django-dash Documentation

Release 0.2.4

Artur Barseghyan <artur.barseghyan@gmail.com>

November 29, 2013

Contents

django-dash (later on named Dash) is a customisable, modular dashboard application framework for Django.

Dash allows users to create their own custom dashboards. Supports theming (in Dash themes are called layouts) and multiple workspaces. Dash comes with extensive pythonic API which allows developers to create new Dash plugins, as well as to modify bundled ones.

To make a clearer association, think of Android for tablets (shortcuts, widgets and apps) or Windows 8 for tablets or desktops.

Dash inherits all those concepts and makes it possible to implement a dashboard system for Django applications with minimal efforts.

CHAPTER 1

Prerequisites

- Django 1.5.+
- Python 2.6.8+, 2.7.+, 3.3.+

Core and plugins that don't rely on third-party Django apps seem to work well with Django 1.6 (Selenium tests pass), although Django 1.6 is not yet proclaimed to be flawlessly supported by Dash.

CHAPTER 2

Key concepts

- Each layout (theme) consist of placeholders. Each plugin widget has its' own specific HTML/JavaScript/CSS.
- There might be multiple themes implemented and installed, but only one can be active for a certain user. Default layout is chosen system wide, but each user (if has an appropriate permission) can choose his preferred layout.
- Placeholder is a space, in which the plugin widgets are placed.
- Placeholders are rectangles consisting of cells. Each placeholder has its' own custom number of rows and columns.
- Workspace is just another named dashboard. Users switch between workspaces in navigation. Amount of workspaces is unlimited.
- Plugin is a (Django) micro app. Most heavy work should happen in plugin. Plugin may have its' own views, urls, etc. Rendering happens with use of plugin widgets.
- Plugin widgets are mainly responsible for rendering of the plugin data. Each plugin widget has its' own specific HTML/JavaScript/CSS. A single plugin widget is registered for a triple (layout, placeholder, plugin).
- Public dashboard (implemented as a contrib app, which makes it optional) allows users to make their workspaces public. If user chooses to make his dashboard public, default workspace becomes public. As for non-default workspaces, user can still make each of them private or public.

CHAPTER 3

Main features

- Customisable layouts (aka theeming).
- Multiple workspaces.
- Tunable access permissions to plugins.
- Public dashboards (as a contrib app).

CHAPTER 4

Installation

1. Install latest stable version from PyPI:

   $ pip install django-dash

   Or latest stable version from GitHub:

   $ pip install -e git+https://github.com/barseghyanartur/django-dash@stable#egg=django-dash

   Or latest stable version from BitBucket:

   $ pip install -e hg+https://bitbucket.org/barseghyanartur/django-dash@stable#egg=django-dash

2. Add *dash* to `INSTALLED_APPS` of the your projects' Django settings. Furthermore, all layouts and plugins to be used, shall be added to the `INSTALLED_APPS` as well.

```
>>> INSTALLED_APPS = (
>>>     # ...
>>>     'dash',
>>>     'dash.contrib.layouts.android',
>>>     'dash.contrib.layouts.windows8',
>>>     'dash.contrib.plugins.dummy',
>>>     'dash.contrib.plugins.image',
>>>     'dash.contrib.plugins.memo',
>>>     'dash.contrib.plugins.news',
>>>     'dash.contrib.plugins.rss_feed',
>>>     'dash.contrib.plugins.url',
>>>     'dash.contrib.plugins.video',
>>>     'dash.contrib.plugins.weather',
>>>     # ...
>>> )
```

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

CHAPTER 5

Demo

In order to be able to quickly evaluate the *django-dash*, 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_dash_example_app_installer.sh*:

```
$ wget https://raw.github.com/barseghyanartur/django-dash/stable/example/django_dash_example_app_installer.sh
```

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

```
$ chmod +x django_dash_example_app_installer.sh
```

```
$ ./django_dash_example_app_installer.sh
```

Open your browser and test the app.

Dashboard:

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

Django admin interface:

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

If quick installer doesn't work for you, see the manual steps on running the example project (https://github.com/barseghyanartur/django-dash/tree/stable/example).

Take a look at the templates in "example/example/templates" directory for getting a better idea of how to transform your own- or thirdy-part- templates into Dash templates.

Also, the example project (https://github.com/barseghyanartur/django-dash/tree/stable/example/example/foo) has example layouts, plugins and widgets implemented. Take it as a good example of how to add widgets for existing plugins to your own customly made layout.

See the documentation for some screen shots:

- PythonHosted http://pythonhosted.org/django-dash/#screenshots

- ReadTheDocs http://django-dash.readthedocs.org/en/latest/#screenshots

Creating a new layout

Dash comes with several bundled layouts. Do check their source code as example.

Let's say, our imaginary layout has two placeholders. One large placeholder for all kinds of widgets (called *main*) and a tiny one for shortcuts (called *shortcuts*).

Placeholder *main*:

- Single cell size : 150 x 110 pixels
- Dimensions : 6 cols, 5 rows

Placeholder *shortcuts*:

- Single cell size : 60 x 55 pixels
- Dimensions : 1 cols, 10 rows

See the figure below to get an idea of what placeholders are:

- Placeholder *main* consts of cells from 11 to 56.
- Placeholder *shortcuts* consists of cells from 1 to 10.

A single plugin widget may occupy one or more cells. Plugin widgets are rectangles.

To make it clear, see following cases:

- Plugin widget has 2 cols and 1 row. Then, for example, it may occupy cells (11 and 12).
- Plugin widget has 2 cols and 2 rows. Then, for example, it may occupy cells (11, 12, 21 and 22).
- Plugin widget has 1 col and 3 rows. Then, for example, it may occupy cells (11, 21 and 31).
- Plugin widget has 4 cols and 3 rows. Then, for example, it may occupy cells (22, 23, 24, 25, 32, 33, 34, 35, 42, 43, 44 and 45).

```
>>>                                 'main'                                'shortcuts'
>>>------------------------------------------------------------------------ -----
>>>|           |           |           |           |           |           | |  1  |
>>>|           |           |           |           |           |           | |     |
>>>|    11     |    12     |    13     |    14     |    15     |    16     | -----
>>>|           |           |           |           |           |           | |  2  |
>>>|           |           |           |           |           |           | |     |
>>>------------------------------------------------------------------------ -----
>>>|           |           |           |           |           |           | |     |
```

```
>>>|           |           |           |           |           |           | |  3  |
>>>|    21     |    22     |    23     |    24     |    25     |    26     | -----
>>>|           |           |           |           |           |           | |  4  |
>>>|           |           |           |           |           |           | |     |
>>>-------------------------------------------------------------------- -----
>>>|           |           |           |           |           |           | |     |
>>>|           |           |           |           |           |           | |  5  |
>>>|    31     |    32     |    33     |    34     |    35     |    36     | -----
>>>|           |           |           |           |           |           | |  6  |
>>>|           |           |           |           |           |           | |     |
>>>-------------------------------------------------------------------- -----
>>>|           |           |           |           |           |           | |     |
>>>|           |           |           |           |           |           | |  7  |
>>>|    41     |    42     |    43     |    44     |    45     |    46     | -----
>>>|           |           |           |           |           |           | |  8  |
>>>|           |           |           |           |           |           | |     |
>>>-------------------------------------------------------------------- -----
>>>|           |           |           |           |           |           | |     |
>>>|           |           |           |           |           |           | |  9  |
>>>|    51     |    52     |    53     |    54     |    55     |    56     | -----
>>>|           |           |           |           |           |           | | 10  |
>>>|           |           |           |           |           |           | |     |
>>>-------------------------------------------------------------------- -----
```

There are some rules/guideles you should follow.

Let's assume that layout is named *example*. The layout directory should then have the following structure.

```
>>> path/to/layout/example/
>>> -- static
>>> |   -- css
>>> |   |   -- dash_layout_example.css # Contains layout-specific CSS
>>> |   -- images
>>> |   -- js
>>> |       -- dash_layout_example.js # Contains layout specific JavaScripts
>>> -- templates
>>> |   -- example
>>> |       -- edit_layout.html # Master edit layout
>>> |       -- view_layout.html # Master view layout
>>> -- __init__.py
>>> -- dash_layouts.py # Where Layouts and Placeholders are defined and registered
```

Layout and placeholder classes should be placed in the *dash_layouts.py* file.

Each layout should be put into the `INSTALLED_APPS` of your Django projects' settings module.

```
>>> INSTALLED_APPS = (
>>>     # ...
>>>     'path.to.layout.example',
>>>     # ...
>>> )
```

6.1 path/to/layout/example/dash_layouts.py

Step by step review of a how to create and register a layout and placeholders. Note, that Dash autodiscovers your layouts by name of the file *dash_layouts.py*. The module, in which the layouts are defined, has to be named *dash_layouts.py*.

Required imports.

```
>>> from dash.base import BaseDashboardLayout, BaseDashboardPlaceholder
>>> from dash.base import layout_registry
```

Defining the Main placeholder.

```
>>> class ExampleMainPlaceholder(BaseDashboardPlaceholder):
>>>     uid = 'main' # Unique ID of the placeholder.
>>>     cols = 6 # Number of columns in the placeholder.
>>>     rows = 5 # Number of rows in the placeholder.
>>>     cell_width = 150 # Width of a single cell in the placeholder.
>>>     cell_height = 110 # Height of a single cell in the placeholder.
```

Defining the Shortcuts placeholder.

```
>>> class ExampleShortcutsPlaceholder(BaseDashboardPlaceholder):
>>>      uid = 'shortcuts' # UID of the placeholder.
>>>      cols = 1 # Number of columns in the placeholder.
>>>      rows = 10 # Number of rows in the placeholder.
>>>      cell_width = 60 # Width of a single cell in the placeholder.
>>>      cell_height = 55 # Height of a single cell in the placeholder.
```

Defining and registering the Layout.

```
>>> class ExampleLayout(BaseDashboardLayout):
>>>      uid = 'example' # Layout UID.
>>>      name = 'Example' # Layout name.
>>>
>>>      # View template. Master template used in view mode.
>>>      view_template_name = 'example/view_layout.html'
>>>
>>>      # Edit template. Master template used in edit mode.
>>>      edit_template_name = 'example/edit_layout.html'
>>>
>>>      # All placeholders listed. Note, that placeholders are rendered in the
>>>      # order specified here.
>>>      placeholders = [ExampleMainPlaceholder, ExampleShortcutsPlaceholder]
>>>
>>>      # Cell units used in the entire layout. Allowed values are: 'px', 'pt',
>>>      # 'em' or '%'. In the ``ExampleMainPlaceholder`` cell_width is set to 150.
>>>      #  It means that in this particular case its' actual width would be `150px`.
>>>      cell_units = 'px'
>>>
>>>      # Layout specific CSS.
>>>      media_css = ('css/dash_layout_example.css',)
>>>
>>>      # Layout specific JS.
>>>      media_js = ('js/dash_layout_example.js',)
>>>
>>> # Registering the layout.
>>> layout_registry.register(ExampleLayout)
```

6.2 HTML templates

You custom layout should be interited from base layout templates (view or edit). Both view and edit layouts share a lot of things, still edit layout is a bit more "heavy".

- view_layout.html should inherit from “dash/layouts/base_view_layout.html”.
- edit_layout.html should inherit from “dash/layouts/base_edit_layout.html”.

Both “dash/layouts/base_view_layout.html” and “dash/layouts/base_edit_layout.html” inherit from “dash/layouts/base_layout.html”, which in its’ turn inherits from “dash/base.html”.

Note, that when rendered to HTML, each Dash template, gets a body class “layout” + layouts’ unique identifier (UID). So, the `ExampleLayout` layout would automatically get the class “layout-example”.

<body class=”layout-example”>

In case of Android layout (UID “android”) it would be as follows.

<body class=”layout-android”>

Base your layout specific custom CSS on presence of those classes.

Same goes for Placeholders. Each placeholder gets *id_* + placeholders’ UID and the classes “placeholder” and “placeholder-” + placeholders’ UID. So, the `ExampleMainPlaceholder` would look as follows.

<div id=”id_main” class=”placeholder placeholder-main”>

And the `ExampleShortcutsPlaceholder` placeholder would look as follows.

<div id=”id_shortcuts” class=”placeholder placeholder-shortcuts”>

Same goes for plugin widgets. Apart from some other classes that each plugin widget would get for positioning, it gets the “plugin” and “plugin-” + plugin UID. See the following example (for the plugin Dummy with UID “dummy”). Each plugin also gets an automatic UID on the moment when rendered. In the example below it’s the “p6d06f17d-e142-4f45-b9c1-893c38fc2b01”.

<div id=”p6d06f17d-e142-4f45-b9c1-893c38fc2b01” class=”plugin plugin-dummy”>

Layout, placeholder, plugin and plugin widget have properties for getting their HTML specific classes and IDs.

Layout (instance)

```
>>> layout.html_class
```

Placeholder (instance)

```
>>> placeholder.html_id
>>> placeholder.html_class
```

Plugin (instance)

```
>>> plugin.html_id
>>> plugin.html_class
```

Plugin widget (static call)

```
>>> plugin_widget.html_class # Static one
```

CHAPTER 7

Creating a new plugin

Dash comes with several bundled plugins. Do check their source code as example.

Making of a plugin or a plugin widget is quite simple, although there are some rules/guideles you should follow.

Let's assume that plugin is named *sample_memo*. The plugin directory should then have the following structure.

Note, that you are advised to prefix all your plugin specific media files with *dash_plugin_* for the sake of common sense.

```
>>> path/to/plugin/sample_memo/
>>> -- static
>>> |   -- css
>>> |   |   -- dash_plugin_sample_memo.css # Plugin specific CSS
>>> |   -- images
>>> |   -- js
>>> |       -- dash_plugin_sample_memo.js # Plugin specific JavaScripts
>>> -- templates
>>> |   -- sample_memo
>>> |       -- render_main.html # Plugin widget templ. for `main` Placeholder
>>> |       -- render_short.html # Plugin widget templ. for `shortcuts` Placeholder
>>> -- __init__.py
>>> -- dash_plugins.py # Where plugins and widgets are defined and registered
>>> -- dash_widgets.py # Where the plugin widgets are defined
>>> -- forms.py # Plugin configuration form
```

In some cases, you would need plugin specific overridable settings (see `dash.contrib.plugins.weather` 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 *DASH_PLUGIN_* prefix.

7.1 path/to/plugin/sample_memo/dash_plugins.py

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

7.1.1 Define and register the plugin

Required imports.

```
>>> from dash.base import BaseDashboardPlugin, plugin_registry
>>> from path.to.plugin.sample_memo.forms import SampleMemoForm
```

Defining the Sample Memo plugin.

```
>>> class SampleMemoPlugin(BaseDashboardPlugin):
>>>     uid = 'sample_memo' # Plugin UID
>>>     name = _("Memo") # Plugin name
>>>     group = _("Memo") # Group to which the plugin belongs to
>>>     form = SampleMemoForm # Plugin forms are explained later
```

Registering the Sample Memo plugin.

```
>>> plugin_registry.register(SampleMemoPlugin)
```

7.1.2 Register plugin widgets

Plugin widgets are defined in *dash_widgets.py* module (described later), but registered in the *dash_plugins.py*, which is autodiscovered by Dash.

Required imports.

```
>>> from dash.base import plugin_widget_registry
>>> from path.to.plugin.sample_memo.dash_widgets import (
>>>     SampleMemoExampleMainWidget, SampleMemoExampleShortcutWidget
>>>     )
```

Registering the Sample Memo plugin widgets for Layout *example*.

```
>>> plugin_widget_registry.register(SampleMemoExampleMainWidget)
>>> plugin_widget_registry.register(SampleMemoExampleShortcutWidget)
```

7.2 path/to/plugin/sample_memo/dash_widgets.py

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

Take into consideration, that *dash_widgets.py* is not an autodiscovered file pattern. All your plugin widgets should be registered in modules named *dash_plugins.py*.

