objects = <django.db.models.manager.Manager object>**

`FormWizardHandlerEntry.`**plugin_uid**
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

fobi.settings module

- *RESTRICT_PLUGIN_ACCESS* (bool): If set to True, (Django) permission system for fobi plugins is enabled.
- *FORM_ELEMENT_PLUGINS_MODULE_NAME* (str): Name of the module to placed in the (external) apps in which the fobi form element plugin code should be implemented and registered.
- *FORM_HANDLER_PLUGINS_MODULE_NAME* (str): Name of the module to placed in the (external) apps in which the fobi form handler plugin code should be implemented and registered.
- *FORM_CALLBACKS_MODULE_NAME* (str): Name of the module to placed in the (external) apps in which the fobi form callback code should be implemented and registered.
- *FORM_HANDLER_PLUGINS_EXECUTION_ORDER* (tuple): Order in which the form handler plugins are to be executed.
- *FORM_WIZARD_HANDLER_PLUGINS_EXECUTION_ORDER* (tuple): Order in which the form handler plugins are to be executed.
- *DEBUG*

fobi.test module

fobi.utils module

Another helper module. This module can NOT be safely imported from any fobi (sub)module - thus should be imported carefully.

fobi.utils.**get_allowed_plugin_uids**(*plugin_model_cls*, *user*)
Get allowed plugins uids for user given.

Parameters

- **plugin_model_cls** (fobi.models.AbstractPluginModel) – Subclass of `fobi.models.AbstractPluginModel`.
- **user** (*django.contrib.auth.models.User*) –

Return list

fobi.utils.**get_user_plugins**(*get_allowed_plugin_uids_func*, *get_registered_plugins_func*, *registry*, *user*)
Get user plugins.

Gets a list of user plugins in a form if tuple (plugin name, plugin description). If not yet autodiscovered, autodiscovers them.

Parameters

- **get_allowed_plugin_uids_func** (*callable*) –
- **get_registered_plugins_func** (*callable*) –
- **registry** (fobi.base.BaseRegistry) – Subclass of `fobi.base.BaseRegistry` instance.
- **user** (*django.contrib.auth.models.User*) –

Return list

fobi.utils.**get_user_plugin_uids**(*get_allowed_plugin_uids_func*, *get_registered_plugin_uids_func*, *registry*, *user*)
Gets a list of user plugin uids as a list.

If not yet auto-discovered, auto-discovers them.

Parameters

- **get_allowed_plugin_uids_func** (*callable*) –
- **get_registered_plugin_uids_func** (*callable*) –
- **registry** (fobi.base.BaseRegistry) – Subclass of `fobi.base.BaseRegistry` instance.
- **user** (*django.contrib.auth.models.User*) –

Return list

fobi.utils.**sync_plugins**()
Sync registered plugins.

Syncs the registered plugin list with data in `fobi.models.FormFieldPluginModel`, `fobi.models.FormHandlerPluginModel` and `fobi.models.FormWizardHandlerPluginModel`.

fobi.utils.**get_allowed_form_element_plugin_uids**(*user*)
Get allowed form element plugin uids.

fobi.utils.**get_user_form_element_plugins**(*user*)
Get user form element plugins.

fobi.utils.**get_allowed_form_handler_plugin_uids**(*user*)
Get allowed form handler plugin uids.

fobi.utils.**get_allowed_form_wizard_handler_plugin_uids**(*user*)
Get allowed form wizard handler plugin uids.

fobi.utils.**get_user_form_handler_plugins**(*user*, *exclude_used_singles=False*, *used_form_handler_plugin_uids=[]*)
Get list of plugins allowed for user.

Parameters

- **user** (*django.contrib.auth.models.User*) –
- **exclude_used_singles** (*bool*) –
- **used_form_handler_plugin_uids** (*list*) –

Return list

fobi.utils.**get_user_form_wizard_handler_plugins**(*user*, *exclude_used_singles=False*, *used_form_wizard_handler_plugin_uids=[]*)
Get list of plugins allowed for user.

Parameters

- **user** (*django.contrib.auth.models.User*) –
- **exclude_used_singles** (*bool*) –
- **used_form_wizard_handler_plugin_uids** (*list*) –

Return list

fobi.utils.**get_user_form_handler_plugin_uids**(*user*)
Get user form handler plugin uids.

fobi.utils.**get_user_form_wizard_handler_plugin_uids**(*user*)
Get user form handler plugin uids.

fobi.utils.**get_user_plugins_grouped**(*get_allowed_plugin_uids_func*, *get_registered_plugins_grouped_func*, *registry*, *user*, *sort_items=True*)
Get user plugins grouped.

