Tipboard Documentation

Release 1.4.2

Allegro Group

March 06, 2017

Contents

1 Overview **3**
- 1.1 Project assumptions . . . 3

2 How to install **5**
- 2.1 Prerequisites . . . 5
- 2.2 Preparing environment for installation . . . 5
- 2.3 Installing with pip . . . 6
- 2.4 Verification . . . 6

3 Configuration **7**
- 3.1 Default configuration . . . 7
- 3.2 Launching Tipboard app . . . 7
- 3.3 Customising tile layout . . . 8

4 Tiles **11**
- 4.1 Customising tiles . . . 11
- 4.2 Color palette . . . 11
- 4.3 Common elements . . . 12
- 4.4 Library of available tiles . . . 12

5 API **35**
- 5.1 API key . . . 35
- 5.2 Available resources . . . 35

6 Extras **39**
- 6.1 `jira-ds.py` . . . 39
- 6.2 `client_code_example.py` . . . 39
- 6.3 `fabfile.py` . . . 39

7 Change Log **41**
- 7.1 1.4.1 . . . 41
- 7.2 1.4.0 . . . 41
- 7.3 1.3.1 . . . 41
- 7.4 1.3.0 . . . 42
- 7.5 1.2.0 . . . 42
- 7.6 1.1.0 . . . 42
- 7.7 1.0.0 . . . 43

8 License **45**

HTTP Routing Table 47

Contents:

CHAPTER 1

Overview

Tipboard is a system for creating dashboards, written in JavaScript and Python. Its widgets ('tiles' in Tipboard's terminology) are completely separated from data sources, which provides great flexibility and relatively high degree of possible customizations.

Because of its intended target (displaying various data and statistics in your office), it is optimized for larger screens.

Similar projects: Geckoboard, Dashing.

Project assumptions

1. Defining a dashboard layout (number of lines, columns, types of tiles etc.).
2. Clear separation between tiles and data sources.
3. Ability to create own tiles and scripts querying data sources (e.g. Jira, Bamboo, Confluence etc.).
4. Feeding tiles via REST API.

CHAPTER 2

How to install

You can install Tipboard on a variety of sensible operating systems. This guide assumes **Ubuntu Server 12.04 LTS** and presents shell command examples accordingly.

Prerequisites

Tipboard requires Python 2.7 which can be installed with this command:

```
$ sudo apt-get install python-dev python-virtualenv
```

Another dependency which needs to be satisfied before proceeding further is Redis server:

```
$ sudo apt-get install redis-server
```

Optional yet recommended packages

One of such packages is `supervisor` - it facilitates program administration (e.g. its reboot), especially if there are a few instances launched on the machine.

Based on the Tornado framework, Tipboard has a built-in server available, but a typical use case assumes communication with the world via reverse proxy (e.g. using `nginx` or `apache`).

Note: Although configuration of reverse proxy is out of scope of this manual, we would like to emphasise that Tipboard use Web Sockets – a relatively new mechanism – and thus you should ensure a server in a version that will support it (e.g. `nginx` >= 1.3.13 or `apache2` >= 2.4.6). By default Ubuntu 12.04 offers older versions – you may then use backports.

Note: It will be useful to have an updated version of `pip` (i.e. >= 1.4) and `virtualenv` (i.e. >= 1.10).

Preparing environment for installation

Start by creating a user, the privileges of whom will be used by the application (for the needs of this manual, let's create the user "pylabs"):

```
$ sudo adduser pylabs --home /home/pylabs --shell /bin/bash
$ sudo su - pylabs
```

Virtual environment

Continue by creating a virtual environment that will help you conveniently separate your instance from what you already have installed in the system (let's say we name it "tb-env"):

```
$ cd /home/pylabs
$ virtualenv tb-env
```

Activate the created virtual environment with the following command:

```
$ source /home/pylabs/tb-env/bin/activate
```

Note: It is worth saving the above line in the `~/.profile` file. As a result, the virtual environment will be activated automatically whenever you log in on the machine.

Note: Further setup assumes an activated virtual environment, which is denoted by `(tb-env)` prefix in your shell prompt.

Installing with pip

After creating and activating virtualenv, install the latest (current) version of Tipboard package available on pypi ("Python Package Index") with the following command:

```
(tb-env)$ pip install tipboard
```

Verification

To verify if installation has been successful, launch this command:

```
(tb-env)$ tipboard runserver
```

If you see the message "Listening on port..." instead of errors, it means that installation was successful and you may proceed to the next section.