Required imports.

```
>>> from django.template.loader import render_to_string
>>> from dash.base import BaseDashboardPluginWidget
```

Memo plugin widget for Example layout (Placeholder *main*).

```
>>> class SampleExampleMemoExampleMainWidget(BaseDashboardPluginWidget):
>>>     layout_uid = 'example' # Layout for which the widget is written
>>>     placeholder_uid = 'main' # Placeholder within the layout for which
>>>                              # the widget is written
>>>     plugin_uid = 'sample_memo' # Plugin for which the widget is written
>>>     cols = 2 # Number of widget columns
```

```
>>>     rows = 2 # Number of widget rows
>>>
>>>     def render(self, request=None):
>>>         context = {'plugin': self.plugin}
>>>         return render_to_string('sample_memo/render_main.html', context)
```

Memo plugin widget for Example layout (Placeholder *shortcuts*).

```
>>> class SampleMemoExampleShortcutWidget(SampleMemoExampleMainWidget):
>>>     placeholder_uid = 'shortcuts'
>>>     cols = 1
>>>     rows = 1
>>>
>>>     def render(self, request=None):
>>>         context = {'plugin': self.plugin}
>>>         return render_to_string('sample_memo/render_shortcuts.html', context)
```

7.3 path/to/plugin/sample_memo/forms.py

What are the plugin forms? Very simple - if plugin is configurable, it has a form. If you need to have a custom CSS or a JavaScript included when rendering a speicifc form, use Django's class Media directive in the form.

Required imports.

```
>>> from django import forms
>>> from dash.base import DashboardPluginFormBase
```

Memo form (for *Sample Memo* plugin).

```
>>> class SampleMemoForm(forms.Form, DashboardPluginFormBase):
>>>     plugin_data_fields = [
>>>         ("title", ""),
>>>         ("text", "")
>>>     ]
>>>
>>>     title = forms.CharField(label=_("Title"), required=False)
>>>     text = forms.CharField(label=_("Text"), required=True, \
>>>                            widget=forms.widgets.Textarea)
>>>
>>>     def __init__(self, *args, **kwargs):
>>>         super(MemoForm, self).__init__(*args, **kwargs)
```

Now, that everything is ready, make sure your that both layout and the plugin modules are added to `INSTALLED_APPS` for your projects' Django settings.

```
>>> INSTALLED_APPS = (
>>>     # ...
>>>     'path.to.layout.example',
>>>     'path.to.plugin.sample_memo',
>>>     # ...
>>> )
```

After it's done, go to terminal and type the following command.

```
$ ./manage.py dash_sync_plugins
```

If your HTTP server is running, you would then be able to access your dashboard.

- View URL: http://127.0.0.1:8000/dashboard/
- Edit URL: http://127.0.0.1:8000/dashboard/edit/