Parameters

- **get_allowed_plugin_uids_func** (*callable*) –
- **get_registered_plugins_grouped_func** (*callable*) –
- **registry** (fobi.base.BaseRegistry) – Subclass of `fobi.base.BaseRegistry` instance.
- **user** (*django.contrib.auth.models.User*) –
- **sort_items** (*bool*) –

Return dict

fobi.utils.**get_user_form_element_plugins_grouped**(*user*)
Get user form element plugins grouped.

fobi.utils.**get_user_form_handler_plugins_grouped**(*user*)
Get user form handler plugins grouped.

fobi.utils.**get_user_form_wizard_handler_plugins_grouped**(*user*)
Get user form wizard handler plugins grouped.

fobi.utils.**prepare_form_entry_export_data**(*form_entry*, *form_element_entries=None*, *form_handler_entries=None*)
Prepare form entry export data.

Parameters

- **form_entry** (*fobi.modes.FormEntry*) – Instance of.
- **form_element_entries** (*django.db.models.QuerySet*) – QuerySet of FormElementEntry instances.
- **form_handler_entries** (*django.db.models.QuerySet*) – QuerySet of FormHandlerEntry instances.

Return str

fobi.utils.**perform_form_entry_import**(*request*, *form_data*)
Perform form entry import.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_data** (*dict*) –

:return :class:‘fobi.modes.FormEntry: Instance of.

fobi.validators module

fobi.validators.**url_exists**(*url*, *local=False*)
Check if URL exists.

Parameters

- **url** (*str*) –
- **local** (*bool*) –

Return bool

fobi.views module

Views.

fobi.views.**add_form_element_entry**(*request*, **args*, ***kwargs*)
Add form element entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_id** (*int*) –
- **form_element_plugin_uid** (*int*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**add_form_handler_entry**(*request*, **args*, ***kwargs*)
Add form handler entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_id** (*int*) –
- **form_handler_plugin_uid** (*int*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**add_form_wizard_form_entry**(*request*, **args*, ***kwargs*)
Add form wizard form entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_wizard_entry_id** (*int*) –
- **form_entry_id** (*int*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**add_form_wizard_handler_entry**(*request*, **args*, ***kwargs*)
Add form handler entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_id** (*int*) –
- **form_handler_plugin_uid** (*int*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**create_form_entry**(*request*, **args*, ***kwargs*)
Create form entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*str*) –

Return django.http.HttpResponse

fobi.views.**create_form_wizard_entry**(*request*, **args*, ***kwargs*)
Create form wizard entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*str*) –

Return django.http.HttpResponse

fobi.views.**dashboard**(*request*, **args*, ***kwargs*)
Dashboard.

Parameters

- **request** (*django.http.HttpRequest*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**delete_form_element_entry**(*request*, **args*, ***kwargs*)
Delete form element entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_element_entry_id** (*int*) –

Return django.http.HttpResponse

fobi.views.**delete_form_entry**(*request*, **args*, ***kwargs*)
Delete form entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_id** (*int*) –
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**delete_form_handler_entry**(*request*, **args*, ***kwargs*)
Delete form handler entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_handler_entry_id** (*int*) –

Return django.http.HttpResponse

fobi.views.**delete_form_wizard_entry**(*request*, **args*, ***kwargs*)
Delete form wizard entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_wizard_entry_id** (*int*) –
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**delete_form_wizard_form_entry**(*request*, **args*, ***kwargs*)
Delete form wizard form entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_wizard_form_entry_id** (*int*) –

Return django.http.HttpResponse

fobi.views.**edit_form_element_entry**(*request*, **args*, ***kwargs*)
Edit form element entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_element_entry_id** (*int*) –
- **fobi.base.BaseTheme** – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**edit_form_entry**(*request*, **args*, ***kwargs*)
Edit form entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_id** (*int*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*str*) –