CHAPTER 3

Configuration

The description below assumes that you have installed Tipboard correctly and use a default configuration that is the starting point for steps presented below (see section *How to install*).

Default configuration

First thing that you need to do after a successful installation is to create an empty config that will provide a base for your customizations. Type in this command:

```
(tb-env)$ tipboard create_project <name_of_project>
```

It will create `~/.tipboard` directory with the following content:

- `settings-local.yaml` file that defines the layout of tiles on the dashboard you are creating;
- `settings-local.py` file in which you can overwrite default (global) application settings; a description of options and their default values has been presented in this file;
- `custom_tiles` subdir to place your own tiles.

Note: Before you send anything to your tiles, you have to get your API key first, which is described in the *API key* section.

Launching Tipboard app

Having default config in place, you may launch Tipboard with the command:

```
(tb-env)$ tipboard runserver [<host>] [<port>]
```

...where `host` and `port` parameters are optional (by default these are `localhost` and `7272`; if you want the application to listen on all the network interfaces, set `host` to `0.0.0.0`).

You can now point your web browser to `http://localhost:7272` - you should see a basic, empty layout with tiles in 2 lines of 4 columns each.

Customising tile layout

As mentioned previously, the layout of tiles in a dashboard is defined by `layout_config.yaml` file. The file is in the YAML format, the description of which is beyond the scope of this manual. However, it is worth indicating that YAML has certain format requirements – **indentation should have a unified structure** (be a multiplication of a number, e.g. 4), when creating indentations **spaces should not be mixed with tabs**.

Below you can find a list of options that can be saved in the file. Indentations indicate the position of a given option in the configuration (e.g. `details` are superior to `page_title`).

```
details
    page_tile
layout
    row_X_of_Y
    col_X_of_Y
        tile_template
        tile_id
        tile
        timeout
```

where:

details
: A section that contains additional configuration parameters; for the time being it is only 'page_title'; depending on his needs, the users add other elements.

page_tile
: A section that defines the title of a page to appear in the web browser after entering the dashboard.

layout
: A section that contains a proper configuration of the tile layout.

row_X_of_Y
: Defines a row hight; a sum of Xs should equal Y.

col_X_of_Y
: Similar to above but concerns a column width in a given row.

tile_template
: The name of a tile template to be displayed (e.g. `pie_chart`, `line_chart`, `cumulative_flow`)

tile_id
: A tile identifier in a HTML document and key identifier in Redis.

title
: A title to be displayed in the upper part of the tile.

timeout
: The length (in seconds) of data life (if data is not sent during this time, you will be informed that the data is stalled). Since interval used by the application to check for those timeouts is 5 seconds, it doesn't make sense to set this value smaller than this.

  New in version 1.3.0.

The method of using `row_X_of_Y` and `col_X_of_Y` has been presented in the examples below. If you want to see how it's done "from the kitchen", and you have some basic knowledge of CSS styling, have a look here;

Note: If you want to present a lot of data on your dashboard, consider dividing all your tiles into two (or more) separate dashboards. Tiles offer a limited capacity and if you "feed" them with too much data (e.g. long lines of text), it is possible the dashboard will get broken.

Setting tiles' rotation

One of the most useful functions is defining tiles to rotate. In a single container (i.e. in one of the fields indicated by `col_X_of_Y` and `row_X_of_Y`), you may define a few tiles to be displayed in this location as items rotating at intervals defined in the configuration (similar to ads rotating on bus/tram stops, so-called citylights). To achieve that:

- add the `flip-time-xx` class to a container, where `xx` is rotation interval in seconds;
- add tile to the container.

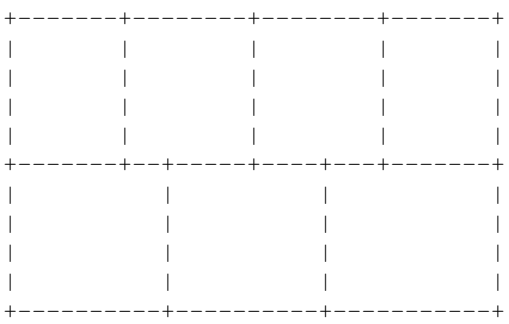
The example below presents a container with two tiles (one of the `empty` type, the other of the `text` type) to rotate every 2 seconds (`flip-time-2`). The rotation will start with the `empty` type tile:

```
layout:
  - row_1_of_2:
    - col_1_of_4 flip-time-2:
      - tile_template: empty
        tile_id: empty
        title: Empty Tile 2

      - tile_template: text
        tile_id: text
        title: Empty Tile
```