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

CHAPTER 8

Permissions

Plugin system allows administrators to specify the access rights to every plugin. Dash permissions are based on Django Users and User Groups. Access rights are managable via Django admin (/administration/dash/dashboardplugin/). Note, that your admin URL prefix may vary from the one given in example (it's usually "/admin/", while in example it's "/administration/"). If user doesn't have the rights to access plugin, it doesn't appear on his dashboard 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'              |
>>>-------------------------------------------------------------------------------
>>>| Video (big_video)           | John Doe             | Dashboard users       |
>>>-------------------------------------------------------------------------------
>>>| TinyMCE memo (tinymce_memo) |                      | Dashboard users       |
>>>-------------------------------------------------------------------------------
>>>| News (news)                 | Oscar, John Doe      | Staff members         |
>>>-------------------------------------------------------------------------------
>>>| URL (url)                   |                      | Dashboard users       |
>>>-------------------------------------------------------------------------------
>>>| Video (video)               |                      | Dashboard users       |
>>>-------------------------------------------------------------------------------
>>>| Dummy (dummy)               |                      | Testers               |
>>>-------------------------------------------------------------------------------
>>>| Dummy (large_dummy)         |                      | Testers               |
>>>-------------------------------------------------------------------------------
>>>| Memo (big_memo)             |                      | Dashboard users       |
>>>-------------------------------------------------------------------------------
```

CHAPTER 9

Management commands

There are several management commands.

- *dash_find_broken_dashboard_entries*. Find broken dashboard entries that occur when some plugin which did exist in the system, no longer exists.
- *dash_sync_plugins*. Should be ran each time a new plugin is being added to the Dash.
- *dash_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 10

Tuning

There are number of Dash settings you can override in the settings module of your Django project:

- *DASH_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.
- *DASH_ACTIVE_LAYOUT* (str): Active (default) layout UID. Defaults to "android".
- *DASH_LAYOUT_CELL_UNITS* (str): Allowed values for layout cell units. Defaults to ("em", "px", "pt", "%").
- *DASH_DISPLAY_AUTH_LINK* (bool): If set to True, the log out link is shown in the Dash drop-down menu. Defaults to True.

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

CHAPTER 11

Styling tips

Font Awesome is used for icons. As a convension, all icons of font-awesome are placed within a span. Next to their original class, they all should be getting an extra class “iconic”. Follow that rule when making a new layout or a plugin (HTML). It allows to make the styling easy, since icon colours could be then changed within no time.

CHAPTER 12

Bundled plugins and layouts

Dash ships with number of bundled (demo) plugins and layouts that are mainly made to demonstrate its' abilities. In order to work amoung various layouts (themes), each plugin has a single widget registered for a single layout. It's possible to unregister a bundled widget and replace it with a custom one.

12.1 Bundled plugins

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

- Dummy plugin. Mainly made for quick testing. Still, is perfect example of how to write a plugin and widgets. https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/plugins/dummy
- Image plugin. Allows users to put images on their dashboard. If you plan to make a plugin that deals with file uploads, make sure to check the source of this one first. https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/plugins/image
- Memo plugin. Allows users to put short notes on their dashboard. https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/plugins/memo
- News plugin. Shows how to embed your Django news application (front-end part of it) into a Dash plugin widget. https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/plugins/news
- RSS feed plugin. Allows users to put any RSS feed right into the dashboard. https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/plugins/rss_feed
- URL plugin. Allows users to put links to their dashboard. https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/plugins/url
- Video plugin. Allows users to put YouTube or Vimeo videos to their dashboard. https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/plugins/video
- Weather plugin. Allows to put a weather widget into dashboard. https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/plugins/weather

12.2 Bundled layouts

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

- Android (like) layout. Has two placeholders: main (6 cols x 5 rows, each block sized 150x110 px) and shortcuts (1 col x 10 rows, each block sized 60x55 px). https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/layouts/android
- Windows 8 (like) layout. Has two placeholders: main (6 cols x 4 rows, each block sized 140x135 px) and sidebar (2 cols x 4 rows, each block sized 140x135 px). https://github.com/barseghyanartur/django-dash/tree/stable/src/dash/contrib/layouts/windows8

CHAPTER 13

Naming conventions

Although you are free to name your plugins and widgets as you want (except that you should comply with PEP http://www.python.org/dev/peps/pep-0008/#function-names), there are some naming conventions introduced, that you are recommended to follow.

- **Example1x1Plugin: 1x1 example plugin**
  - Example1x1AndroidMainWidget: 1x1 widget for 1x1 example plugin (layout Android, placeholder 'main')
  - Example1x1AndroidShortcutsWidget: 1x1 widget for 1x1 example plugin (layout Android, placeholder 'shortcuts')
  - Example1x1Windows8MainWidget: 1x1 widget for 1x1 example plugin (layout Windows 8, placeholder 'main')
  - Example1x1Windows8SidebarWidget: 1x1 widget for 1x1 example plugin (layout Windows 8, placeholder 'sidebar')
- **Example2x3Plugin: 2x3 example plugin**
  - Example2x3Windows8MainWidget: 2x3 widget for 2x3 example plugin (layout Windows 8, placeholder 'main')
  - Example2x3Windows8SidebarWidget: 2x3 widget for 2x3 example plugin (layout Windows 8, placeholder 'sidebar')
- **Example6x1Plugin: 6x1 example plugin**
  - Example6x1YourLayoutSidebarWidget: 6x1 widget for 6x1 example plugin (layout Your Layout, placeholder 'main')

Available translations

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

CHAPTER 15

License

GPL 2.0/LGPL 2.1

CHAPTER 16

Support

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

CHAPTER 17

Author

Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER 18

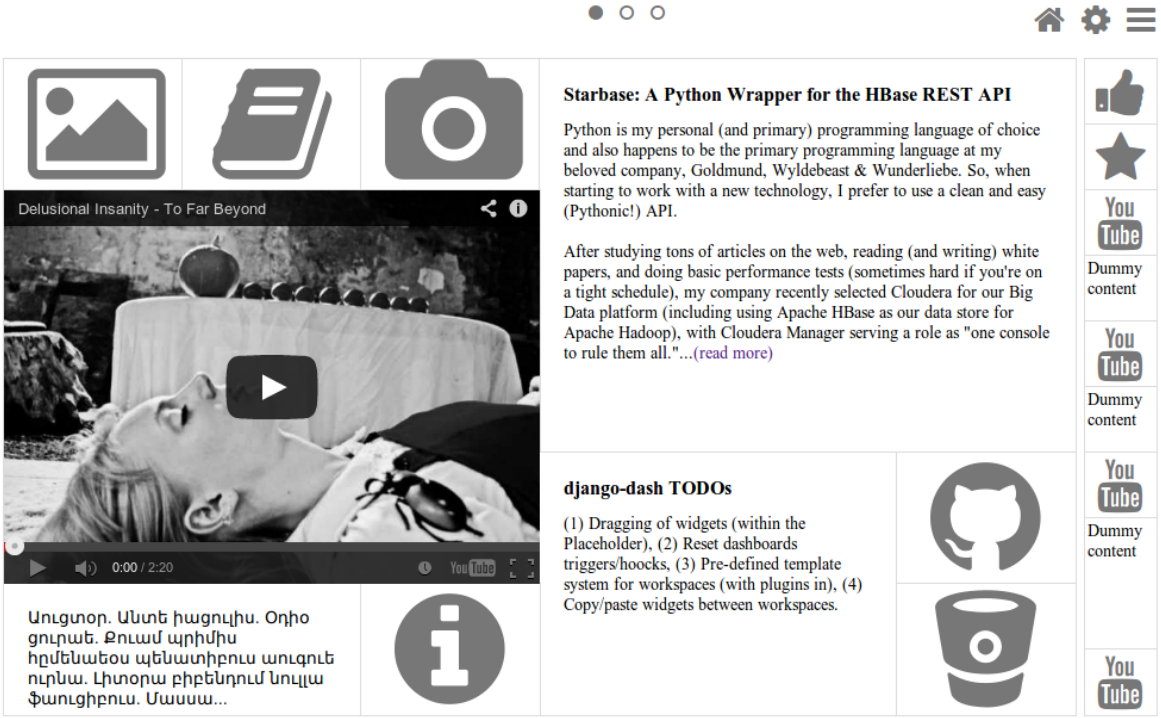
Screenshots

18.1 Android layout

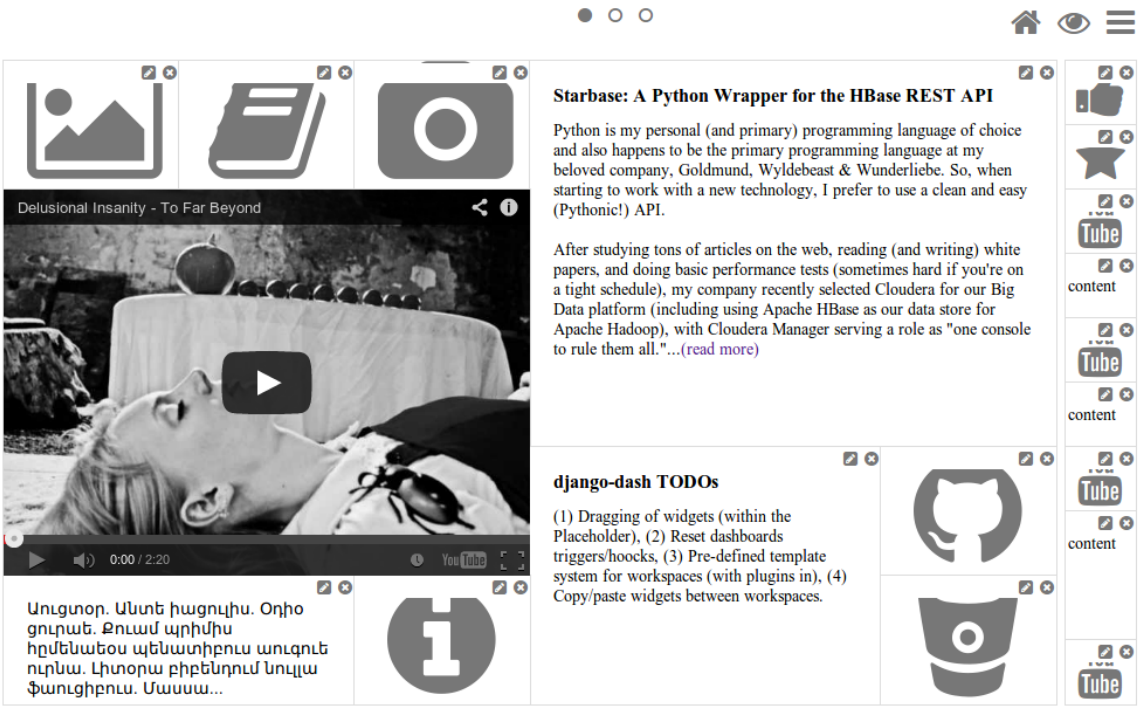
Several screenshots of Android layout are presented below.

Dashboard workspace (view mode) on which you can see the following plugins used:

- URL plugin
- TinyMCE Memo plugin
- Memo plugin
- Video plugin

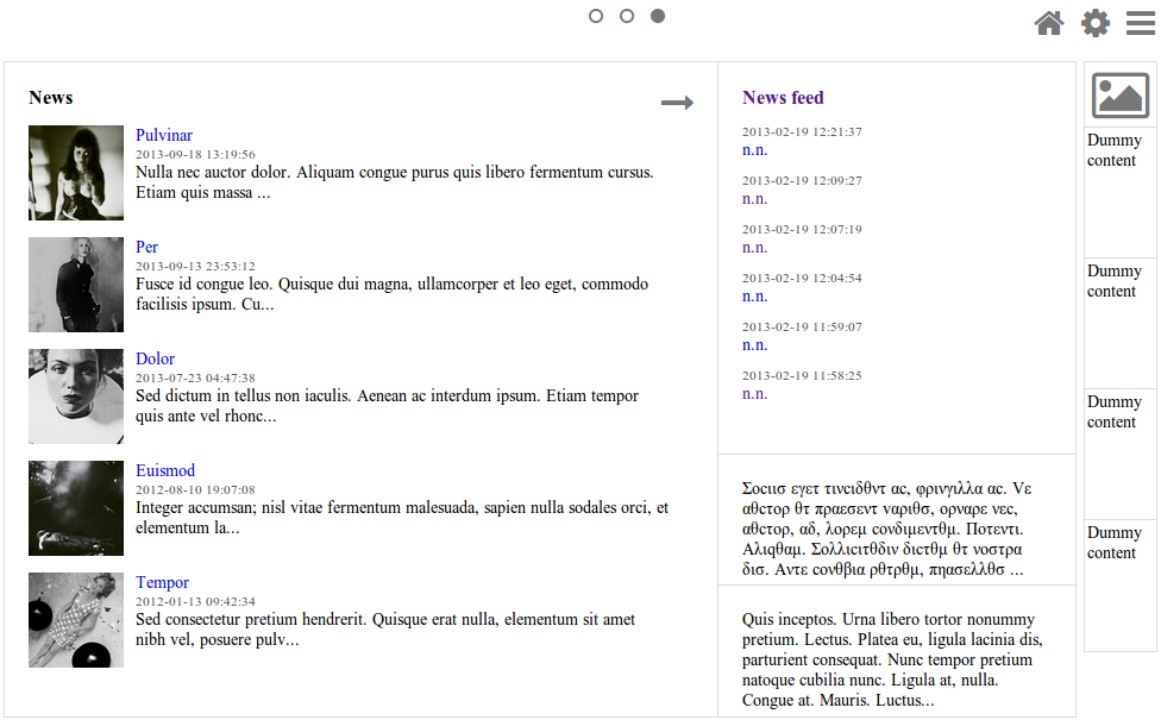


Dashboard workspace (edit mode), which is a edit mode of the previous dashboard workspace.

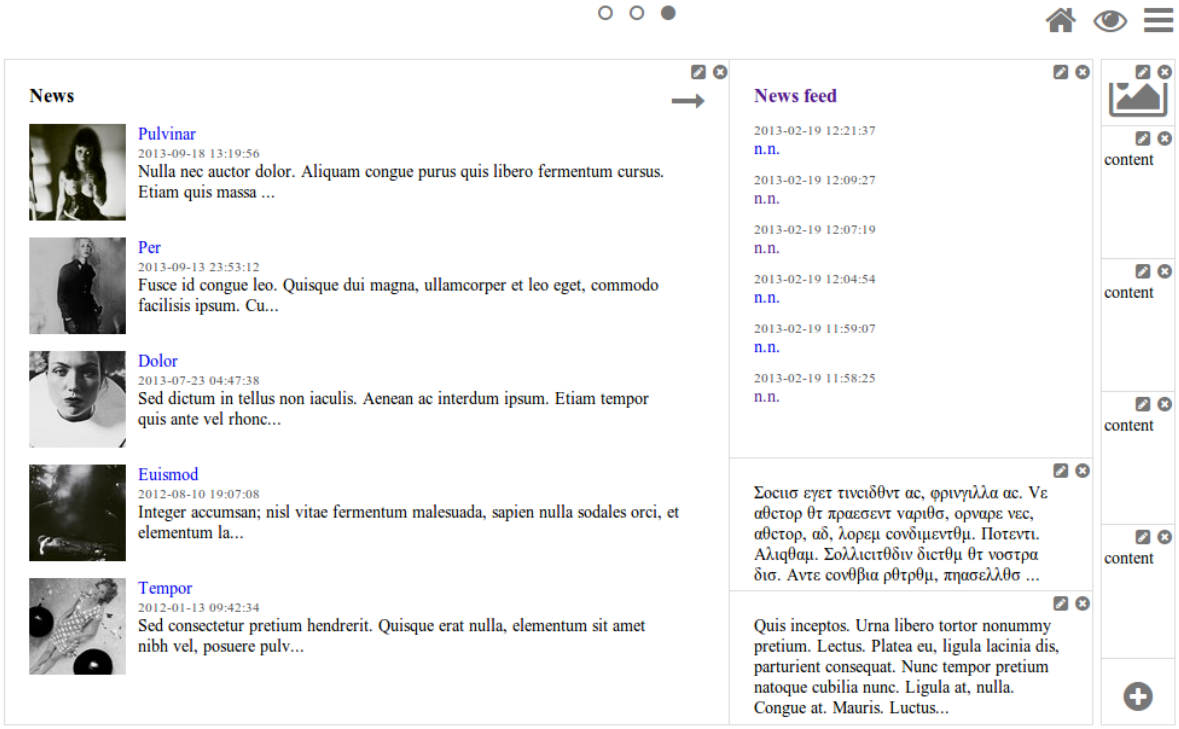


Dashboard workspace (view mode) on which you can see the following plugins used:

- News plugin
- RSS feed plugin
- Dummy plugin
- URL plugin



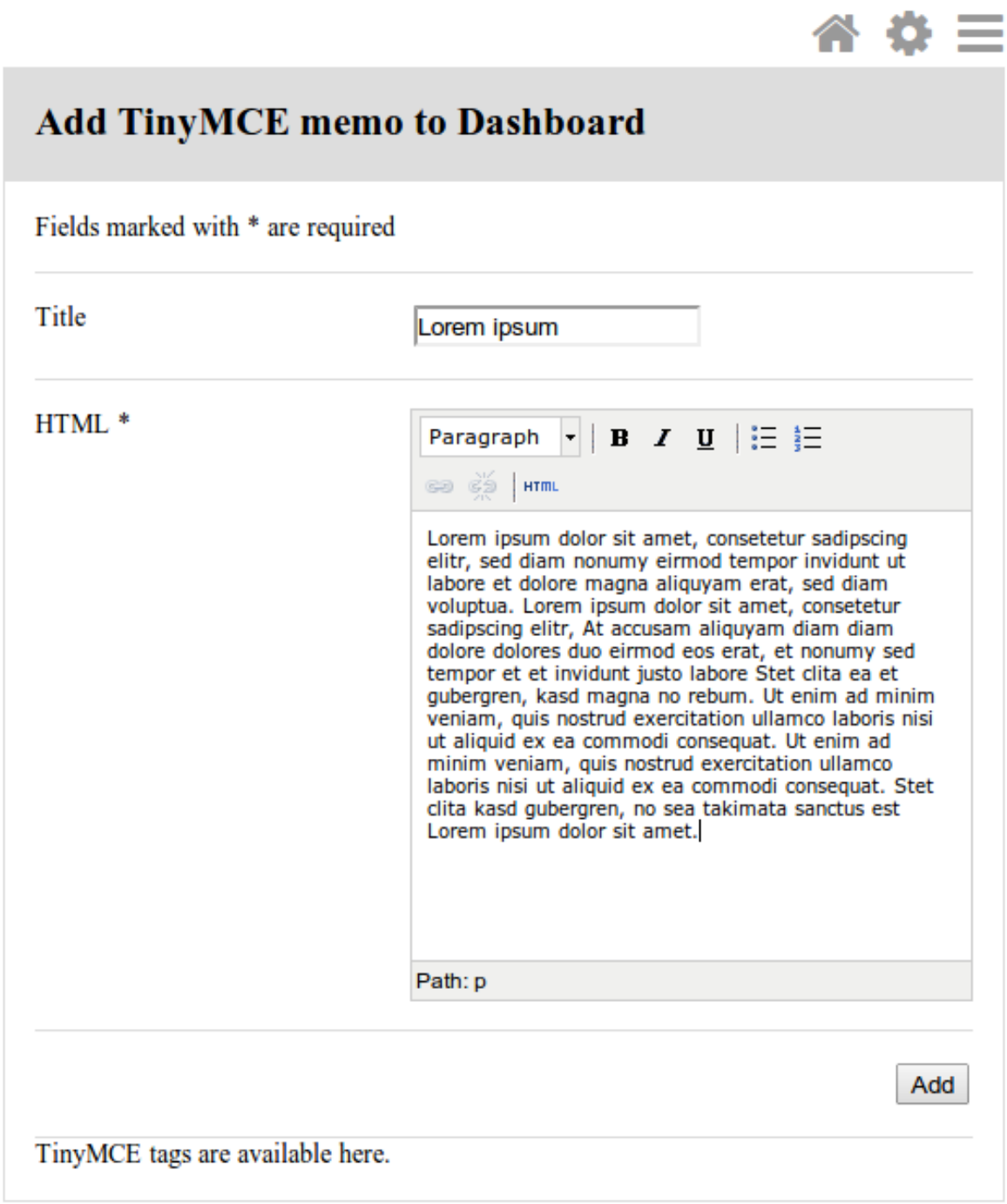
Dashboard workspace (edit mode), which is a edit mode of the previous dashboard workspace.



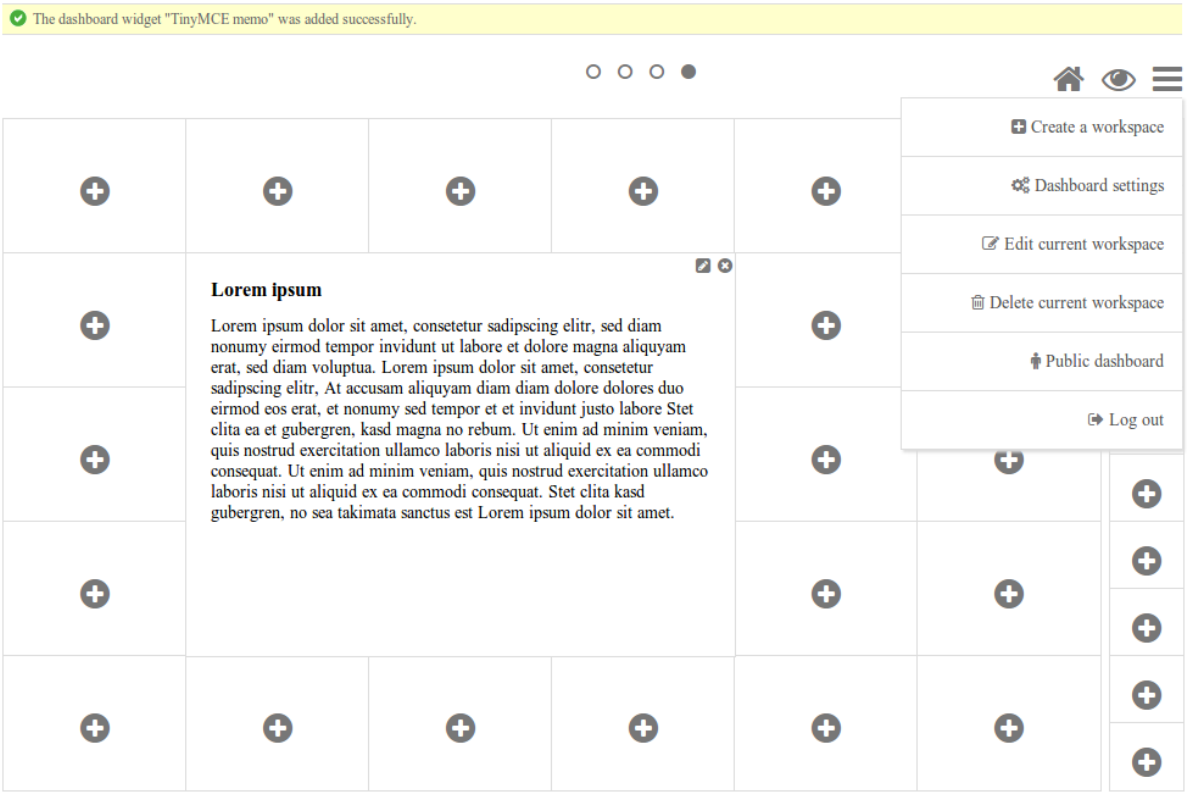
Dashboard workspace (edit mode) is an empty dashboard workspace in edit mode.

Choose widget to added to the dashboard workspace.

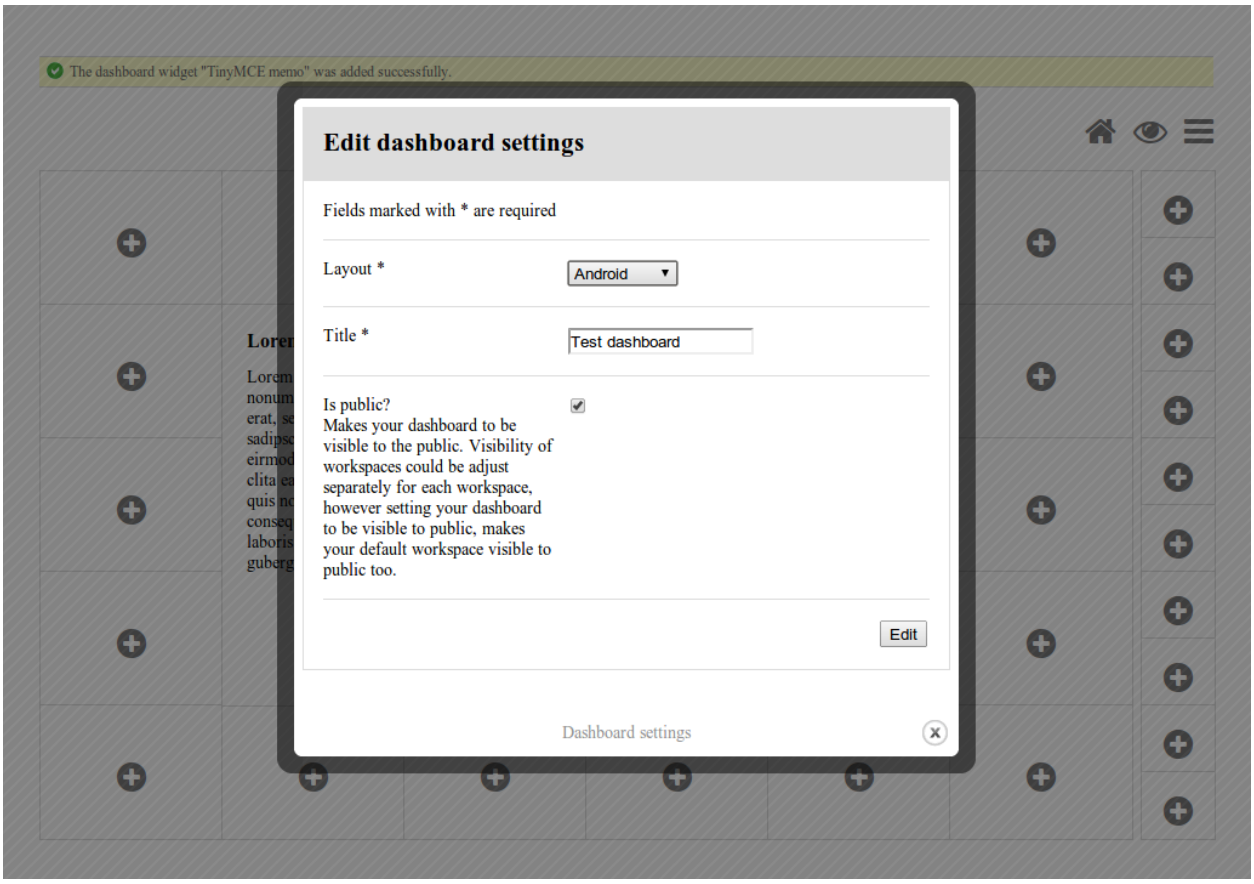
TinyMCE plugin widget form



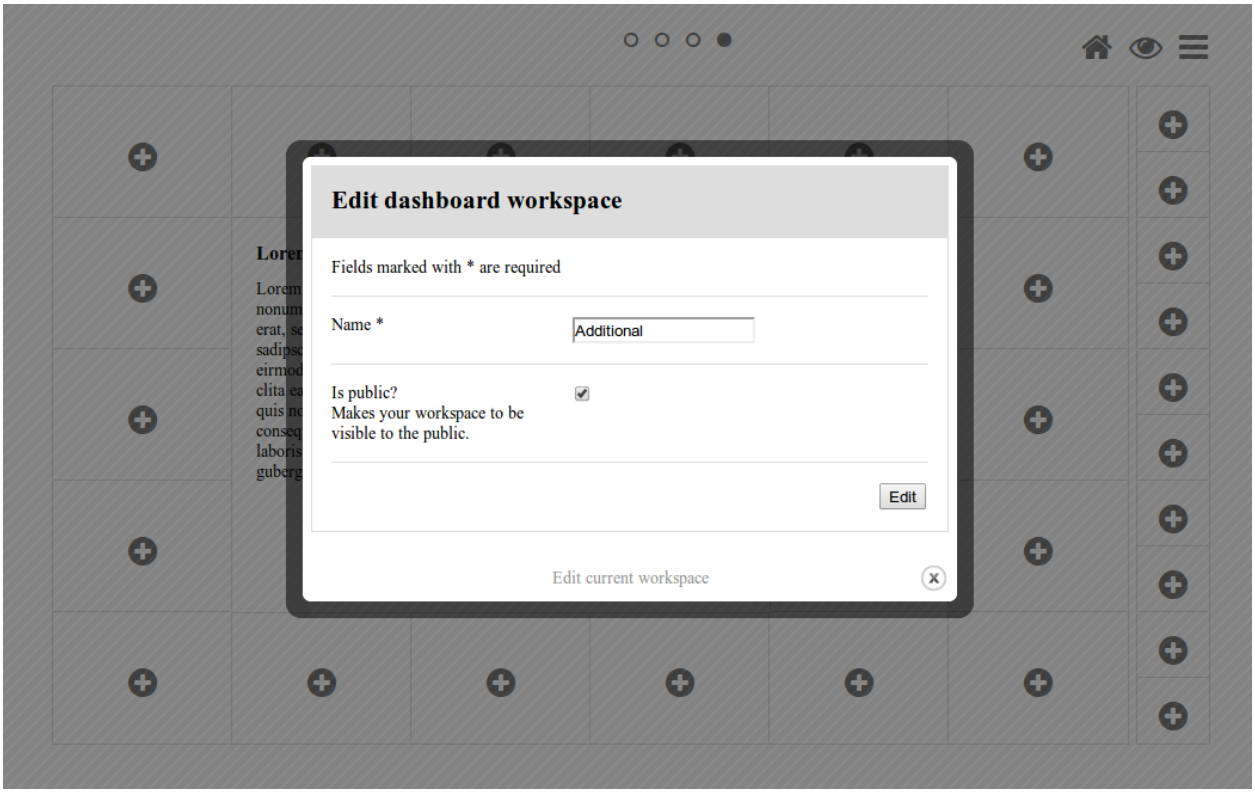
Dashboard workspace (edit mode) on which the TinyMCE plugin widget has been just added. Menu is unfolded.



A form to edit global dashboard settings.



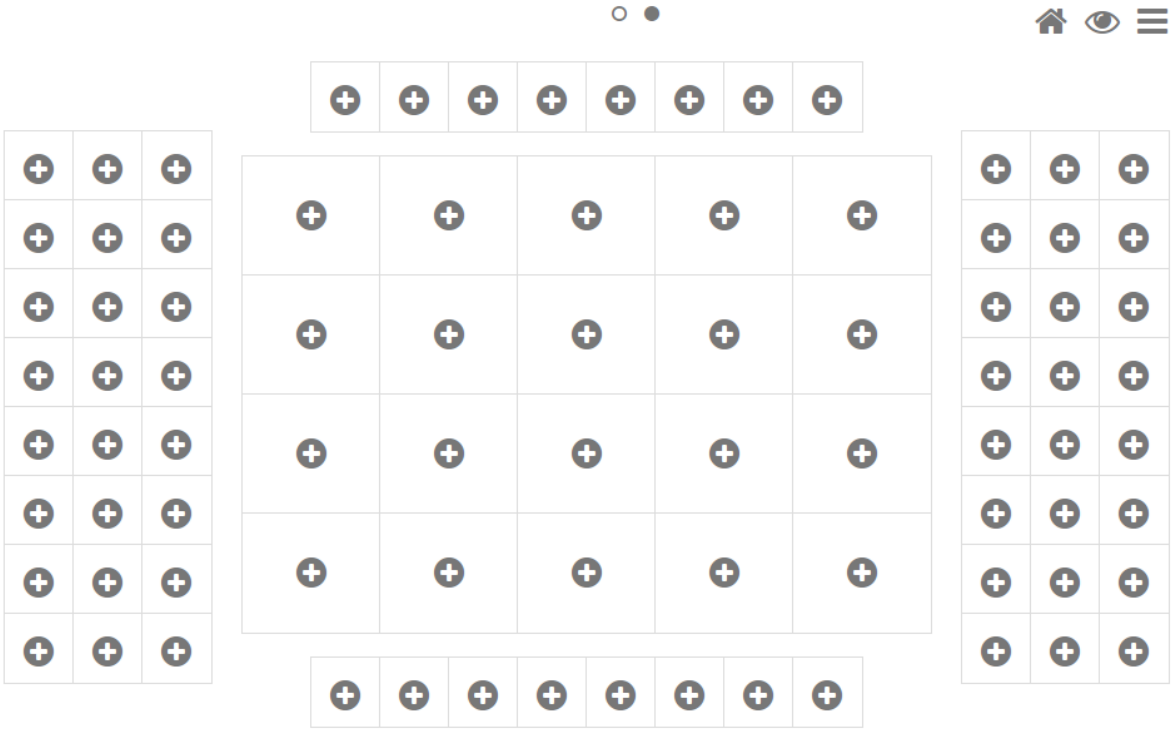
A form to edit settings of current dashboard workspace.



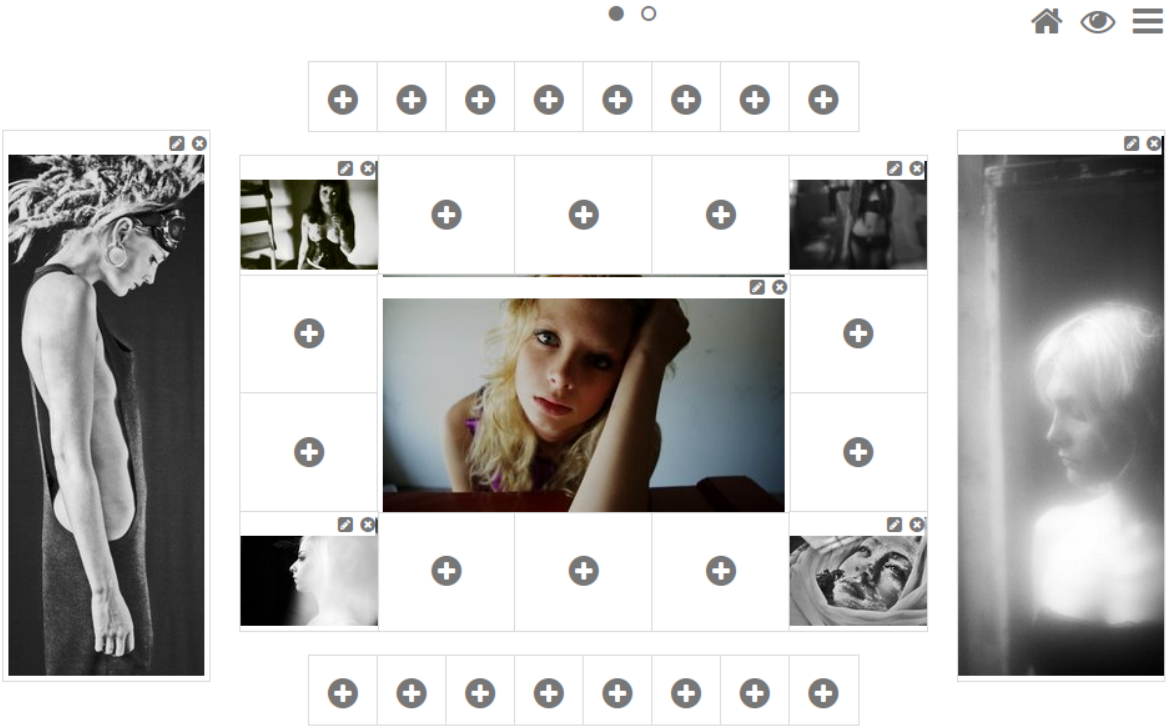
18.2 Example layout

Several screenshots of Example layout are presented below.

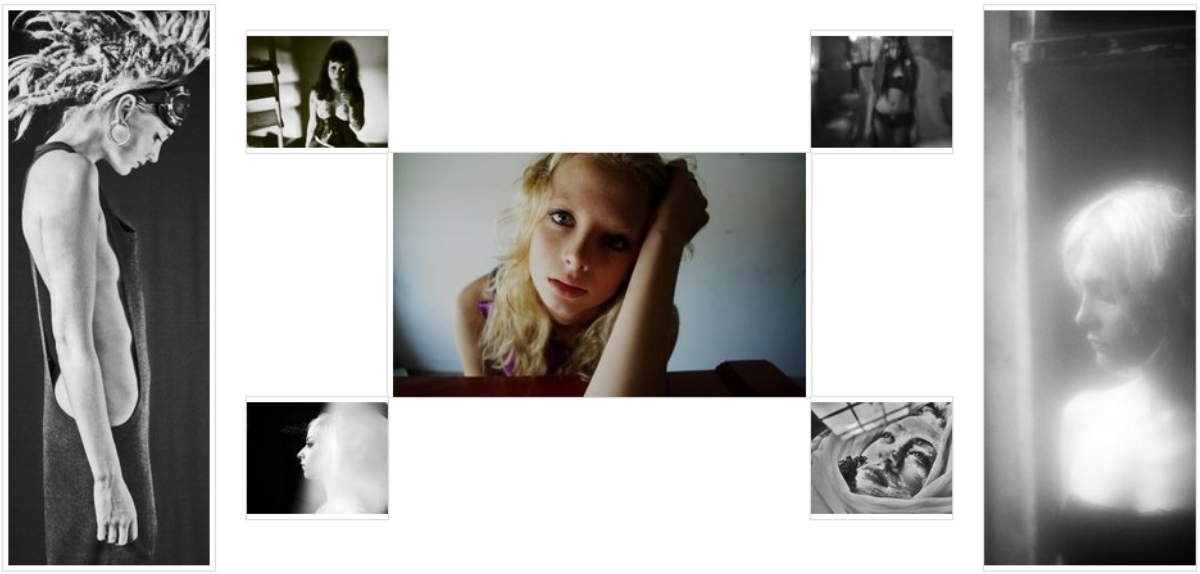
Dashboard workspace (edit mode) is an empty dashboard workspace in edit mode.



Dashboard workspace (edit mode) - previous dashboard workspace was filled with images.



Dashboard workspace (view mode) of the previows dashboard workspace



CHAPTER 19

Documentation!

Contents:

19.1 dash package

19.1.1 Subpackages

dash.contrib package

Subpackages

dash.contrib.apps package

Subpackages

dash.contrib.apps.public_dashboard package

Submodules

dash.contrib.apps.public_dashboard.urls module

dash.contrib.apps.public_dashboard.views module

dash.contrib.apps.public_dashboard.views.**public_dashboard**(*request*, *username*, *workspace=None*, *template_name='public_dashboard/public_dashboa*

Public dashboard.

Parameters

- **django.http.HttpRequest** –
- **username** (*string*) –
- **workspace** (*string*) – Workspace slug.

- **template_name** (*string*) –

Return django.http.HttpResponse

Module contents

Module contents

dash.contrib.layouts package

Subpackages

dash.contrib.layouts.android package

Submodules

dash.contrib.layouts.android.dash_layouts module

Module contents

dash.contrib.layouts.windows8 package

Submodules

dash.contrib.layouts.windows8.dash_layouts module

Module contents

Module contents

dash.contrib.plugins package

Subpackages

dash.contrib.plugins.dummy package

Submodules

dash.contrib.plugins.dummy.dash_plugins module

class dash.contrib.plugins.dummy.dash_plugins.**Dummy1x1Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.base.BaseDashboardPlugin

Dummy1x1 dashboard plugin.

form
alias of DummyForm

get_form()

group = <django.utils.functional.__proxy__ object at 0x4cd9450>

name = <django.utils.functional.__proxy__ object at 0x4cd95d0>

post_processor()
If no text available, use dummy.

uid = 'dummy_1x1'

class dash.contrib.plugins.dummy.dash_plugins.**Dummy1x2Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.dummy.dash_plugins.Dummy1x1Plugin

(Large) dummy1x2 (portrait) dashboard plugin.

group = <django.utils.functional.__proxy__ object at 0x4cb60d0>

name = <django.utils.functional.__proxy__ object at 0x4cd9810>

uid = 'dummy_1x2'

class dash.contrib.plugins.dummy.dash_plugins.**Dummy2x1Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.dummy.dash_plugins.Dummy1x1Plugin

(Large) dummy2x1 dashboard plugin.

group = <django.utils.functional.__proxy__ object at 0x4cd9750>

name = <django.utils.functional.__proxy__ object at 0x4cd96d0>

uid = 'dummy_2x1'

class dash.contrib.plugins.dummy.dash_plugins.**Dummy2x2Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.dummy.dash_plugins.Dummy1x1Plugin

Dummy2x2 dashboard plugin.

group = <django.utils.functional.__proxy__ object at 0x4cb6450>

name = <django.utils.functional.__proxy__ object at 0x4cb6590>

uid = 'dummy_2x2'

class dash.contrib.plugins.dummy.dash_plugins.**Dummy3x3Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.dummy.dash_plugins.Dummy1x1Plugin

(Big) dummy3x3 dashboard plugin.

group = <django.utils.functional.__proxy__ object at 0x4cb6510>

name = <django.utils.functional.__proxy__ object at 0x4cb6050>

uid = 'dummy_3x3'

dash.contrib.plugins.dummy.dash_widgets module

class dash.contrib.plugins.dummy.dash_widgets.**Dummy1x1AndroidMainWidget**(*plugin*)

Bases: dash.base.BaseDashboardPluginWidget

Dummy plugin widget for Android layout (placeholder *main*).

layout_uid = 'android'

media_css = []

media_js = []

placeholder_uid = 'main'

plugin_uid = 'dummy_1x1'

render(*request=None*)

class dash.contrib.plugins.dummy.dash_widgets.**Dummy1x1AndroidShortcutWidget**(*plugin*)

Bases: dash.contrib.plugins.dummy.dash_widgets.Dummy1x1AndroidMainWidget

Dummy plugin widget for Android layout (placeholder *shortcuts*).

placeholder_uid = 'shortcuts'

render(*request=None*)

class dash.contrib.plugins.dummy.dash_widgets.**Dummy1x1Windows8MainWidget**(*plugin*)

Bases: dash.contrib.plugins.dummy.dash_widgets.Dummy1x1AndroidMainWidget

Dummy plugin widget for Windows8 (placeholder *main*).

layout_uid = 'windows8'

class dash.contrib.plugins.dummy.dash_widgets.**Dummy1x1Windows8SidebarWidget**(*plugin*)

Bases: dash.contrib.plugins.dummy.dash_widgets.Dummy1x1Windows8MainWidget

Dummy plugin widget for Windows8 (placeholder *sidebar*).

placeholder_uid = 'sidebar'

class dash.contrib.plugins.dummy.dash_widgets.**Dummy1x2AndroidMainWidget**(*plugin*)

Bases: dash.contrib.plugins.dummy.dash_widgets.Dummy1x1AndroidMainWidget

Large dummy portrait plugin widget for Android (placeholder *main*).

cols = 1

plugin_uid = 'dummy_1x2'

rows = 2

class dash.contrib.plugins.dummy.dash_widgets.**Dummy1x2AndroidShortcutWidget**(*plugin*)
Bases: dash.contrib.plugins.dummy.dash_widgets.Dummy1x1AndroidShortcutWidget

Large dummy portrait plugin widget for Android (placeholder *shortcuts*).

cols = 1

plugin_uid = 'dummy_1x2'

rows = 2

class dash.contrib.plugins.dummy.dash_widgets.**Dummy2x1AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.dummy.dash_widgets.Dummy1x1AndroidMainWidget

Large dummy plugin widget for Android (placeholder *main*).

cols = 2

plugin_uid = 'dummy_2x1'

rows = 1

class dash.contrib.plugins.dummy.dash_widgets.**Dummy3x3AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.dummy.dash_widgets.Dummy1x1AndroidMainWidget

Big dummy portrait plugin widget for Android (placeholder *main*).

cols = 3

plugin_uid = 'dummy_3x3'

rows = 3

dash.contrib.plugins.dummy.defaults module

dash.contrib.plugins.dummy.forms module

class dash.contrib.plugins.dummy.forms.**DummyForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)
Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

Dummy form (for main *placeholder*).

base_fields = {'show_title': <django.forms.fields.BooleanField object at 0x4cd1e10>, 'generate_lipsum': <django.form

media

plugin_data_fields = [('show_title', False), ('generate_lipsum', False), ('lipsum_language', ''), ('lipsum_max_chars'

save_plugin_data(*request=None*)
We want to save the generated lorem ipsum text for later use. Thus, although we don't show it to the user, in case when generate_lipsum field is set to True, we silently generate the text and save it into the plugin data.

class dash.contrib.plugins.dummy.forms.**DummyShortcutsForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)

Bases: dash.contrib.plugins.dummy.forms.DummyForm

Dummy form for *shortucts* placeholder.

base_fields = {'show_title': <django.forms.fields.BooleanField object at 0x4cd1e10>, 'generate_lipsum': <django.form

media

Module contents

dash.contrib.plugins.image package

Submodules

dash.contrib.plugins.image.conf module

dash.contrib.plugins.image.conf.**get_setting**(*setting*, *override=None*)

Get a setting from dash.contrib.plugins.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.

dash.contrib.plugins.image.dash_plugins module

dash.contrib.plugins.image.dash_widgets module

class dash.contrib.plugins.image.dash_widgets.**Image1x1AndroidMainWidget**(*plugin*)

Bases: dash.base.BaseDashboardPluginWidget

Image1x1 plugin widget for Android layout (placeholder *main*).

cols = 1

layout_uid = 'android'

media_css = ('css/dash_plugin_image.css',)

media_js = ('js/dash_plugin_image.js',)

placeholder_uid = 'main'

plugin_uid = 'image_1x1'

render(*request=None*)

rows = 1

class dash.contrib.plugins.image.dash_widgets.**Image2x2AndroidMainWidget**(*plugin*)

Bases: dash.contrib.plugins.image.dash_widgets.Image1x1AndroidMainWidget

Image2x2 plugin widget for Android layout (placeholder *main*).

cols = 2

plugin_uid = 'image_2x2'

rows = 2

class dash.contrib.plugins.image.dash_widgets.**Image3x3AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.image.dash_widgets.Image1x1AndroidMainWidget

Image3x3 plugin widget for Android layout (placeholder *main*).

cols = 3

plugin_uid = 'image_3x3'

rows = 3

class dash.contrib.plugins.image.dash_widgets.**Image3x2AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.image.dash_widgets.Image1x1AndroidMainWidget

Image3x2 plugin widget for Android layout (placeholder *main*).

cols = 3

plugin_uid = 'image_3x2'

rows = 2

class dash.contrib.plugins.image.dash_widgets.**Image2x3AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.image.dash_widgets.Image1x1AndroidMainWidget

Image2x3 plugin widget for Android layout (placeholder *main*).

cols = 2

plugin_uid = 'image_2x3'

rows = 3

class dash.contrib.plugins.image.dash_widgets.**Image1x1Windows8MainWidget**(*plugin*)
Bases: dash.contrib.plugins.image.dash_widgets.Image1x1AndroidMainWidget

Image1x1 plugin widget for Windows 8 layout (placeholder *main*).

cols = 1

layout_uid = 'windows8'

placeholder_uid = 'main'

plugin_uid = 'image_1x1'

rows = 1

class dash.contrib.plugins.image.dash_widgets.**Image1x1Windows8SidebarWidget**(*plugin*)
Bases: dash.contrib.plugins.image.dash_widgets.Image1x1Windows8MainWidget

Image plugin widget for Windows 8 layout (placeholder *sidebar*).

placeholder_uid = 'sidebar'

dash.contrib.plugins.image.defaults module

dash.contrib.plugins.image.forms module

class dash.contrib.plugins.image.forms.**ImageForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)

Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

Image form for *ImagePlugin* plugin.

base_fields = {'title': <django.forms.fields.CharField object at 0x3b69810>, 'image': <django.forms.fields.ImageField

media

plugin_data_fields = [('title', ''), ('image', ''), ('fit_method', 'scale'), ('show_link', True)]

save_plugin_data(*request=None*)

Saving the plugin data and moving the file.

dash.contrib.plugins.image.helpers module

dash.contrib.plugins.image.helpers.**handle_uploaded_file**(*image_file*)

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

Return string Path to the image (relative).

dash.contrib.plugins.image.helpers.**get_crop_filter**(*fit_method*)

dash.contrib.plugins.image.helpers.**delete_file**(*image_file*)

Delete file from disc.

dash.contrib.plugins.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

dash.contrib.plugins.memo package

Submodules

dash.contrib.plugins.memo.dash_plugins module

class dash.contrib.plugins.memo.dash_plugins.**Memo1x1Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.memo.dash_plugins.Memo2x2Plugin

Memo1x1 dashboard plugin.

uid = 'memo_1x1'

class dash.contrib.plugins.memo.dash_plugins.**Memo2x2Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.base.BaseDashboardPlugin

Memo dashboard plugin.

form

alias of MemoForm

group = <django.utils.functional.__proxy__ object at 0x4ddb310>

name = <django.utils.functional.__proxy__ object at 0x4ddbed0>

uid = 'memo_2x2'

class dash.contrib.plugins.memo.dash_plugins.**Memo3x3Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.memo.dash_plugins.Memo2x2Plugin

Exact copy of the memo plugin, just rendered bigger.

uid = 'memo_3x3'

class dash.contrib.plugins.memo.dash_plugins.**Memo4x5Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.memo.dash_plugins.Memo2x2Plugin

Exact copy of the memo plugin, just rendered bigger.

uid = 'memo_4x5'

class dash.contrib.plugins.memo.dash_plugins.**TinyMCEMemo2x2Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.base.BaseDashboardPlugin

Memo dashboard plugin.

form
alias of TinyMCEMemoForm

group = <django.utils.functional.__proxy__ object at 0x4ddb590>

help_text = <django.utils.functional.__proxy__ object at 0x4ddbd50>

name = <django.utils.functional.__proxy__ object at 0x4ddb490>

uid = 'tinymce_memo_2x2'

class dash.contrib.plugins.memo.dash_plugins.**TinyMCEMemo3x3Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)
Bases: dash.contrib.plugins.memo.dash_plugins.TinyMCEMemo2x2Plugin

Exact copy of the memo plugin, just rendered bigger.

uid = 'tinymce_memo_3x3'

dash.contrib.plugins.memo.dash_widgets module

class dash.contrib.plugins.memo.dash_widgets.**Memo1x1AndroidShortcutWidget**(*plugin*)
Bases: dash.contrib.plugins.memo.dash_widgets.Memo2x2AndroidMainWidget

Memo1x1 plugin widget for Android layout (placeholder *shortcuts*).

cols = 1

placeholder_uid = 'shortcuts'

plugin_uid = 'memo_1x1'

render(*request=None*)

rows = 1

class dash.contrib.plugins.memo.dash_widgets.**Memo2x2AndroidMainWidget**(*plugin*)
Bases: dash.base.BaseDashboardPluginWidget

Memo2x2 plugin widget for Android layout (placeholder *main*).

cols = 2

layout_uid = 'android'

placeholder_uid = 'main'

plugin_uid = 'memo_2x2'

render(*request=None*)

rows = 2

class dash.contrib.plugins.memo.dash_widgets.**Memo2x2Windows8MainWidget**(*plugin*)
Bases: dash.base.BaseDashboardPluginWidget

Memo2x2 plugin widget for Windows 8 layout (placeholder *main*).

cols = 2

layout_uid = 'windows8'

placeholder_uid = 'main'

plugin_uid = 'memo_2x2'

render(*request=None*)

rows = 2

class dash.contrib.plugins.memo.dash_widgets.**Memo2x2Windows8SidebarWidget**(*plugin*)
Bases: dash.contrib.plugins.memo.dash_widgets.Memo2x2Windows8MainWidget

Memo2x2 plugin widget for Windows8 layout (placeholder *sidebar*).

cols = 1

placeholder_uid = 'sidebar'

render(*request=None*)

rows = 1

class dash.contrib.plugins.memo.dash_widgets.**Memo3x3AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.memo.dash_widgets.Memo2x2AndroidMainWidget

Memo3x3 plugin widget for Android layout (placeholder *main*).

cols = 3

plugin_uid = 'memo_3x3'

rows = 3

class dash.contrib.plugins.memo.dash_widgets.**Memo3x3Windows8MainWidget**(*plugin*)
Bases: dash.contrib.plugins.memo.dash_widgets.Memo3x3AndroidMainWidget

Memo3x3 plugin widget for Windows8 layout (placeholder *main*).

layout_uid = 'windows8'

class dash.contrib.plugins.memo.dash_widgets.**Memo3x3Windows8MainWidget**(*plugin*)
Bases: dash.contrib.plugins.memo.dash_widgets.Memo3x3AndroidMainWidget

Memo3x3 plugin widget for Windows8 layout (placeholder *main*).

layout_uid = 'windows8'

class dash.contrib.plugins.memo.dash_widgets.**Memo4x5AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.memo.dash_widgets.Memo2x2AndroidMainWidget

Huge memo plugin widget for Android layout (placeholder *main*).

cols = 4

plugin_uid = 'memo_4x5'

rows = 5

class dash.contrib.plugins.memo.dash_widgets.**TinyMCEMemo2x2AndroidMainWidget**(*plugin*)
Bases: dash.base.BaseDashboardPluginWidget

TinyMCE memo plugin widget for Android layout (placeholder *main*).

cols = 2

layout_uid = 'android'

placeholder_uid = 'main'

plugin_uid = 'tinymce_memo_2x2'

render(*request=None*)

rows = 2

class dash.contrib.plugins.memo.dash_widgets.**TinyMCEMemo3x3AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.memo.dash_widgets.TinyMCEMemo2x2AndroidMainWidget

Memo3x3 plugin widget for Android layout (placeholder *main*).

cols = 3

plugin_uid = 'tinymce_memo_3x3'

rows = 3

dash.contrib.plugins.memo.forms module

class dash.contrib.plugins.memo.forms.**MemoForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)
Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

Memo form (for Memo plugin).

base_fields = {'title': <django.forms.fields.CharField object at 0x4ddbad0>, 'text': <django.forms.fields.CharField ob

media

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

class dash.contrib.plugins.memo.forms.**TinyMCEMemoForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)
Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

TinyMCE memo form (for TinyMCEMemo plugin).

base_fields = {'title': <django.forms.fields.CharField object at 0x4ddb7d0>, 'text': <django.forms.fields.CharField ob

media

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

Module contents

dash.contrib.plugins.news package

Subpackages

dash.contrib.plugins.news.management package

Subpackages

dash.contrib.plugins.news.management.commands package

Submodules

dash.contrib.plugins.news.management.commands.news_create_test_data module

class dash.contrib.plugins.news.management.commands.news_create_test_data.**Command**
Bases: django.core.management.base.BaseCommand

Populating dummy news items.

handle(**args*, ***options*)

dash.contrib.plugins.news.management.commands.news_create_test_data.**build_image_factory**()
Downloads the https://github.com/barseghyanartur/delusionalinsanity.images/archive/latest.zip locally, unpacks it to grab the images. Then makes a list of all the images.

Return list List of relative paths to images.

dash.contrib.plugins.news.management.commands.news_create_test_data.**change_date**()

dash.contrib.plugins.news.management.commands.news_create_test_data.**fix_image**(*image*)
Fixes the image path.

Parameters **image** (*string*) – Image path.

Return string Fixed image path.

dash.contrib.plugins.news.management.commands.news_create_test_data.**split_sentences**(*f*)

dash.contrib.plugins.news.management.commands.news_create_test_data.**split_words**(*f*)

Module contents

Module contents

Submodules

dash.contrib.plugins.news.admin module

class dash.contrib.plugins.news.admin.**NewsItemAdmin**(*model*, *admin_site*)
Bases: slim.admin.SlimAdmin

Foo item admin.

class Meta

app_label = <django.utils.functional.__proxy__ object at 0x4df7a50>

NewsItemAdmin.**collapse_slim_fieldset = False**

NewsItemAdmin.**fieldsets = ((None, {'fields': ('title', 'slug', 'body', 'image')}), (<django.utils.functional.__proxy__**

NewsItemAdmin.**list_display = ('title', 'admin_image_preview', 'date_published')**

NewsItemAdmin.**media**

NewsItemAdmin.**ordering = ('-date_published',)**

NewsItemAdmin.**prepopulated_fields = {'slug': ('title',)}**

NewsItemAdmin.**readonly_fields = ('date_created', 'date_updated')**

dash.contrib.plugins.news.constants module

dash.contrib.plugins.news.dash_plugins module

class dash.contrib.plugins.news.dash_plugins.**News2x5Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.base.BaseDashboardPlugin

News plugin.

form
alias of NewsForm

group = <django.utils.functional.__proxy__ object at 0x4d76710>

name = <django.utils.functional.__proxy__ object at 0x4d76590>

post_processor()
Getting news items for the current active language.

uid = 'news_2x5'

class dash.contrib.plugins.news.dash_plugins.**News4x5Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.base.BaseDashboardPlugin

News plugin.

form
alias of NewsForm

group = <django.utils.functional.__proxy__ object at 0x4d76d90>

name = <django.utils.functional.__proxy__ object at 0x4d762d0>

uid = 'news_4x5'

dash.contrib.plugins.news.dash_widgets module

class dash.contrib.plugins.news.dash_widgets.**News2x5AndroidMainWidget**(*plugin*)

Bases: dash.base.BaseDashboardPluginWidget

News plugin widget for Android layout (placeholder *main*).

cols = 2

layout_uid = 'android'

media_css = ['css/dash_plugin_news.css']

media_js = ['js/dash_plugin_2x5_news.js']

placeholder_uid = 'main'

plugin_uid = 'news_2x5'

render(*request=None*)

rows = 5

class dash.contrib.plugins.news.dash_widgets.**News4x5AndroidMainWidget**(*plugin*)

Bases: dash.contrib.plugins.news.dash_widgets.News2x5AndroidMainWidget