Return django.http.HttpResponse

fobi.views.**edit_form_handler_entry**(*request*, **args*, ***kwargs*)
Edit form handler entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_handler_entry_id** (*int*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**edit_form_wizard_handler_entry**(*request*, **args*, ***kwargs*)
Edit form handler entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_wizard_handler_entry_id** (*int*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**export_form_entry**(*request*, **args*, ***kwargs*)
Export form entry to JSON.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_id** (*int*) –
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**form_entry_submitted**(*request*, *form_entry_slug=None*, *template_name=None*)
Form entry submitted.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_slug** (*string*) –
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**form_importer**(*request*, **args*, ***kwargs*)
Form importer.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_importer_plugin_uid** (*str*) –
- **template_name** (*str*) –

fobi.views.**form_wizard_entry_submitted**(*request*, *form_wizard_entry_slug=None*, *template_name=None*)
Form wizard entry submitted.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_wizard_entry_slug** (*string*) –
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**form_wizards_dashboard**(*request*, **args*, ***kwargs*)
Dashboard for form wizards.

Parameters

- **request** (*django.http.HttpRequest*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

class fobi.views.**FormWizardView**(***kwargs*)
Bases: *fobi.wizard.views.dynamic.DynamicSessionWizardView*

Dynamic form wizard.

done (*form_list*, ***kwargs*)
Done.

file_storage = <django.core.files.storage.FileSystemStorage object>

get_context_data (*form*, ***kwargs*)
Get context data.

get_form_entry_for_step (*step*)
Get form entry title for step.

get_ignorable_field_names (*form_element_entries*)
Get ignorable field names.

get_initial_wizard_data (*request*, **args*, ***kwargs*)
Get initial wizard data.

post (**args*, ***kwargs*)
POST requests.

This method handles POST requests.

The wizard will render either the current step (if form validation wasn't successful), the next step (if the current step was stored successful) or the done view (if no more steps are available)

render_done (*form*, ***kwargs*)
Render done.

This method gets called when all forms passed. The method should also re-validate all steps to prevent manipulation. If any form fails to validate, *render_revalidation_failure* should get called. If everything is fine call *done*.

fobi.views.**import_form_entry** (*request*, **args*, ***kwargs*)
Import form entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**import_form_wizard_entry** (*request*, **args*, ***kwargs*)
Import form wizard entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.views.**view_form_entry** (*request*, *form_entry_slug*, *theme=None*, *template_name=None*)
View created form.

Parameters

- **request** (*django.http.HttpRequest*) –
- **form_entry_slug** (*string*) –
- **theme** (*fobi.base.BaseTheme*) – Theme instance.
- **template_name** (*string*) –

Return django.http.HttpResponse

fobi.widgets module

class fobi.widgets.**NumberInput**(*attrs=None*)
Bases: django.forms.widgets.TextInput

input_type = u'number'

media

class fobi.widgets.**BooleanRadioSelect**(**args*, ***kwargs*)
Bases: django.forms.widgets.RadioSelect

Boolean radio select for Django.

Example

```
>>> class DummyForm(forms.Form):
>>>     agree = forms.BooleanField(label=_("Agree?"),
>>>                                required=False,
>>>                                widget=BooleanRadioSelect)
```

media

class fobi.widgets.**RichSelect**(*attrs=None*, *choices=()*, *prepend_html=None*, *append_html=None*, *override_name=None*)
Bases: django.forms.widgets.Select

Rich select widget with some rich enhancements.

Based on original Select widget and intended to be a drop-off replacement.

media

render(*name*, *value*, *attrs=None*)
Renders the element, having prepended and appended extra parts.

class fobi.widgets.**RichSelectInverseQuotes**(*attrs=None*, *choices=()*, *prepend_html=None*, *append_html=None*, *override_name=None*)
Bases: *fobi.widgets.RichSelect*

Almost same as original, but uses alternative flatatt function.

Uses inverse quotes.

media

render(*name*, *value*, *attrs=None*)

Module contents

Quick start

Tutorial for very quick start with django-fobi. Consists of several parts listed below:

```
- Part 1: Standard Django installation
- Part 2: Integration with DjangoCMS (coming soon)
```

Part 1: standard Django installation

Example project code available here.

Installation and configuration

Install the package in your environment.

```
pip install django-fobi
```

INSTALLED_APPS

Add `fobi` core and the plugins to the `INSTALLED_APPS` of the your *settings* module.