Sample layout

Let's assume we want to define a layout as on the scheme below (i.e. a division into 2 equal rows, with the upper one divided into 4 columns, and the lower one divided into 3 columns):



...its corresponding configuration file should look as follows (for brevity, I will present only the `layout` section, skipping the `tile_template`, `title_id`, etc.):

```
layout:
    row_1_of_2:
        col_1_of_4:
        col_1_of_4:
        col_1_of_4:
        col_1_of_4:
    row_1_of_2:
        col_1_of_3:
        col_1_of_3:
        col_1_of_3:
```

Multiple dashboards per application's instance

New in version 1.3.0.

It is possible to define multiple dashboards per application's instance. In order to achieve that, you just create separate layout config files (one per every dashboard) - having done that, your dashboards will be available at:

```
http://localhost:7272/<name_of_layout_config_file>
```

For example, having two layout config files `my_first_dashboard.yaml` and `my_second_dashboard.yaml`, the corresponding dashboards can be accessed via:

```
http://localhost:7272/my_first_dashboard
http://localhost:7272/my_second_dashboard
```

Note: You have to strip the `.yaml` file extension when constructing your URLs.

When it comes to feeding those dashboards with data, the future data location is specified by tile IDs (unique within application instance). Therefore, there is no need to specify different URLs for different dashboards - having tiles' IDs, Tipboard will make sure that your data is delivered where it should be.

Multiple rotating dashboards

New in version 1.3.0.

If you have defined several dashboards (as described above), you may want to rotate (flip) them periodically. If you are unsure what that means, think of extensions like Revolver (Chrome) or Tab Slideshow (Firefox).

To achieve that, you need:

- at least two dashboards (well, that's kind of obvious)
- in the file `settings-local.py` add the variable `FLIPBOARD_INTERVAL = <seconds>` (e.g. `FLIPBOARD_INTERVAL = 5`)

The above solution will make all your dashboards rotate - if you want to limit this behavior and rotate only certain dashboards, just add another parameter `FLIPBOARD_SEQUENCE` which is just a list of dashboard names that should be taken into account, e.g.:

```
FLIPBOARD_SEQUENCE = ['my_first_dashboard', 'my_third_dashboard']
```

Note: Every change in `settings-local.py` file requires restart of the application.

CHAPTER 4

Tiles

Every tile consists of an obligatory `.html` file and two optional `.css` and `.js` files. All three files belonging to a tile should have the same name that corresponds with the tile name – e.g. with the `pie_chart` tile these are `pie_chart.html`, `pie_chart.css` and `pie_chart.js` files respectively.

Customising tiles

If you want to modify a tile (e.g. change a CSS attribute, which obviously cannot be done via API), copy a desired file in the folder of tiles delivered with the application (i.e. `<path_to_your_virtualenv>/lib/python2.7/site-packages/tipboard/tiles`), paste it in your tile folder (i.e. `~/.tipboard/custom_tiles`) and edit according to your needs.

Files in your `custom_tiles` folder take precedence over those shipped by default with the application and thus you can easily replace desired elements (e.g. if you want to change the text colour, just copy and edit the `.css` file – without touching `.html` and `.js` files). We plan to introduce a command simplifying this process in the future.

Color palette

Color palette used by Tipboard's tiles is defined as shown in the table below. To retain consistency, we strongly suggest sticking to them while customising tiles.

Value	Name
#000000	black
#FFFFFF	white
#25282D	tile_background
#DC5945	red
#FF9618	yellow
#94C140	green
#12B0C5	blue
#9C4274	violet
#EC663C	orange
#54C5C0	naval