Huge news plugin widget for Android layout (placeholder *main*).

cols = 4

media_css = ['css/dash_plugin_news.css']

media_js = ['js/dash_plugin_4x5_news.js']

plugin_uid = 'news_4x5'

render(*request=None*)

rows = 5

dash.contrib.plugins.news.defaults module

dash.contrib.plugins.news.forms module

class dash.contrib.plugins.news.forms.**NewsForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)

Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

Form for main NewsPlugin.

base_fields = {'show_title': <django.forms.fields.BooleanField object at 0x4d76b10>, 'max_items': <django.forms.fiel

media

plugin_data_fields = [('show_title', True), ('max_items', 6), ('truncate_after', 35), ('cache_for', 3600)]

dash.contrib.plugins.news.models module

class dash.contrib.plugins.news.models.**NewsItem**(**args*, ***kwargs*)

Bases: django.db.models.base.Model, slim.models.Slim

News item.

- *title*: Title of the news item.
- *body*: Teaser of the news item. WYSIWYG.
- *image*: Headline image of the news item.
- *date_published*: Date item is published. On creating defaults to datetime.datetime.now.
- *language*: Language.

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception NewsItem.**MultipleObjectsReturned**

Bases: django.core.exceptions.MultipleObjectsReturned

NewsItem.**admin_image_preview**()

Preview of the image. For admin use mainly.

Return string

NewsItem.**get_absolute_url**(**moreargs*, ***morekwargs*)

NewsItem.**get_language_display**(**moreargs*, ***morekwargs*)

NewsItem.**objects = <django.db.models.manager.Manager object at 0x4df7890>**

NewsItem.**translation_of**

NewsItem.**translations**

dash.contrib.plugins.news.urls module

dash.contrib.plugins.news.views module

dash.contrib.plugins.news.views.**browse**(**args*, ***kwargs*)

In the template, we show all available NewsItems for current language.

Parameters

- **request** (*django.http.HttpRequest*) –
- **template_name** (*string*) –

Return django.http.HttpResponse

dash.contrib.plugins.news.views.**detail**(*request*, *slug*, *template_name='news/detail.html'*, *template_name_ajax='news/detail_ajax.html'*)

News item detail. In the template, we show the title and the body of the News item and links to all its' all available translations.

Parameters

- **request** (*django.http.HttpRequest*) –
- **slug** (*string*) – Foo item slug.
- **template_name** (*string*) –

Return django.http.HttpResponse

Module contents

dash.contrib.plugins.rss_feed package

Subpackages

dash.contrib.plugins.rss_feed.templatetags package

Submodules

dash.contrib.plugins.rss_feed.templatetags.rss_feed_tags module

dash.contrib.plugins.rss_feed.templatetags.rss_feed_tags.**convert_to_datetime**(*value*)

Module contents

Submodules

dash.contrib.plugins.rss_feed.dash_plugins module

class dash.contrib.plugins.rss_feed.dash_plugins.**ReadRSSFeed2x3Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.base.BaseDashboardPlugin

Read RSS feed into HTML plugin.

form

alias of ReadRSSFeedForm

group = <django.utils.functional.__proxy__ object at 0x4fbca90>

name = <django.utils.functional.__proxy__ object at 0x4e451d0>

uid = 'read_rss_feed_2x3'

class dash.contrib.plugins.rss_feed.dash_plugins.**ReadRSSFeed3x3Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.rss_feed.dash_plugins.ReadRSSFeed2x3Plugin

Big read RSS feed into HTML plugin.

name = <django.utils.functional.__proxy__ object at 0x4fddc90>

uid = 'read_rss_feed_3x3'

dash.contrib.plugins.rss_feed.dash_widgets module

class dash.contrib.plugins.rss_feed.dash_widgets.**ReadRSSFeed2x3AndroidMainWidget**(*plugin*)

Bases: dash.base.BaseDashboardPluginWidget

Read RSS feed plugin widget for Android layout (placeholder *main*).

cols = 2

layout_uid = 'android'

media_css = ['css/dash_plugin_read_rss_feed.css']

media_js = ['js/dash_plugin_read_rss_feed.js']

placeholder_uid = 'main'

plugin_uid = 'read_rss_feed_2x3'

render(*request=None*)

rows = 3

class dash.contrib.plugins.rss_feed.dash_widgets.**ReadRSSFeed2x3Windows8MainWidget**(*plugin*)

Bases: dash.contrib.plugins.rss_feed.dash_widgets.ReadRSSFeed2x3AndroidMainWidget

Read RSS feed plugin widget for Windows8 (placeholder *main*).

layout_uid = 'windows8'

class dash.contrib.plugins.rss_feed.dash_widgets.**ReadRSSFeed3x3AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.rss_feed.dash_widgets.ReadRSSFeed2x3AndroidMainWidget

Big read RSS feed plugin widget for Android layout (placeholder *main*).

cols = 3

plugin_uid = 'read_rss_feed_3x3'

rows = 3

dash.contrib.plugins.rss_feed.defaults module

dash.contrib.plugins.rss_feed.forms module

class dash.contrib.plugins.rss_feed.forms.**ReadRSSFeedForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)
Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

Form for main ReadRSSFeedPlugin.

base_fields = {'feed_url': <django.forms.fields.URLField object at 0x4fdd7d0>, 'custom_feed_title': <django.forms.fi

media

plugin_data_fields = [('feed_url', ''), ('custom_feed_title', ''), ('show_feed_title', True), ('max_items', 6), ('truncate

dash.contrib.plugins.rss_feed.helpers module

dash.contrib.plugins.rss_feed.helpers.**max_num_template**(*max_items*, *default*)

dash.contrib.plugins.rss_feed.urls module

dash.contrib.plugins.rss_feed.views module

dash.contrib.plugins.rss_feed.views.**get_feed**(**args*, ***kwargs*)

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

Return django.http.HttpResponse

Module contents

dash.contrib.plugins.url package

Submodules

dash.contrib.plugins.url.conf module

dash.contrib.plugins.url.conf.**get_setting**(*setting*, *override=None*)

Get a setting from dash.contrib.plugins.url 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.

dash.contrib.plugins.url.dash_plugins module

class dash.contrib.plugins.url.dash_plugins.**URL1x1Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.base.BaseDashboardPlugin

URL dashboard plugin.

form

alias of URLForm

group = <django.utils.functional.__proxy__ object at 0x4a5db10>

html_class

If plugin has an image, we add a class *iconic* to it.

name = <django.utils.functional.__proxy__ object at 0x4a5d250>

uid = 'url_1x1'

dash.contrib.plugins.url.dash_widgets module

class dash.contrib.plugins.url.dash_widgets.**URL1x1AndroidMainWidget**(*plugin*)

Bases: dash.base.BaseDashboardPluginWidget

URL plugin widget for Android layout (placeholder *main*).

cols = 1

layout_uid = 'android'

media_css = ('css/dash_plugin_url_android.css',)

placeholder_uid = 'main'

plugin_uid = 'url_1x1'

render(*request=None*)

rows = 1

class dash.contrib.plugins.url.dash_widgets.**URL1x1AndroidShortcutWidget**(*plugin*)

Bases: dash.contrib.plugins.url.dash_widgets.URL1x1AndroidMainWidget

URL plugin widget for Android layout (placeholder *shortcuts*).

cols = 1

placeholder_uid = 'shortcuts'

render(*request=None*)

rows = 1

class dash.contrib.plugins.url.dash_widgets.**URL1x1Windows8MainWidget**(*plugin*)
Bases: dash.base.BaseDashboardPluginWidget

URL plugin widget for Windows 8 layout (placeholder *main*).

cols = 1

layout_uid = 'windows8'

media_css = ('css/dash_plugin_url_windows8.css',)

placeholder_uid = 'main'

plugin_uid = 'url_1x1'

render(*request=None*)

rows = 1

class dash.contrib.plugins.url.dash_widgets.**URL1x1Windows8SidebarWidget**(*plugin*)
Bases: dash.contrib.plugins.url.dash_widgets.URL1x1Windows8MainWidget

URL plugin widget for Windows 8 layout (placeholder *sidebar*).

placeholder_uid = 'sidebar'

dash.contrib.plugins.url.defaults module

dash.contrib.plugins.url.forms module

class dash.contrib.plugins.url.forms.**URLForm**(**args*, ***kwargs*)
Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

URL form for *URL1x1Plugin* plugin.

class Media

css = {'all': ('css/dash_plugin_url_form.css',)}

js = ('js/dash_plugin_url_form.js',)

URLForm.**base_fields = {'title': <django.forms.fields.CharField object at 0x4310510>, 'url': <django.forms.fields.UR**

URLForm.**media**

URLForm.**plugin_data_fields = [('title', ''), ('url', ''), ('external', False), ('image', '')]**

dash.contrib.plugins.url.settings module

Module contents

dash.contrib.plugins.video package

Submodules

dash.contrib.plugins.video.dash_plugins module

class dash.contrib.plugins.video.dash_plugins.**Video2x2Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.base.BaseDashboardPlugin

Video dashboard plugin.

form
alias of VideoForm

group = <django.utils.functional.__proxy__ object at 0x4df7a10>

html_class
If plugin has an image, we add a class *iconic* to it.

name = <django.utils.functional.__proxy__ object at 0x4e17490>

post_processor()

uid = 'video_2x2'

class dash.contrib.plugins.video.dash_plugins.**Video3x3Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.video.dash_plugins.Video2x2Plugin

Video dashboard plugin.

name = <django.utils.functional.__proxy__ object at 0x4df7950>

uid = 'video_3x3'

class dash.contrib.plugins.video.dash_plugins.**Video4x4Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.video.dash_plugins.Video2x2Plugin

Video dashboard plugin.

name = <django.utils.functional.__proxy__ object at 0x4df7ed0>

uid = 'video_4x4'

class dash.contrib.plugins.video.dash_plugins.**Video5x5Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)

Bases: dash.contrib.plugins.video.dash_plugins.Video2x2Plugin

Video dashboard plugin.

name = <django.utils.functional.__proxy__ object at 0x4df7690>

uid = 'video_5x5'

dash.contrib.plugins.video.dash_widgets module

class dash.contrib.plugins.video.dash_widgets.**Video2x2AndroidMainWidget**(*plugin*)
Bases: dash.base.BaseDashboardPluginWidget

Video plugin widget for Android layout (placeholder *main*).

cols = 2

layout_uid = 'android'

media_css = ('css/dash_plugin_video.css',)

placeholder_uid = 'main'

plugin_uid = 'video_2x2'

render(*request=None*)

rows = 2

class dash.contrib.plugins.video.dash_widgets.**Video3x3AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.video.dash_widgets.Video2x2AndroidMainWidget

Big video plugin widget for Android layout (placeholder *main*).

cols = 3

plugin_uid = 'video_3x3'

rows = 3

class dash.contrib.plugins.video.dash_widgets.**Video4x4AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.video.dash_widgets.Video2x2AndroidMainWidget

Huge video plugin widget for Android layout (placeholder *main*).

cols = 4

plugin_uid = 'video_4x4'

rows = 4

class dash.contrib.plugins.video.dash_widgets.**Video5x5AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.video.dash_widgets.Video2x2AndroidMainWidget

Gigantic video plugin widget for Android layout (placeholder *main*).

cols = 5

plugin_uid = 'video_5x5'

rows = 5

class dash.contrib.plugins.video.dash_widgets.**Video2x2Windows8MainWidget**(*plugin*)
Bases: dash.base.BaseDashboardPluginWidget

Video plugin widget for Windows 8 layout (placeholder *main*).

cols = 2

layout_uid = 'windows8'

media_css = ('css/dash_plugin_video.css',)

placeholder_uid = 'main'

plugin_uid = 'video_2x2'

render(*request=None*)

rows = 2

class dash.contrib.plugins.video.dash_widgets.**Video2x2Windows8SidebarWidget**(*plugin*)
Bases: dash.contrib.plugins.video.dash_widgets.Video2x2Windows8MainWidget

Video plugin widget for Windows 8 layout (placeholder *sidebar*).

placeholder_uid = 'sidebar'

dash.contrib.plugins.video.forms module

class dash.contrib.plugins.video.forms.**VideoForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=u':'*, *empty_permitted=False*)
Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

Video form for VideoPlugin plugin.

base_fields = {'title': <django.forms.fields.CharField object at 0x4e17510>, 'url': <django.forms.fields.URLField obj

media

plugin_data_fields = [('title', ''), ('url', '')]

Module contents

dash.contrib.plugins.weather package

Submodules

dash.contrib.plugins.weather.conf module

dash.contrib.plugins.weather.conf.**get_setting**(*setting*, *override=None*)
Get a setting from dash.contrib.plugins.weather 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.

dash.contrib.plugins.weather.dash_plugins module

class dash.contrib.plugins.weather.dash_plugins.**Weather2x2Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)
Bases: dash.base.BaseDashboardPlugin

Weather dashboard plugin.

form
alias of WeatherForm

group = <django.utils.functional.__proxy__ object at 0x4e28350>

name = <django.utils.functional.__proxy__ object at 0x4e28dd0>

post_processor()
If no text available, use dummy.

uid = 'weather'

class dash.contrib.plugins.weather.dash_plugins.**Weather3x3Plugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)
Bases: dash.contrib.plugins.weather.dash_plugins.Weather2x2Plugin

Big weather dashboard plugin.

group = <django.utils.functional.__proxy__ object at 0x4e28110>

name = <django.utils.functional.__proxy__ object at 0x4e28310>

uid = 'weather_3x3'

dash.contrib.plugins.weather.dash_widgets module

class dash.contrib.plugins.weather.dash_widgets.**Weather2x2AndroidMainWidget**(*plugin*)
Bases: dash.base.BaseDashboardPluginWidget

Weather plugin widget for Android layout (placeholder *main*).

cols = 2

layout_uid = 'android'

media_css = ['css/dash_plugin_weather.css']

placeholder_uid = 'main'

plugin_uid = 'weather_2x2'

render(*request=None*)

rows = 2

class dash.contrib.plugins.weather.dash_widgets.**Weather3x3AndroidMainWidget**(*plugin*)
Bases: dash.contrib.plugins.weather.dash_widgets.Weather2x2AndroidMainWidget

Big weather plugin widget for Android layout (placeholder *main*).

cols = 3

plugin_uid = 'weather_3x3'

rows = 3

dash.contrib.plugins.weather.defaults module

dash.contrib.plugins.weather.forms module

class dash.contrib.plugins.weather.forms.**WeatherForm**(**args*, ***kwargs*)
Bases: django.forms.forms.Form, dash.base.DashboardPluginFormBase

Form for main WeatherPlugin.

base_fields = {'custom_title': <django.forms.fields.CharField object at 0x4e28210>, 'show_feed_title': <django.forms

media

plugin_data_fields = [('custom_title', ''), ('show_title', True), ('cache_for', 3600), ('public_ip', ''), ('weather_data_

save_plugin_data(*request=None*)

For showing the weather, we need an IP address. Although we don't make it possible for the user to specify it manually, we silently obtain it and save into the plugin data.

dash.contrib.plugins.weather.settings module

Module contents

Module contents

Module contents

dash.management package

Subpackages

dash.management.commands package

Submodules

dash.management.commands.dash_find_broken_dashboard_entries module

class dash.management.commands.dash_find_broken_dashboard_entries.**Command**

Bases: django.core.management.base.BaseCommand

handle(**args*, ***options*)

Adds the missing plugins to database (dash.models.DashboardPlugin). This command shall be ran every time a developer adds a new plugin.

dash.management.commands.dash_sync_plugins module

class dash.management.commands.dash_sync_plugins.**Command**

Bases: django.core.management.base.BaseCommand

handle(**args*, ***options*)

Adds the missing plugins to database (dash.models.DashboardPlugin). This command shall be ran every time a developer adds a new plugin.

dash.management.commands.dash_update_plugin_data module

class dash.management.commands.dash_update_plugin_data.**Command**

Bases: django.core.management.base.BaseCommand

handle(**args*, ***options*)

Updates the plugin data for all dashboard 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.

Module contents

Module contents

dash.templatetags package

Submodules

dash.templatetags.dash_tags module

dash.templatetags.dash_tags.**get_dash_plugin**(*parser*, *token*)
Gets the plugin. Note, that dashboard_entry shall be a instance of dash.models.DashboardEntry.

Syntax {% get_dash_plugin dashboard_entry as [context_var_name] %}

Example {% get_dash_plugin dashboard_entry as plugin %}

{% get_dash_plugin dashboard_entry as plugin %} {{ plugin.render }}

Module contents

19.1.2 Submodules

19.1.3 dash.admin module

class dash.admin.**DashboardEntryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Dashboard entry admin.

class Meta

app_label = <django.utils.functional.__proxy__ object at 0x31ecd10>

DashboardEntryAdmin.**fieldsets = ((None, {'fields': ('plugin_uid', 'plugin_data', 'layout_uid', 'placeholder_uid',**

DashboardEntryAdmin.**list_display = ('plugin_uid', 'plugin_uid_code', 'plugin_data', 'layout_uid', 'placeholde**

DashboardEntryAdmin.**list_editable = ('position',)**

DashboardEntryAdmin.**list_filter = ('user', 'workspace', 'layout_uid', 'placeholder_uid', 'plugin_uid')**

DashboardEntryAdmin.**media**

DashboardEntryAdmin.**queryset**(*request*)

DashboardEntryAdmin.**readonly_fields = ('plugin_uid_code',)**

class dash.admin.**DashboardPluginAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Dashboard plugin admin.

class Meta

app_label = <django.utils.functional.__proxy__ object at 0x31ecdd0>

DashboardPluginAdmin.**fieldsets = ((None, {'fields': ('plugin_uid', 'users', 'groups')}),)**

DashboardPluginAdmin.**filter_horizontal** = ('users', 'groups')

DashboardPluginAdmin.**list_display** = ('plugin_uid_admin', 'users_list', 'groups_list')

DashboardPluginAdmin.**media**

DashboardPluginAdmin.**queryset**(*request*)

DashboardPluginAdmin.**readonly_fields** = ('plugin_uid', 'plugin_uid_admin')

class dash.admin.**DashboardSettingsAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Dashboard plugin admin.

class Meta

app_label = <django.utils.functional.__proxy__ object at 0x31ecf50>

DashboardSettingsAdmin.**fieldsets** = ((None, {'fields': ('title', 'user', 'layout_uid', 'is_public')}),)

DashboardSettingsAdmin.**list_display** = ('title', 'user', 'layout_uid', 'is_public')

DashboardSettingsAdmin.**media**

DashboardSettingsAdmin.**queryset**(*request*)

class dash.admin.**DashboardWorkspaceAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

Dashboard workspace admin.

class Meta

app_label = <django.utils.functional.__proxy__ object at 0x31ecad0>

DashboardWorkspaceAdmin.**fieldsets** = ((None, {'fields': ('name', 'position', 'is_public')}), (<django.utils.functio

DashboardWorkspaceAdmin.**list_display** = ('name', 'slug', 'layout_uid', 'position', 'user', 'is_public')

DashboardWorkspaceAdmin.**list_editable** = ('position',)

DashboardWorkspaceAdmin.**list_filter** = ('layout_uid', 'is_public')

DashboardWorkspaceAdmin.**media**

DashboardWorkspaceAdmin.**readonly_fields** = ('slug',)

19.1.4 dash.base module

All *uids* are supposed to be pythonic function names (see PEP http://www.python.org/dev/peps/pep-0008/#function-names).

class dash.base.**BaseDashboardLayout**(*user=None*)
Bases: object

Base layout.

Layouts consist of placeholders.

Properties

- *uid* (string): Layout unique identifier (globally).
- *name* (string): Layout name.

- *description* (string): Layout description.
- ***placeholders* (iterable): Iterable (list, tuple or set) of *dash.base.BaseDashboardPlaceholder*** subclasses.
- *view_template_name* (string): Temlate name used to render the layout (view).
- *edit_template_name* (string): Template named used to render the layout (edit).
- *html_classes* (string): Extra HTML class that layout should get.
- *cell_units* (string):
- *media_css* (list): List all specific stylesheets.
- *media_js* (list): List all specific javascripts.

cell_units = None

collect_widget_media(*dashboard_entries*)
Collects the widget media files.

Parameters **dashboard_entries** (*iterable*) – Iterable of `dash.models.DashboardEntry` instances.

Return list

description = None

edit_template_name = None

edit_template_name_ajax = None

get_css(*placeholders*)
Gets placeholder specific css.

Parameters **placeholders** (*iterable*) – Iterable of *dash.base.BaseDashboardPlaceholder* subclassed instances.

Return string

get_edit_template_name(*request=None*)

get_grouped_dashboard_entries(*dashboard_entries*)
Gets dashboard entries grouped by placeholder.

Parameters **dashboard_entries** (*iterable*) – Iterable of *dash.models.DashboardEntry* objects.

Return list

get_media_css()
Gets all CSS media files (for the layout + plugins).

Return list

get_media_js()
Gets all JavaScript media files (for the layout + plugins).

Return list

get_placeholder(*uid*, *default=None*)

get_placeholder_instances(*dashboard_entries=None*, *workspace=None*, *request=None*)
Gets placeholder instances.

Parameters

- **dashboard_entries** (*iterable*) – Iterable of *dash.models.DashboardEntry* objects.

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

Return list List of *dash.base.BaseDashboardPlaceholder* subclassed instances.

get_placeholder_uids (*request=None*)
Gets the list of placeholder uids.

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

Return list

get_placeholders (*request=None*)
Gets the list of placeholders registered for the layout.

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

Return itetable List of placeholder classes. Override in your layout if you need a custom behaviour.

get_view_template_name (*request=None*)

html_class
Class used in the HTML.

Return string

html_classes = []

media_css = []

media_js = []

name = None

placeholders = []

primary_html_class

render_for_edit (*dashboard_entries=None*, *workspace=None*, *request=None*)
Renders the layout.

NOTE: This is not used at the moment. You most likely want the *dash.views.edit_dashboard* view.

Parameters

- **dashboard_entries** (*iterable*) –
- **workspace** (*string*) – Current workspace.
- **request** (*django.http.HttpRequest*) –

Return string

render_for_view (*dashboard_entries=None*, *workspace=None*, *request=None*)
Renders the layout.

NOTE: This is not used at the moment. You most likely want the *dash.views.dashboard* view.

Parameters

- **dashboard_entries** (*iterable*) –
- **workspace** (*string*) – Current workspace.
- **request** (*django.http.HttpRequest*) –

Return string

uid = None

view_template_name = None

view_template_name_ajax = None

class dash.base.**BaseDashboardPlaceholder**(*layout*)
Bases: object

Base placeholder.

Properties

- *uid* (string): Unique identifier (shouldn't repeat within a single layout).

cell_height = None

cell_units

cell_width = None

cols = None

css
CSS styles for the placeholders and plugins. The placeholder dimensions as well as columns sizes, should be handled here. Since we are in a placeholder and a placeholder has a defined number of rows and columns and each reneder has just a fixed amount of rows and columns defined, we can render the top left corners generic css classes.

Cells do NOT have margins or paddings. This is essential (since all the plugins are positioned absolutely). If you want to have padding in your plugin widget, specify the *plugin-content-wrapper* class style in your specific layout/theme.

Example

.placeholder .plugin .plugin-content-wrapper { padding: 5px;

}

Return string

edit_template_name = ''

get_edit_template_name()

get_view_template_name()

html_class
Class used in the HTML.

Return string

html_classes = []

html_id
ID used in the HTML. Unique.

Return string

load_dashboard_entries(*dashboard_entries=None*)
Feed the dashboard entries to the layout for rendering later.

Parameters **dashboard_entries** (*iterable*) – Iterable of *dash.models.DashboardEntry* objects.

primary_html_class

render_for_edit()
Renders the placeholder for edit mode.

Parameters

- **workspace** (*string*) – Current workspace slug.
- **request** (*django.http.HttpRequest*) –

Return string

render_for_view()
Renders the placeholder for view mode.

Return string

rows = None

uid = None

view_template_name = ''

class dash.base.**BaseDashboardPlugin**(*layout_uid*, *placeholder_uid*, *workspace=None*, *user=None*, *position=None*)
Bases: object

Base dashboard plugin from which every plugin 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** dash.views.add_dashboard_entry and dash.views.edit_dashboard_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 *dash.views.add_dashboard_entry* default template.
- *edit_form_template* (string): Edit form template (optional). If given, overrides the *dash.views.edit_dashboard_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

delete_plugin_data()
Used in dash.views.delete_dashboard_entry. Fired automatically, when dash.models.DashboardEntry 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 dash only.

description = None

edit_form_template = None

form = None

get_form()
Get the plugin form class. Override this method in your subclassed dash.base.DashboardPlugin

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*)
Used `dash.views.add_dashboard_entry` view to gets initialised form for object to be created.

get_initialised_create_form_or_404(*data=None*, *files=None*)
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.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*, *instance=None*)
Used in `dash.views.edit_dashboard_entry` view.

get_initialised_edit_form_or_404(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *error_class=<class 'django.forms.util.ErrorList'>*, *label_suffix=':'*, *empty_permitted=False*)
Same as `get_initialised_edit_form` but raises `django.http.Http404` on errors.

get_instance()

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_position()
Gets the exact position of the plugin widget in the placeholder (row number, col number).

Return tuple Tuple of row and col numbers.

get_widget(*request=None*, *as_instance=False*)
Gets the plugin widget.

Parameters

- **request** (*django.http.HttpRequest*) –
- **as_instance** (*bool*) –

Return mixed Subclass of *dash.base.BaseDashboardPluginWidget* or instance of subclassed *dash.base.BaseDashboardPluginWidget* object.

group = 'General'

help_text = None

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

load_plugin_data(*plugin_data*)
Loads the plugin data saved in `dash.models.DashboardEntry`. Plugin data is saved in JSON string.

Parameters **plugin_data** (*string*) – JSON string with plugin data.

name = None

post_processor()
Redefine in your subclassed plugin when necessary.

Post process plugin data here (before rendering). This methid is being called after the data has been loaded into the plugin.

Note, that request (django.http.HttpRequest) is available (self.request).

pre_processor()
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*)
Init plugin with data.

process_plugin_data(*fetch_related_data=False*)
Processes the plugin data.

render(*request=None*)
Renders the plugin HTML (for dashboard workspace).

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

Return string

uid = None

update_plugin_data(*dashboard_entry*)
Used in `dash.management.commands.dash_update_plugin_data`.

Some plugins would contain data fetched from various sources (models, remote data). Since dashboard entries are by definition loaded extremely much, you are advised to store as much data as possible in `plugin_data` field of `dash.models.DashboardEntry`. 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 `dash_update_plugin_data` management command (see `dash.management.commands.dash_update_plugin_data` module).

Parameters **dash.models.DashboardEntry** – Instance of `dash.models.DashboardEntry`.

class `dash.base.`**BaseDashboardPluginWidget**(*plugin*)
Bases: `object`

Base plugin widget.

So, if we would want to register a plugin widget (renderer) for some layout, we would first define the plugin widget and then just write:

```
>>> plugin_widget_registry.register(DummyPluginWidget)
```

Plugin widget is always being registered for a placeholder. Placeholder in its' turn has number of rows and columns. Since we register each widget for a (layout, placeholder, plugin) combination separately, it fits the needs and requirements perfectly. In that way we are able to tell, wheither plugin has a widget available and actually valid (qua dimensions) for the placeholder. Plugin is just data. Nothing more. Widget operates with that data. Thus, widget has number of rows and columns it occupies in the placeholder registered. By default, number of rows and columns is set to 1, which means that a plugin occupies just 1 cell. But, certainly, there can be plugins that occupy more space in a placeholder.

cols = 1

get_height()
Gets widget height.

Return int

get_size(*delta_width=0*, *delta_height=0*)
Gets widget size.

Parameters

- **delta_width** (*int*) –
- **delta_height** (*int*) –

Return tuple

get_width()
Gets widget width.

Return int

html_class = ‘’

html_classes = []

layout_uid = None

media_css = []

media_js = []

placeholder_uid = None

plugin_uid = None

render(*request=None*)

rows = 1

dash.base.**get_registered_plugins**()
Gets a list of registered plugins in a form if tuple (plugin name, plugin description). If not yet autodiscovered, autodiscovers them.

Return list

dash.base.**get_registered_plugin_uids**()
Gets a list of registered plugin uids as a list . If not yet autodiscovered, autodiscovers them.

Return list

dash.base.**validate_plugin_uid**(*plugin_uid*)
Validates the plugin uid.

Parameters plugin_uid (*string*) –

Return bool

dash.base.**get_registered_layouts**()
Gets registered layouts.

dash.base.**get_registered_layout_uids**()
Gets uids of registered layouts.

dash.base.**get_layout**(*layout_uid=None*, *as_instance=False*)
Gets the layout by `layout_uid` given. If left empty, takes the default one chosen in settings.

Raises a `dash.exceptions.NoActiveLayoutChosen` when no default layout could be found.

Return dash.base.BaseDashboardLayout Sublcass of *dash.base.BaseDashboardLayout*.

`dash.base.`**validate_placeholder_uid**(*layout*, *placeholder_uid*)
Validates the placeholder.

Parameters

- **layout_uid** (*string*) –
- **placeholder_uid** (*string*) –

Return bool

`dash.base.`**ensure_autodiscover**()
Ensures that plugins are autodiscovered.

class `dash.base.`**DashboardPluginFormBase**
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 *dash.contrib.plugins.image.forms* as a good example.

Example