1. The core.

```
'fobi',
```

2. The preferred theme. Bootstrap 3 theme is the default. If you have chosen a different theme, update the value of `FOBI_DEFAULT_THEME` accordingly.

```
'fobi.contrib.themes.bootstrap3',
```

3. The form field plugins. Plugins are like blocks. You are recommended to have them all installed. Note, that the following plugins do not have additional dependencies, while some others (like fobi.contrib.plugins.form_elements.security.captcha or fobi.contrib.plugins.form_elements.security.recaptcha would require additional packages to be installed. If so, make sure to have installed and configured those dependencies prior adding the dependant add-ons to the *settings* module.

```
'fobi.contrib.plugins.form_elements.fields.boolean',
'fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple',
'fobi.contrib.plugins.form_elements.fields.date',
'fobi.contrib.plugins.form_elements.fields.date_drop_down',
'fobi.contrib.plugins.form_elements.fields.datetime',
'fobi.contrib.plugins.form_elements.fields.decimal',
'fobi.contrib.plugins.form_elements.fields.email',
'fobi.contrib.plugins.form_elements.fields.file',
'fobi.contrib.plugins.form_elements.fields.float',
'fobi.contrib.plugins.form_elements.fields.hidden',
'fobi.contrib.plugins.form_elements.fields.input',
'fobi.contrib.plugins.form_elements.fields.integer',
'fobi.contrib.plugins.form_elements.fields.ip_address',
'fobi.contrib.plugins.form_elements.fields.null_boolean',
'fobi.contrib.plugins.form_elements.fields.password',
'fobi.contrib.plugins.form_elements.fields.radio',
'fobi.contrib.plugins.form_elements.fields.regex',
'fobi.contrib.plugins.form_elements.fields.select',
'fobi.contrib.plugins.form_elements.fields.select_model_object',
'fobi.contrib.plugins.form_elements.fields.select_multiple',
'fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects',
'fobi.contrib.plugins.form_elements.fields.slug',
'fobi.contrib.plugins.form_elements.fields.text',
'fobi.contrib.plugins.form_elements.fields.textarea',
'fobi.contrib.plugins.form_elements.fields.time',
'fobi.contrib.plugins.form_elements.fields.url',
```

4. The presentational form elements (images, texts, videos).

```
'easy_thumbnails', # Required by `content_image` plugin
'fobi.contrib.plugins.form_elements.content.content_image',
'fobi.contrib.plugins.form_elements.content.content_text',
'fobi.contrib.plugins.form_elements.content.content_video',
```

5. Form handlers. Note, that some of them may require database sync/migration.

```
'fobi.contrib.plugins.form_handlers.db_store',
'fobi.contrib.plugins.form_handlers.http_repost',
'fobi.contrib.plugins.form_handlers.mail',
```

Putting all together, you would have something like this.

```
INSTALLED_APPS = (
    # Used by fobi
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',

    # ...
    # Core
    'fobi',

    # Theme
    'fobi.contrib.themes.bootstrap3',

    # Form field plugins
    'fobi.contrib.plugins.form_elements.fields.boolean',
    'fobi.contrib.plugins.form_elements.fields.checkbox_select_multiple',
    'fobi.contrib.plugins.form_elements.fields.date',
    'fobi.contrib.plugins.form_elements.fields.date_drop_down',
    'fobi.contrib.plugins.form_elements.fields.datetime',
    'fobi.contrib.plugins.form_elements.fields.decimal',
    'fobi.contrib.plugins.form_elements.fields.email',
    'fobi.contrib.plugins.form_elements.fields.file',
    'fobi.contrib.plugins.form_elements.fields.float',
    'fobi.contrib.plugins.form_elements.fields.hidden',
    'fobi.contrib.plugins.form_elements.fields.input',
    'fobi.contrib.plugins.form_elements.fields.integer',
    'fobi.contrib.plugins.form_elements.fields.ip_address',
    'fobi.contrib.plugins.form_elements.fields.null_boolean',
    'fobi.contrib.plugins.form_elements.fields.password',
    'fobi.contrib.plugins.form_elements.fields.radio',
    'fobi.contrib.plugins.form_elements.fields.regex',
    'fobi.contrib.plugins.form_elements.fields.select',
    'fobi.contrib.plugins.form_elements.fields.select_model_object',
    'fobi.contrib.plugins.form_elements.fields.select_multiple',
    'fobi.contrib.plugins.form_elements.fields.select_multiple_model_objects',
    'fobi.contrib.plugins.form_elements.fields.slug',
    'fobi.contrib.plugins.form_elements.fields.text',
    'fobi.contrib.plugins.form_elements.fields.textarea',
    'fobi.contrib.plugins.form_elements.fields.time',
    'fobi.contrib.plugins.form_elements.fields.url',
```

```
    # Form element plugins
    'easy_thumbnails', # Required by `content_image` plugin
    'fobi.contrib.plugins.form_elements.content.content_image',
    'fobi.contrib.plugins.form_elements.content.content_text',
    'fobi.contrib.plugins.form_elements.content.content_video',

    # Form handlers
    'fobi.contrib.plugins.form_handlers.db_store',
    'fobi.contrib.plugins.form_handlers.http_repost',
    'fobi.contrib.plugins.form_handlers.mail',

    # ...
)
```