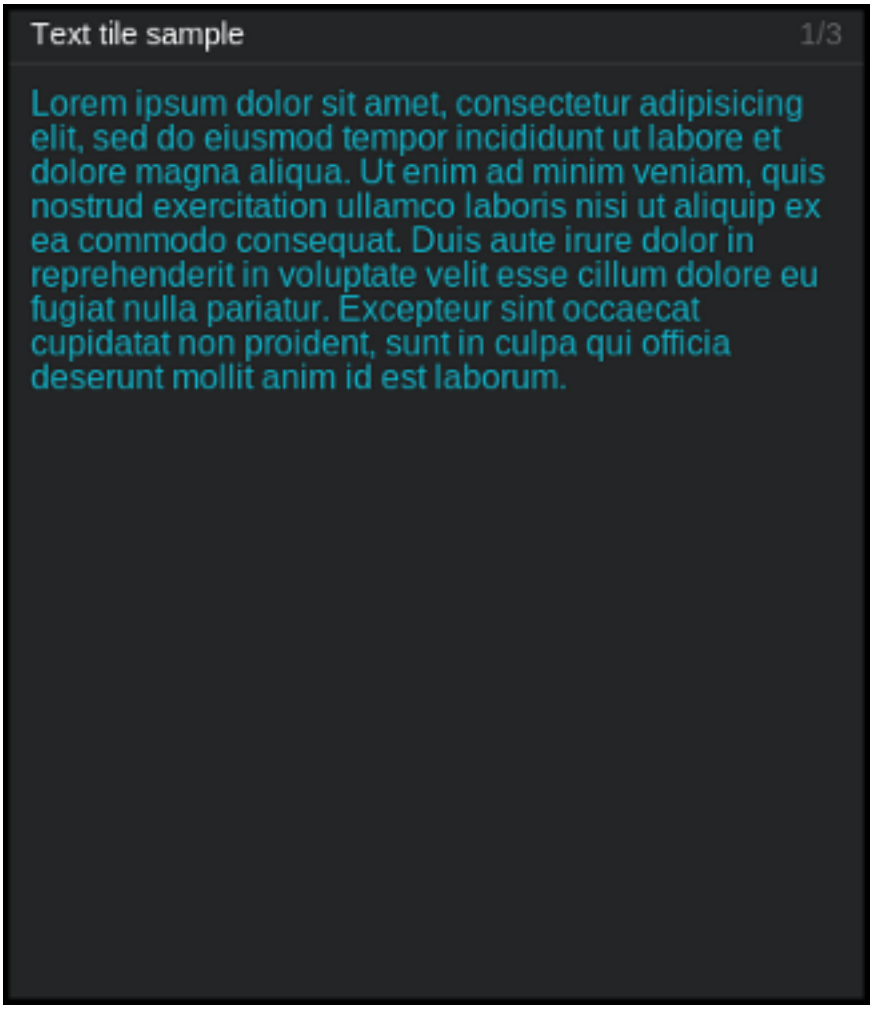
Common elements

- tile's content (`data` key) and its configuration (`value` key) should be send as two separate requests - once you have established desired configuration it does not make much sense to send it over and over again;
- in order to reset tile's config, you just send an empty `value` key (e.g. `value = {}`).

Library of available tiles

In the following pages we present a "library" of available tiles (i.e. those bundled with the application), which should serve as a specification how to send data to them and how to set up its configuration options.

`text`



Description

Simple text-tile designated to display... (surprise!) text.

Content

```
data = {"text": "<text_content>"}
```

where:

text_content
: A textual content to be displayed.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=text"
     -d "key=mytext"
     -d 'data={"text": "Hello world!"}'
```

Configuration

