django-xross Documentation

Release 0.4.0

Igor ‘idle sign‘ Starikov

March 28, 2015

Contents

https://github.com/idlesign/django-xross

CHAPTER 1

Description

Reusable application for Django nicely bridging client and server sides.

Streamline you server and client interaction using some declarative techniques in your HTML, and a couple of xross functions in your views.

CHAPTER 2

Requirements

1. Python 3.3+
2. Django 1.4+
3. jQuery (make it available in templates)

CHAPTER 3

Table of Contents

3.1 Quickstart

xross requires a few steps to serve you well.

> **Warning:** Do not forget to add the **xross** application to INSTALLED_APPS in your settings file (usually 'settings.py').

Somewhere in your *views.py*:

```
from django.shortcuts import render
from xross.toolbox import xross_view, xross_listener  # That's all we need from xross.


def get_quote(request, vysotsky_only=False):
    """This function (operation in terms of xross) will be used by xross to get a random quote using
    Note that this function could be used as an ordinary view also.

    """
    if vysotsky_only:
        quote = ...  # Some random quote by Vladimir Vysotsky.
    else:
        quote = ...  # Some random quote by any author.
    return render(request, 'mytemplates/sub_quote.html', {'quote': quote})


def list_news(request):
    """This function (operation in terms of xross) will be used by xross to load news using AJAX.
    Note that this function could be used as an ordinary view too.

    """
    news = ...  # Here we fetch some news from DB.
    return render(request, 'mytemplates/sub_news.html', {'news': news})


@xross_view(get_quote, list_news)  # Decorate your view - instruct xross to use 'get_quote' and 'list
def index_page(request):
    """This is our view to streamline."""

    xross_listener()  # xross will handle AJAX calls from that moment.

    return render(request, 'mytemplates/index.html')
```

Now to your *mytemplates/index.html*. Here we work with **xross** in quite a declarative way:

```
<!DOCTYPE html>
<html>
<head>
    <!-- xross depends on jQuery. Include it. -->
    <script src="http://yandex.st/jquery/2.1.1/jquery.min.js"></script>

    <!-- Now xross itself. -->
    <script src="{{ STATIC_URL }}js/xross/xross.min.js"></script>
    <script type="text/javascript">
        xross.automate();  // Instruct xross to watch for page elements with `xross` class.
    </script>
</head>
<body>
    <div id="list_news" class="xross">
        <!--
            Contents of this div will be replaced with news from Django's `list_news()`
            automatically on page load.
            That's the default of xross, but it knows some other nice little tricks.
            Watch for one of those below.
         -->
    </div>

    <!--
        Now let's put here a button which adds a random quote (using `get_quote()`)
        into `quotes_here` div below when clicked.

        Notice that we use some `data-x` attributes to program desired xross behaviour (`x` prefix su

            1. data-xvysotsky_only="true" - True will be passed into `vysotsky_only` keyword argument

            2. data-xtarget="quotes_here" - Defines a target html element (here a div with id `quotes

            3. data-xsuccess="append" - Defines an action to be performed by xross upon a target elem
               In this example we `append` a quote to `quotes_here`.

    -->
    <button id="get_quote" data-xvysotsky_only="true" data-xtarget="quotes_here" data-xsuccess="apper

    <div id="quotes_here">
        <!--
            Click the above button and a quote by Vladimir Vysotsky will be placed here.
        -->
    </div>

</body>
</html>
```

Note: Note that every **xross**-related DOM element has an ID attribute.

And two very simple templates:

mytemplates/sub_news.html:

```
{% for item in news %}
    <div>
        <div>{{ item.title }}</div>
        <div>{{ item.text }}</div>
```

```
    </div>
{% endfor %}
```

mytemplates/sub_quote.html:

```
<div>
    <blockquote>{{ quote.text }}</blockquote>
    <div><i>by {{ quote.author }}</i></div>
</div>
```

3.2 Python Part

Here you'll find some information on Python part of **xross**.

Note: Functions described here are located in **xross.toolbox**.

3.2.1 Operations

xross uses `operation` term to describe a function which is used for handling a **xross** request.

Practically any function can be used as an `operation`.

- **View function**:

```
def my_view_and_op(request, some_id, xross=None):
    """This view could be used both as a separate view,
    and xross operation.

    If use as an operation:

        'request': is a request from the main view (those are decorated with @xross_view());

        'some_id': is get from your template (namely, by default from 'data-xsome_id' attribute
        of a page element with 'my_view_and_op' id);

        'xross': is a xross handler object. That can contain some useful stuff (e.g stuff in 'xross.
        dictionary could be passed with 'xross_listen()' -- see below).
        NB: This keyword argument may be omitted from operation signature if not used.

    """
    ...
```

- **Ordinary function**:

```
def my_op_func(some_id):
    """NB: it also could be made to accept 'xross' keyword argument
    to have access to xross handler object."""

    ...
```

- **Method** (that applies also to class-based views):

```
from django.views.generic.base import View


class MyView(View):
```

```
    def my_op_method(self, request):
        """NB: it also could be made to accept 'xross' keyword argument
        to have access to xross handler object."""

        ...
```

3.2.2 xross_view()

Arguments: *op_functions

This decorator should be used to decorate those applications views that require **xross** functionality.