TEMPLATE_CONTEXT_PROCESSORS

Add `django.core.context_processors.request` and `fobi.context_processors.theme` to `TEMPLATE_CONTEXT_PROCESSORS` of your *settings* module.

```
TEMPLATE_CONTEXT_PROCESSORS = (
    # ...
    "django.core.context_processors.request",
    "fobi.context_processors.theme", # Obligatory
    "fobi.context_processors.dynamic_values", # Optional
    # ...
)
```

urlpatterns

Add the following line to `urlpatterns` of your *urls* module.

```
urlpatterns = [
    # ...

    # DB Store plugin URLs
    url(r'^fobi/plugins/form-handlers/db-store/',
        include('fobi.contrib.plugins.form_handlers.db_store.urls')),

    # View URLs
    url(r'^fobi/', include('fobi.urls.view')),

    # Edit URLs
    url(r'^fobi/', include('fobi.urls.edit')),

    # ...
]
```

Update the database

1. First you should be syncing/migrating the database. Depending on your Django version and migration app, this step may vary. Typically as follows:

```
$ ./manage.py syncdb
$ ./manage.py migrate --fake-initial
```

2. Sync installed `fobi` plugins. Go to terminal and type the following command.

```
$ ./manage.py fobi_sync_plugins
```

Specify the active theme

Specify the default theme in your *settings* module.

```
FOBI_DEFAULT_THEME = 'bootstrap3'
```

Permissions

`fobi` has been built with permissions in mind. Every single form element plugin or handler is permission based. If user hasn't been given permission to work with a form element or a form handler plugin, he won't be. If you want to switch the permission checks off, set the value of `FOBI_RESTRICT_PLUGIN_ACCESS` to False in your *settings* module.

```
FOBI_RESTRICT_PLUGIN_ACCESS = False
```

Otherwise, after having completed all the steps above, do log into the Django administration and assign the permissions (to certain user or a group) for every single form element or form handler plugin. Bulk assignments work as well.

- http://yourdomain.com/admin/fobi/formelement/
- http://yourdomain.com/admin/fobi/formhandler/

Also, make sure to have the Django model permissions set for following models:

- fobi.models.FormEntry
- fobi.models.FormElementEntry
- fobi.models.FormHandlerEntry
- fobi.contrib.plugins.form_handlers.db_store.models.SavedFormDataEntry

Part 2: Integration with DjangoCMS

Coming soon...

CHAPTER 36

Indices and tables

- genindex
- modindex
- search

[1.1] Dashboard

[1.2] Create a form

[1.3] Edit form - form elements tab active, no elements yet

[1.4] Edit form - form elements tab active, add a form element menu

[1.5] Edit form - add a form element (URL plugin)

[1.6] Edit form - form elements tab active, with form elements

[1.7] Edit form - form handlers tab active, no handlers yet

[1.8] Edit form - form handlers tab tactive, add form handler menu

[1.9] Edit form - add a form handler (Mail plugin)

[1.10] Edit form - form handlers tab active, with form handlers

[1.11] Edit form - form properties tab active

[1.12] View form

[1.13] View form - form submitted (thanks page)

[1.14] Edit form - add a form element (Video plugin)

[1.15] Edit form - add a form element (Boolean plugin)

[1.16] Edit form

[1.17] View form

[2.1] Edit form - form elements tab active, with form elements

[2.2] Edit form - form elements tab active, add a form element menu

[2.3] Edit form - add a form element (Hidden plugin)

[2.4] Edit form - form handlers tab active, with form handlers

[2.5] Edit form - form properties tab active

[2.6] View form

f

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W