```
value = {"<config_element>": "<config_value>"}
```

where:

config_element
: One of three attributes of displayed text (i.e. `font_size`, `color` and `font_weight`).

config_value
: Value matching above.

Example:

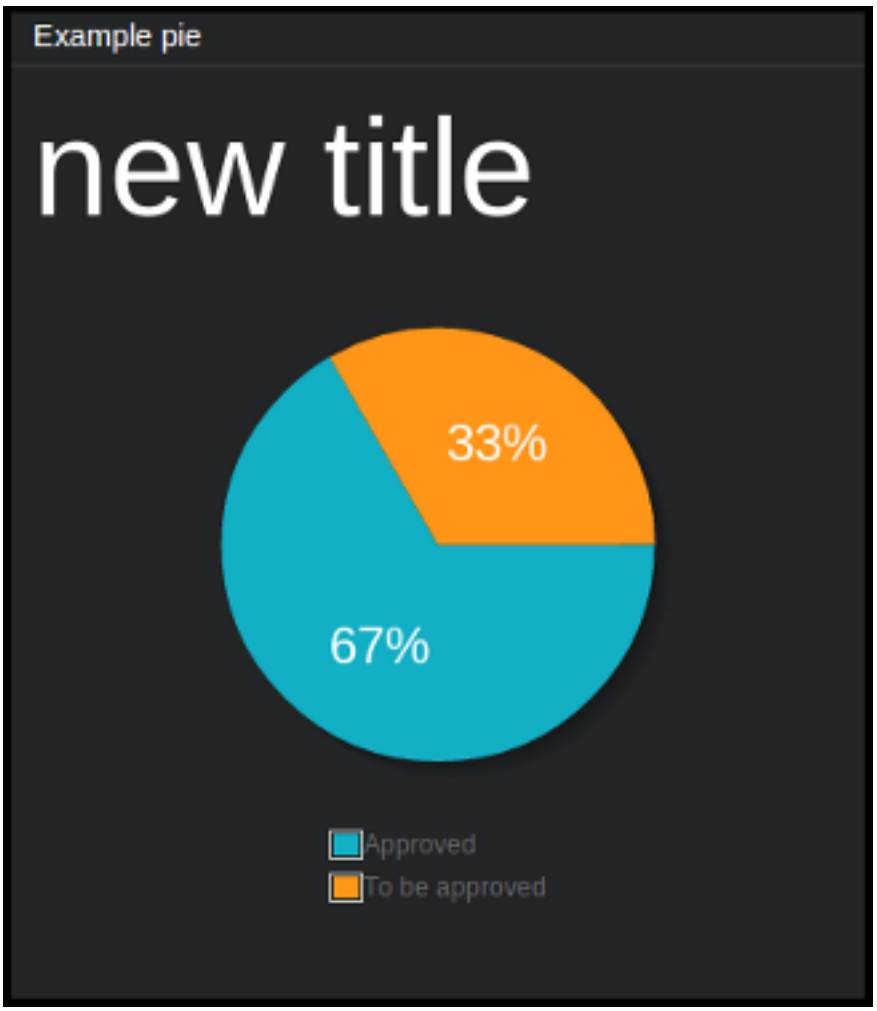
```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/mytext
     -X POST
     -d 'value={"font_color": "#00FF00"}'
```

Note: Parameter `font_size` can be specified in two ways - as a number (e.g. `"font_size": 10`) or as a string (e.g. `"font_size": "10px"`) - both of them have the same effect.

Keys `font_size`, `color`, `font_weight` with empty `config_value` are ignored (in such case, they will inherit those values from parent CSS).

---

pie_chart



Description

"Pie-chart" style chart using jqPlot library, with optional legend.

Content

```
data = {
    "title": "<optional_title>",
    "pie_data": [[identifier1, value1], [identifier2, value2], ...]
}
```

where:

title
: Chart's title (optional).

pie_data
: Data for pie-chart in a form of list of lists, where each sub-list is an identifier-value pair. Percentage of the whole chart shared by given part is calculated automatically by jqPlot - relatively to the sum of values of all parts.

Example:

```
curl http//localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=pie_chart"
     -d "key=example_pie"
     -d 'data={"title": "My title", "pie_data": [["Pie 1", 25], ["Pie 2", 25], ["Pie 3", 50]]}'
```

– this will result in a pie-chart with title “My title”, divided by three parts “Pie 1”, “Pie 2” and “Pie 3”.

Configuration