```
>>> plugin_data_fields = (
>>>     ('name', ''),
>>>     ('active': False)
>>> )
```

get_plugin_data(*request=None*)
Data that would be saved in the `plugin_data` field of the `dash.models.DashboardEntry` subclassed model.

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

plugin_data_fields = None

save_plugin_data(*request=None*)
Dummy, but necessary.

`dash.base.`**collect_widget_media**(*dashboard_entries*)
Collects the widget media for dashboard entries given.

Parameters **dashboard_entries** (*iterable*) – Iterable of `dash.models.DashboardEntry` instances.

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.

19.1.5 dash.conf module

`dash.conf.`**get_setting**(*setting*, *override=None*)
Get a setting from *dash* 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.

19.1.6 dash.decorators module

dash.decorators.**permissions_required**(*perms*, *satisfy='all'*, *login_url=None*, *raise_exception=False*)

Checks 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=[
>>>     'dash.add_dashboardentry', 'dash.change_dashboardentry', 'dash.delete_dashboardentry',
>>>     'dash.add_dashboardworkspace', 'dash.change_dashboardworkspace', 'dash.delete_dashboardw
>>>     'dash.add_dashboardsettings', 'dash.change_dashboardsettings', 'dash.delete_dashboardset
>>> ])
>>> def edit_dashboard(request):
>>>     # your code
```