Pass into it the functions (operations) responsible for handling **xross** requests.

```
from xross.toolbox import xross_view


@xross_view(my_op_func, my_view_and_op)
def index_page(request):
    """This is our view."""
    ...
```

3.2.3 xross_listener()

Arguments: **xross_attrs

Has to be put in your views in places when **xross** handling is expected.

Accepts xross handler attributes as keyword arguments. Those attributes will be available in operation functions from xross handler object (see notes on `xross` keyword argument in Operations section above) in `attrs` attribute.

```
from django.shortcuts import render
from xross.toolbox import xross_view, xross_listener


def my_op_func(some_id, xross=None):

    ...

    item = xross.attrs['that_item']  # 'that_item' is passed here from 'xross_listener()' (see below,

    ...

    return render(request, 'mytemplates/some.html')


@xross_view(my_op_func)
def index_page(request):

    my_item = ...  # Imagine we need to get some item data on every request.

    # Instruct xross to handle AJAX calls from that moment.
    # And make 'that_item' available to operation functions.
    xross_listener({'that_item': my_item})

    ...
```

```
    return render(request, 'mytemplates/index.html')
```

3.2.4 Debugging

While DEBUG in your *settings.py* is set to `True` **xross** will supply you with useful debugging information putting error description in every response to bad requests. Use your browser development console to watch it.

3.3 JavaScript Part

Here you'll find some information on **xross** JavaScript part.

> **Warning:** Do not forget to include jQuery and **xross** itself in your templates:
>
> ```
> <script src="http://yandex.st/jquery/2.1.1/jquery.min.js"></script>
> <script src="{{ STATIC_URL }}js/xross/xross.js"></script>
> ```

3.3.1 xross.debug

Setting `debug` attribute to True allows **xross** to put debug information into browser console.

```
xross.debug = true;  // Remember to set this before other xross calls, e.g. automate().
```

3.3.2 xross.data_items_prefix

Allows to adjust a prefix for **data-** attributes of elements.

Attributes with this prefix will be considered by **xross**.

Default: **x**. E.g: use *myx* to pass all *data-myx* prefixed attributes to xross (data-myxsome, data-myxother, etc.).

3.3.3 xross.automate()

Arguments: xross_class, handler_name

Instructs **xross** to attach its handlers to page elements with a certain class (*xross* by default).

```
// You can instruct xross to watch for page elements with `xross` class.
xross.automate();

// Or any other, e.g. `x`. Automate elements with the default `ajax` handler.
xross.automate('x');
```

3.3.4 xross.describe()

Arguments: el, handler_name, params

Under the cover *automate()* uses this method to describe various page elements in terms of **xross**.

```
// Attach the default ('ajax') handler to 'my_element'.
xross.describe('#my_element');
```

3.3.5 xross handlers

xross relies on so-called *handlers* to perform certain actions.

Each handler can accept certain parameters to adjust its behaviour.

The default handler is `ajax`.

3.3.6 AJAX handler

Alias: **ajax**.

AJAX handler is the default one. It simplifies sending AJAX requests to server and handling responses.

Supported parameters:

- **op**: operation identifier for server side. On server it is usually a name of a function to be executed.

  If not set ID attribute value of a current DOM element is used as operation ID.

  Default: **null**. Examples: null, myoperation.
- **method**: allows to set HTTP method for AJAX requests.

  Default: **GET**. Examples: POST, GET.
- **target**: allows to define a target DOM element over which some actions would be performed on success.

  Accepts a string (elements are addressed by their IDs) or an element object

  Default: **this**. Examples: this, mydiv.
- **event**: allows to define a DOM event which triggers AJAX functionality.

  If set to **auto**, xross will try to detect a proper event basing on element type.

  Default: **auto**. Examples: auto, ready, click.
- **success**: allows to set an action to performed on success.

  Accepts a function or a string (a function path, or action alias).

  Function should accept the same arguments as jQuery.ajax().success() plus a target DOM element.

  Default: **fill**. Examples: fill, replace, my_obj.my_method.

  Action aliases:

  - **empty** - empties target element;
  - **remove** - removes target element;
  - **fill** - replaces target element content with data from server;
  - **replace** - replaces the whole target element with data from server;
  - **append** - appends data from server to target element contents;
  - **prepend** - prepends data from server to target element contents.

- **error**: allows to set an action to performed on request error.

  Accepts a function or a string (a function path, or action alias).

  Function should accept the same arguments as jQuery.ajax().error().

  Default: **log**. Examples: log, my_obj.my_method.

  Action aliases:

  - **log** - dumps error description into browser console.
- **complete**: allows to define a function triggered after both operation success and failure.

  Accepts a function or a string (a function path).

  Function should accept the same arguments as jQuery.ajax().complete().

  Default: **null**. Examples: my_func, my_obj.my_method.
- **form**: allows sending form data to server vie AJAX.

  Accepts a string (forms are addressed by their IDs) or a form object

  Default: **null**. Examples: null, myform.

CHAPTER 4

Get involved into django-xross

Submit issues. If you spotted something weird in application behavior or want to propose a feature you can do that at https://github.com/idlesign/django-xross/issues

Write code. If you are eager to participate in application development, fork it at https://github.com/idlesign/django-xross, write your code, whether it should be a bugfix or a feature implementation, and make a pull request right from the forked project page.

Spread the word. If you have some tips and tricks or any other words in mind that you think might be of interest for the others — publish it.