```
value = {<jqplot_config>}
```

where:

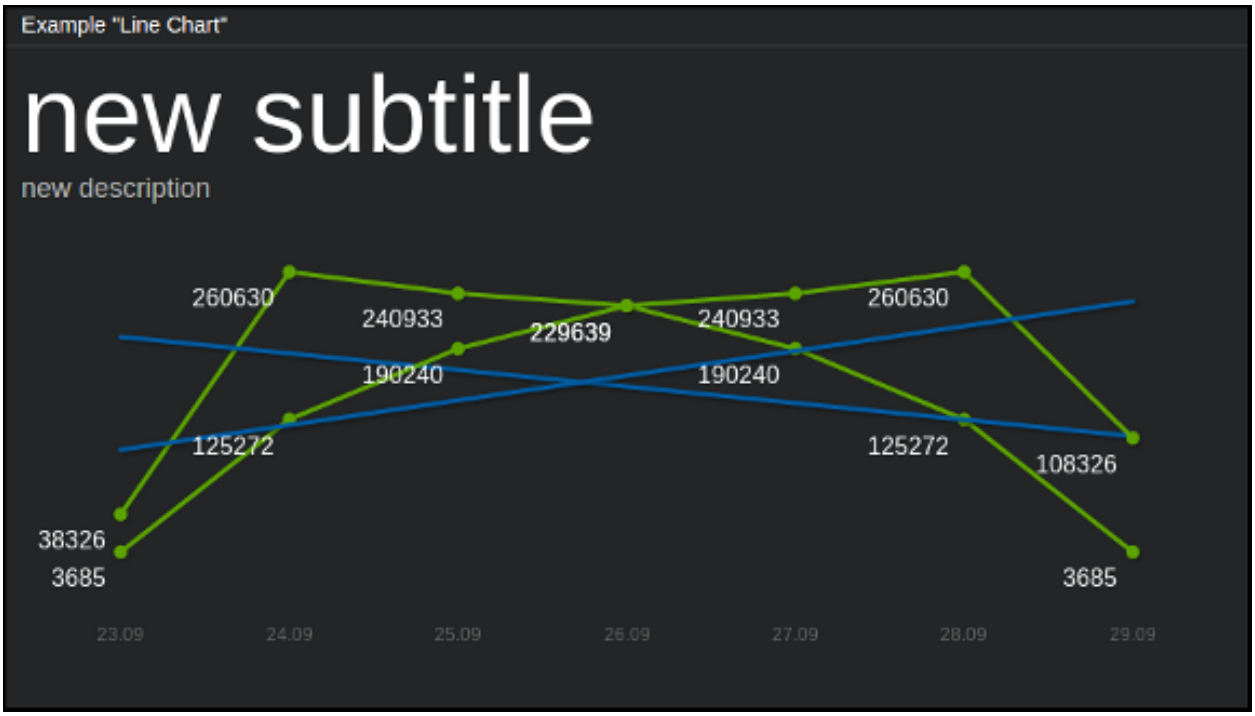
`jqplot_config`
: Configuration params in the form described by jqPlot documentation.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/<tild_id>
     -X POST
     -d 'value={"title": true, "legend": {"show": true, "location": "s"}}'
```

– this will result in a pie-chart with legend turned on at the bottom of the tile (`s` stands for “south”) - its title will be turned on as well.

`line_chart`



Description

Line-chart using jqPlot library. Allows to display arbitrary number of plots on single chart, with automatic generation of trend lines for them (which is turned on by default).

Content

```
data = {
    "subtitle": "<subtitle_text">,
    "description": "<description_text>",
    "series_list": [[<series1>], [<series2>], [<series3>], ...]
}
```

where:

subtitle, description
Additional text fields for charts descriptions (optional - you can pass empty strings here).

series_list
Data for line-charts in a form of list of series, where each series designates single chart; each element of a given series is a pair `[x_axis_value, y_axis_value]`.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=line_chart"
     -d "key=example_line"
     -d 'data={"subtitle": "averages from last week",
               "description": "Sales in our dept",
               "series_list": [[["23.09", 8326], ["24.09", 260630], ["25.09", 240933], ["26.09", 229
                               [["23.09", 3685], ["24.09", 125272], ["25.09", 190240], ["26.09", 229
```

– this will give two plots on a single chart (on x-axis there will be "23.09", "24.09", "25.09" and so on) with heading "Sales in our dept" and subtitle "averages from last week".

Configuration

```
value = {<jqplot_config>, simplify: <simplify_config>}
```

where:

jqplot_config
Configuration params in the form described by jqPlot documentation.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/example_line
     -X POST
     -d 'value={"grid": {"drawGridLines": true,
                         "gridLineColor": "#FFFFFF",
                         "shadow": false,
                         "background": "#000000",
                         "borderWidth": 0}}'
```

– this will set up the grid (in white color), black background and will turn off shadow effects as well as borders.