dash.decorators.**all_permissions_required**(*perms*, *login_url=None*, *raise_exception=False*)

Example

```
>>> @login_required
>>> @all_permissions_required([
>>>     'dash.add_dashboardentry', 'dash.change_dashboardentry', 'dash.delete_dashboardentry',
>>>     'dash.add_dashboardworkspace', 'dash.change_dashboardworkspace', 'dash.delete_dashboardw
>>>     'dash.add_dashboardsettings', 'dash.change_dashboardsettings', 'dash.delete_dashboardset
>>> ])
>>> def edit_dashboard(request):
>>>     # your code
```

dash.decorators.**any_permission_required**(*perms*, *login_url=None*, *raise_exception=False*)

Example

```
>>> @login_required
>>> @any_permission_required([
>>>     'dash.add_dashboardentry', 'dash.change_dashboardentry', 'dash.delete_dashboardentry',
>>>     'dash.add_dashboardworkspace', 'dash.change_dashboardworkspace', 'dash.delete_dashboardw
>>>     'dash.add_dashboardsettings', 'dash.change_dashboardsettings', 'dash.delete_dashboardset
>>> ])
```

```
>>> def edit_dashboard(request):
>>>     # your code
```

dash.decorators.**edit_dashboard_permission_required**(*login_url=None*, *raise_exception=False*)

Checks if user has permissions to edit dashboard. Simply, check is successfull if any of the following permission checks are satisfied:

- Can add dashboard entry
- Can change dashboard entry
- Can delete dashboard entry
- Can add dashboard workspace
- Can change dashboard workspace
- Can delete dashboard workspace
- Can add dashboard settings
- Can change dashboard settings
- Can delete dashboard settings

Example

```
>>> @login_required
>>> @edit_dashboard_permission_required() # Do not forget the brackets!
>>> def edit_dashboard(request):
>>>     # your code
```

19.1.7 dash.defaults module

19.1.8 dash.discover module

dash.discover.**autodiscover**()

Autodiscovers files that should be found by dash.