simplify_config

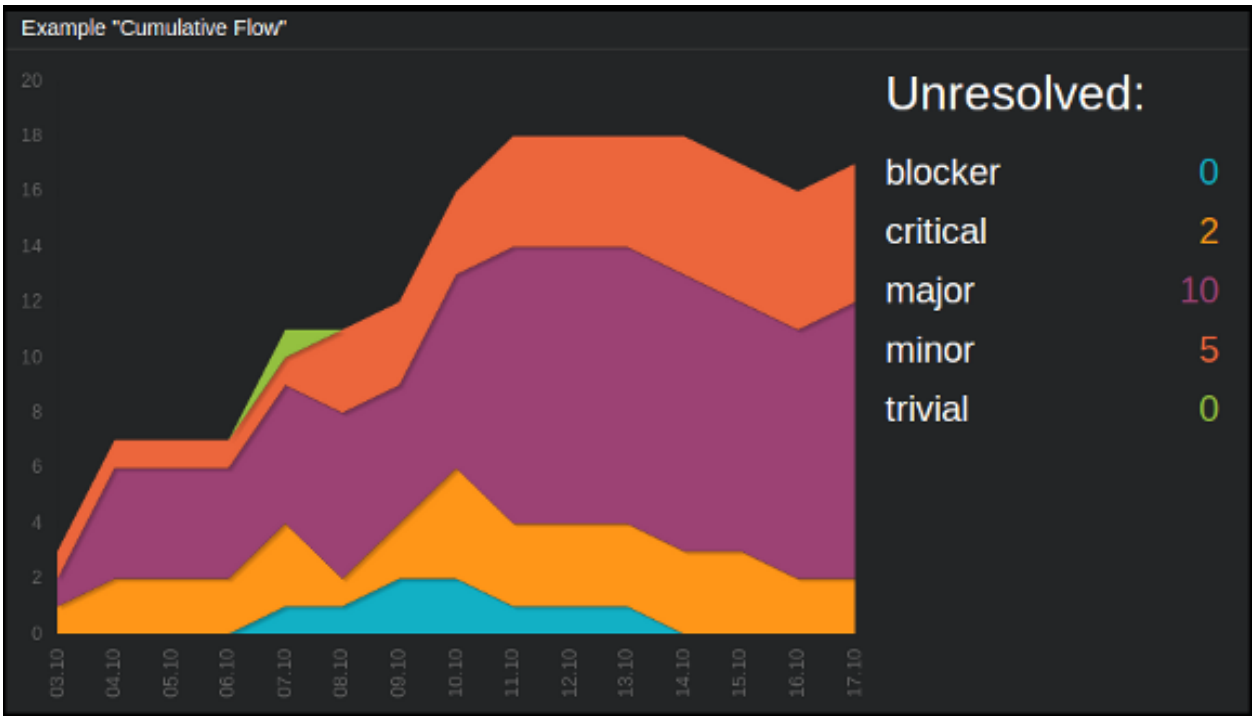
```
simplify_config = {
    tolerancy: 10,
    data_points_limit: 50, // we will TRY to achieve lower number of data points than this
    max_simplifying_steps: 5,
    simplify_step_multiplicator: 1.5
    };
```

Each option is self-describing. This feature tries to optimize dataset to achieve points count lower than *data_points_limit*. If simplify_config is not set, there won't be any simplify process at all (you will just have your raw data displayed).

```
curl -X POST http://127.0.0.1:7272/api/v0.1/dev_key/tileconfig/test_line
     -d 'value={"simplify": {"tolerancy": 2}}'
```

Note: In case of displaying multiple plots on a single chart (e.g. for more than one data series) you have to keep in mind that the `x_axis_value` values should be the same for all of those plots.

cumulative_flow



Description

Cumulative chart using jqPlot library. Allows to display up to seven plots on a single chart.

Content

```
data = {
    "title": "<title>",
    "series_list": [{"label": "<label1>", "series": [<val1>, <val2>, ...]},
                    {"label": "<label2>", "series": [<val1>, <val2>, ...]}]
}
```

where:

`title`
: Title to be displayed above the labels.

`series_list`
: A container (i.e. list of objects) for the data; each such object corresponding to a single plot consists of two keys: `label` and `series`, where the latter is a list of values constructing the plot.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
```

```
    -d "tile=cumulative_flow"
    -d "key=<tile_id>"
    -d 'data={"title": "My title:",
              "series_list": [{"label": "label 1", "series": [ 0, 0, 0, 0, 1, 1, 2, 2, 1, 1, 1, 0, 0
                              {"label": "label 2", "series": [ 0, 5, 0, 0, 1, 0, 0, 3, 0, 0, 0, 7, 8
```

Configuration

```
value = {"ticks": [[<key>, "<value>"], [<key>, "<value>"], ... ]}
```

where:

`ticks`
: List of elements defining x-axis; each such element is a list of form `[k, v]` where `k` is an ordinal number designating position of such tick and `v` is a string which will be displayed in that place.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/<tile_id>
    -X POST
    -d 'value={"ticks": [[1, "mon"], [2, "tue"], [3, "wed"], [4, "thu"], [5, "fri"], [6, "sat"], [7,
```

Note: If `series_list` contains more than one object (which is the case 99% of the time), each one of them should have `series` list of the same length - and this length should be equal to the number of `ticks`.

---

simple_percentage

Description

Tile displaying three arbitrary values (with labels) - one bigger ("main" value) and two smaller (at the bottom-left and bottom-right of the tile). It is possible to change background color for the main value.