19.1.9 dash.exceptions module

exception dash.exceptions.**InvalidRegistryItemType**

Bases: exceptions.ValueError

Raised when an attempt is made to register an item in the registry which does not have a proper type.

exception dash.exceptions.**LayoutDoesNotExist**

Bases: exceptions.Exception

Raised when layout does not exist.

exception dash.exceptions.**PluginWidgetOutOfPlaceholderBoundaries**

Bases: exceptions.Exception

Raised when plugin widget is out of placeholder boundaries.

19.1.10 dash.fields module

class dash.fields.**OrderField**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class django.db.models.fields.NOT_PROVIDED at 0x2ea0598>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=u''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=[]*, *error_messages=None*)

Bases: django.db.models.fields.IntegerField

@author http://djangosnippets.org/users/zenx/ @source http://djangosnippets.org/snippets/1861/

OrderField for models from http://ianonpython.blogspot.com/2008/08/orderfield-for-django-models.html and updated to use a django aggregation function. This field sets a default value as an auto-increment of the maximum value of the field +1.

Ignores the incoming value and instead gets the maximum plus one of the field.

This works really well in combination with "sortable_list.js". There are several things you should know:

- order field shall be null=True, blank=True
- order field shall not be unique

If above mentioned is True, you can use jQuery drag-n-drop widget in your Django-admin. See the following example:

class Media: # This belongs to your Admin class (admin.ModelAdmin)

js = ['/media/js/jquery-1.6.2.min.js', '/media/js/jquery-ui-1.8.16.custom.min.js', '/media/js/sortable_list.js'

]

formfield_(***kwargs*)

pre_save(*model_instance*, *value*)

19.1.11 dash.forms module

class dash.forms.**DashboardWorkspaceForm**(**args*, ***kwargs*)

Bases: django.forms.models.ModelForm

Dashboard workspace form.

class Meta

exclude = ('position', 'layout_uid')

model
alias of DashboardWorkspace

DashboardWorkspaceForm.**base_fields = {'user': <django.forms.models.ModelChoiceField object at 0x3638bd0>**

DashboardWorkspaceForm.**declared_fields = {}**

DashboardWorkspaceForm.**media**

19.1.12 dash.helpers module

dash.helpers.**slugify_workspace**(*s*)

dash.helpers.**lists_overlap**(*sub*, *main*)

dash.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

19.1.13 dash.models module

class dash.models.**DashboardSettings**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Dashboard settings.

Properties

- *user* (django.contrib.auth.models.User: User owning the plugin.
- *layout_uid* (str): Users' preferred layout.
- *title* (str): Dashboard title.
- *is_public* (bool): If set to True, available as public (read-only mode).

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception DashboardSettings.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

DashboardSettings.**objects = <django.db.models.manager.Manager object at 0x318eb90>**

DashboardSettings.**user**

class dash.models.**DashboardWorkspace**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Dashboard workspace.

Properties

- *user* (django.contrib.auth.models.User: User owning the plugin.
- *layout_uid* (str): Layout to which the entry belongs to.
- *name* (str): Dashboard name.
- *slug* (str): Dashboard slug.
- *position* (int): Dashboard position.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception DashboardWorkspace.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

DashboardWorkspace.**dashboardentry_set**

DashboardWorkspace.**get_absolute_url**(**moreargs*, ***morekwargs*)
Absolute URL, which goes to the dashboard workspace page.

Return string

DashboardWorkspace.**get_entries**(*user*)
Gets all dashboard entries for user given.

Parameters **user** (*django.contrib.auth.models.User*) –

Return iterable

DashboardWorkspace.**objects = <django.db.models.manager.Manager object at 0x31e5590>**

DashboardWorkspace.**user**

class dash.models.**DashboardEntry**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Dashboard entry (widget).

Since workspace can be nullable (default), we duplicate the *layout_uid*.

Properties

- *user* (django.contrib.auth.models.User: User owning the plugin.
- *workspace* (dash.models.DashboardWorkspace): Workspace to which the plugin belongs to. If left blank, entry belongs to default workspace.
- *layout_uid* (str): Layout to which the entry belongs to.
- *placeholder_uid* (str): Placeholder to which the entry belongs to.
- *plugin_uid* (str): Plugin name.
- *plugin_data* (str): JSON formatted string with plugin data.
- *position* (int): Entry position.

exception **DoesNotExist**
Bases: django.core.exceptions.ObjectDoesNotExist

exception DashboardEntry.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

DashboardEntry.**get_plugin**(*fetch_related_data=False*, *request=None*)
Gets the plugin class (by plugin_name 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 dash.base.DashboardPlugin Subclass of dash.base.DashboardPlugin.

DashboardEntry.**objects = <dash.models.DashboardEntryManager object at 0x31e5c50>**

DashboardEntry.**plugin_uid_code**()
Mainly used in admin.

DashboardEntry.**user**

DashboardEntry.**workspace**

class dash.models.**DashboardPlugin**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Dashboard plugin. Used when dash.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 dashboard plugin.
- *groups* (django.contrib.auth.models.Group): White list of the user groups allowed to use the dashboard plugin.

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception DashboardPlugin.**MultipleObjectsReturned**
Bases: django.core.exceptions.MultipleObjectsReturned

DashboardPlugin.**groups**

DashboardPlugin.**groups_list**()
Flat list (comma separated string) of groups allowed to use the dashboard plugin. Used in Django admin.

Return string

DashboardPlugin.**objects = <dash.models.DashboardPluginManager object at 0x31e76d0>**

DashboardPlugin.**plugin_uid_admin**()
Mainly used in admin.

DashboardPlugin.**plugin_uid_code**()
Mainly used in admin.

DashboardPlugin.**users**

DashboardPlugin.**users_list**()
Flat list (comma separated string) of users allowed to use the dashboard plugin. Used in Django admin.

Return string

19.1.14 dash.settings module

- *RESTRICT_PLUGIN_ACCESS* (bool): If set to True, (Django) permission system for dash plugins is enabled.
- *PLUGINS_MODULE_NAME* (str): Name of the module to placed in the (external) apps in which the dash plugin code should be implemented and registered.
- *ACTIVE_LAYOUT* (str): Active layout UID.
- *LAYOUTS_MODULE_NAME* (str): Name of the python module to be placed in (external) apps in which the dash layouts should be implemented and registered.
- *DEFAULT_WORKSPACE_NAME* (str): Name of the default workspace.
- *DEFAULT_PLACEHOLDER_VIEW_TEMPLATE_NAME* (str): Default template name for the placeholder view.
- *DEFAULT_PLACEHOLDER_EDIT_TEMPLATE_NAME* (str): Default template name for the placeholder edit.
- *LAYOUT_CELL_UNITS* (str): Layout cell units. Allowed values are *em*, *px*, *pt*, *%*.
- *DISPLAY_AUTH_LINK* (bool): If set to True, the log in or log out link is shown in the Dash drop-down menu.

- *DEBUG*

19.1.15 dash.tests module

class dash.tests.**DashBrowserTest**(*methodName='runTest'*)
Bases: django.test.testcases.LiveServerTestCase

django-dash browser tests.

TODO: At the moment is done for admin only. Normal users shall be tested as well for plugin security workflow (permissions system used).

LIVE_SERVER_URL = None

e = AttributeError("'module' object has no attribute 'LIVE_SERVER_URL'",)

classmethod setUpClass()

classmethod tearDownClass()

test_01_add_dashboard_entry(**args*, ***kwargs*)

test_02_edit_dashboard_entry(**args*, ***kwargs*)

test_03_delete_dashboard_entry(**args*, ***kwargs*)

class dash.tests.**DashCoreTest**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Tests of django-dash core functionality.

setUp()

test_01_registered_layouts(**args*, ***kwargs*)

test_02_active_layout(**args*, ***kwargs*)

test_03_get_layout_placeholders(**args*, ***kwargs*)

test_04_active_layout_render_for_view(**args*, ***kwargs*)

test_05_get_occupied_cells(**args*, ***kwargs*)

dash.tests.**create_dashboard_user**()
Create a user for testing the dashboard.

TODO: At the moment an admin account is being tested. Automated tests with diverse accounts are to be implemented.

dash.tests.**print_info**(*func*)
Prints some useful info.

dash.tests.**setup_dash**()
Set up dash.

19.1.16 dash.urls module

19.1.17 dash.utils module

dash.utils.**get_allowed_plugin_uids**(*user*)
Gets allowed plugins uids for user given.

Parameters **import User** (*django.contrib.auth.models*) –

Return list

dash.utils.**get_user_plugins**(*user*)
Gets a list of user plugins in a form if tuple (plugin name, plugin description). If not yet autodiscovered, autodiscovers them.

Return list

dash.utils.**get_user_plugin_uids**()
Gets a list of user plugin uids as a list . If not yet autodiscovered, autodiscovers them.

Return list

dash.utils.**get_widgets**(*layout*, *placeholder*, *user=None*, *workspace=None*, *position=None*, *occupied_cells=[]*)
Gets widgets. In case if in restricted mode (RESTRICT_PLUGIN_ACCESS is set to True), user argument should be provided. Based on it, the list of plugins is returned. Restrictions are bypassed in case if RESTRICT_PLUGIN_ACCESS is set to False or user given is a superuser.

Placeholders are validated already. We don't need to have validation here.

Parameters

- **layout** (*dash.base.BaseDashLayout*) – Layout object.
- **placeholder_uid** (*string*) – Placeholder uid.
- **user** (*django.contrib.auth.models.User*) –
- **workspace** (*string*) – Workspace slug.
- **position** (*int*) – Plugin position.
- **occupied_cells** (*list*) – List of already occupied cells.

Return list

dash.utils.**update_plugin_data**()
Updates the plugin data for all dashboard entries of all users. Rules for update are specified in the plugin itself.

dash.utils.**sync_plugins**()
Syncs the registered plugin list with data in dash.models.DashboardPlugin.

dash.utils.**get_workspaces**(*user*, *layout_uid=None*, *workspace=None*, *public=False*)
Gets previous, current, next and and a queryset of all workspaces.

Parameters

- **workspace** (*string*) –
- **django.contrib.auth.models.User** –

Return dict

dash.utils.**build_cells_matrix**(*user*, *layout*, *placeholder*, *workspace=None*)
Builds the cells matrix.

Parameters

- **user** (*django.contrib.auth.models.User*) –
- **placeholder_uid** (*string*) –
- **workspace** (*string*) – Workspace slug.

Return list List of cells occupied.

dash.utils.**get_or_create_dashboard_settings**(*user*)
Gets dashboard settings for the user given. If no settings found, creates default settings.

Parameters **django.contrib.auth.models.User** –

Return **dash.models.DashboardSettings** Returns `dash.models.DashboardSettings` instance.

dash.utils.**get_dashboard_settings**(*username*)
Gets dashboard settings for the user given. If no settings found, creates default settings.

Parameters **username** (*string*) –

Return **dash.models.DashboardSettings** Returns `dash.models.DashboardSettings` instance.

dash.utils.**get_public_dashboard_url**(*dashboard_settings*)
Gets resolved public dashboard URL (if public dashboard app is installed == present in the global urls module of the project).

Parameters **dash.models.DashboardSettings** – Instance of *dash.models.DashboardSettings*.

Return **string**

19.1.18 dash.views module

dash.views.**add_dashboard_entry**(*request*, **args*, ***kwargs*)
Add dashboard entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **placeholder_uid** (*string*) – Placeholder UID.
- **plugin_uid** (*string*) – Plugin UID.
- **workspace** (*string*) – Workspace slug.
- **position** (*int*) – If given, provided as position for the plugin (conflict resolution should take place).
- **template_name** (*string*) –
- **template_name_ajax** (*string*) – Template used for AJAX requests.

Return **django.http.HttpResponse**

dash.views.**create_dashboard_workspace**(*request*, **args*, ***kwargs*)
Create dashboard workspace.

Parameters

- **request** (*django.http.HttpRequest*) –
- **template_name** (*string*) –
- **template_name_ajax** (*string*) – Template used for AJAX calls.

Return **django.http.HttpResponse**

dash.views.**dashboard**(*request*, **args*, ***kwargs*)
Dashboard.

Parameters

- **request** (*django.http.HttpRequest*) –
- **workspace** (*string*) – Workspace slug. If given, the workspace loaded. Otherwise we deal with no workspace.

Return django.http.HttpResponse

dash.views.**dashboard_workspaces**(*request*, **args*, ***kwargs*)
Workspaces list.

Parameters

- **request** (*django.http.HttpRequest*) –
- **workspace** (*string*) – Workspace slug.
- **template_name** (*string*) –
- **template_name_ajax** (*string*) – Tempalte used for AJAX requests.

Return django.http.HttpResponse

dash.views.**delete_dashboard_entry**(*request*, **args*, ***kwargs*)
Remove dashboard entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **entry_id** (*int*) – ID of the dashboard entry to delete.

Return django.http.HttpResponse

dash.views.**delete_dashboard_workspace**(*request*, **args*, ***kwargs*)
Delete dashboard workspace.

Parameters

- **request** (*django.http.HttpRequest*) –
- **workspace_id** (*int*) – DashboardWorkspace id.
- **template_name** (*string*) –
- **template_name_ajax** (*string*) – Template used for AJAX calls.

Return django.http.HttpResponse

dash.views.**edit_dashboard**(*request*, **args*, ***kwargs*)
Edit dashboard.

Parameters

- **request** (*django.http.HttpRequest*) –
- **workspace** (*string*) – Workspace slug. If given, the workspace loaded. Otherwise we deal with no workspace.

Return django.http.HttpResponse

dash.views.**edit_dashboard_entry**(*request*, **args*, ***kwargs*)
Edit dashboard entry.

Parameters

- **request** (*django.http.HttpRequest*) –
- **entry_id** (*int*) – ID of the dashboard entry to edit.

- **template_name** (*string*) –
- **template_name_ajax** (*string*) – Tempalte used for AJAX requests.

Return django.http.HttpResponse

dash.views.**edit_dashboard_settings**(*request*, **args*, ***kwargs*)
Edit dashboard settings.

Parameters

- **request** (*django.http.HttpRequest*) –
- **template_name** (*string*) –
- **template_name_ajax** (*string*) – Template used for AJAX calls.

Return django.http.HttpResponse

dash.views.**edit_dashboard_workspace**(*request*, **args*, ***kwargs*)
Edit dashboard workspace.

Parameters

- **request** (*django.http.HttpRequest*) –
- **workspace_id** (*int*) – DashboardWorkspace ID.
- **template_name** (*string*) –
- **template_name_ajax** (*string*) – Template used for AJAX calls.

Return django.http.HttpResponse

dash.views.**plugin_widgets**(*request*, **args*, ***kwargs*)
Plugin widgets view. Lists all the widgets for the placeholder and workspace given.

Parameters

- **request** (*django.http.HttpRequest*) –
- **placeholder_uid** (*string*) – Placeholder UID.
- **position** (*int*) – Position on the dashboard to which the widget is to be added.
- **template_name** (*string*) –
- **template_name_ajax** (*string*) – Tempalte used for AJAX requests.

Return django.http.HttpResponse

19.1.19 dash.widgets module

class dash.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

19.1.20 Module contents

CHAPTER 20

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

d