Content

```
data = {
    "title":       "<title>",
    "subtitle":    "<subtitle>",
    "big_value":   "<value1>",
    "left_value":  "<value2>",
    "right_value": "<value3>"
    "left_label":  "<label1>",
    "right_label": "<label2>"
}
```

where:

title, subtitle
: They serve as a label for the `big_value` ("main" value).

big_value

Main value, which treated as a string, so it can contain symbols like `%` etc.

left_value, right_value
Smaller, bottom-left and bottom-right values.

left_label, right_label
Labels for above values.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=simple_percentage"
     -d "key=<tile_id>"
     -d 'data={"title": "My title", "subtitle": "My subtitle", "big_value": "100%",
               "left_value": "50%", "left_label": "smaller label 1",
               "right_value": "25%", "right_label": "smaller label 2"}'
```

Configuration

```
value = {
    "big_value_color": "<color>",
    "fading_background": <BOOLEAN>
}
```

where:

big_value_color
Background color for `big_value` in a hexadecimal form or color name (e.g. `#94C140` or `green`).

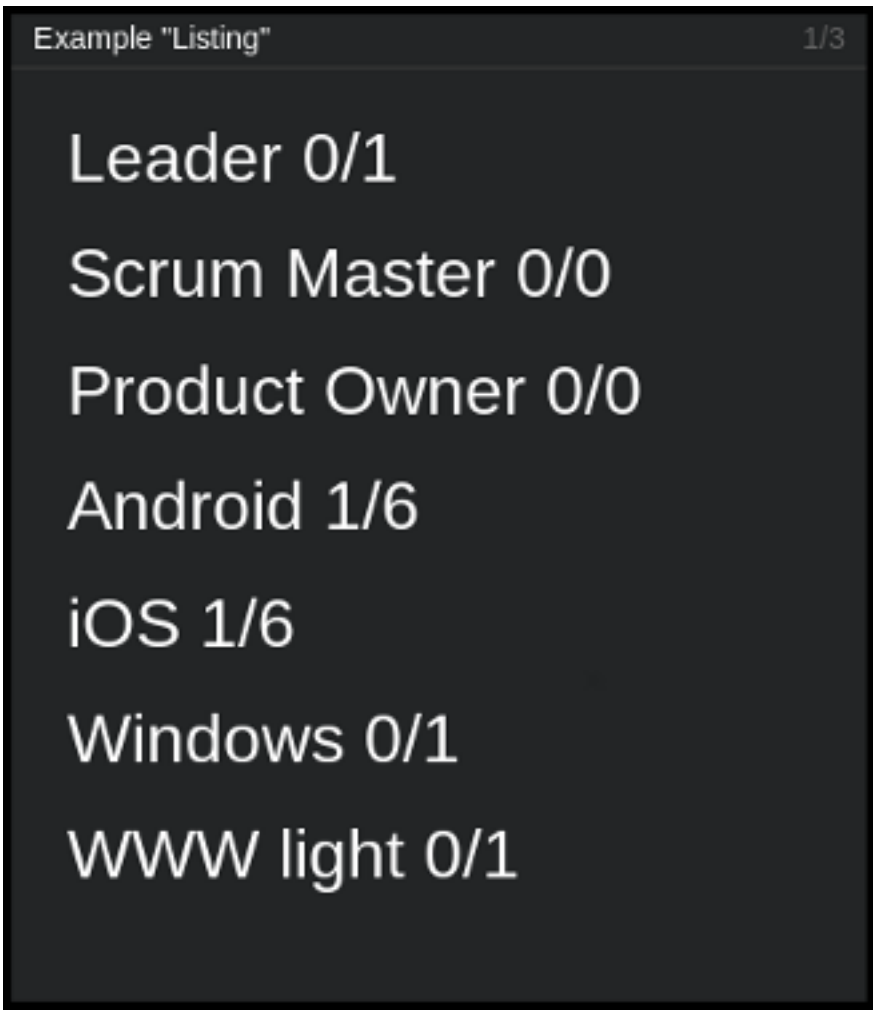
fading_background
Turns on/off background pulsation for `big_value` (may be useful for alerts etc.).

New in version 1.3.0.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/<tile_id>
     -X POST
     -d 'value={"big_value_color": "green", "fading_background": true}'
```

`listing`



Description

Very simple tile for displaying list of entries (up to seven) of arbitrary content. For more sophisticated needs there is a `fancy_listing` tile.

Content

```
data = {"items": ["<entry1>", "<entry2>", ..., "<entry7>"]}
```

where:

`items`
: List of items (entries) to display.

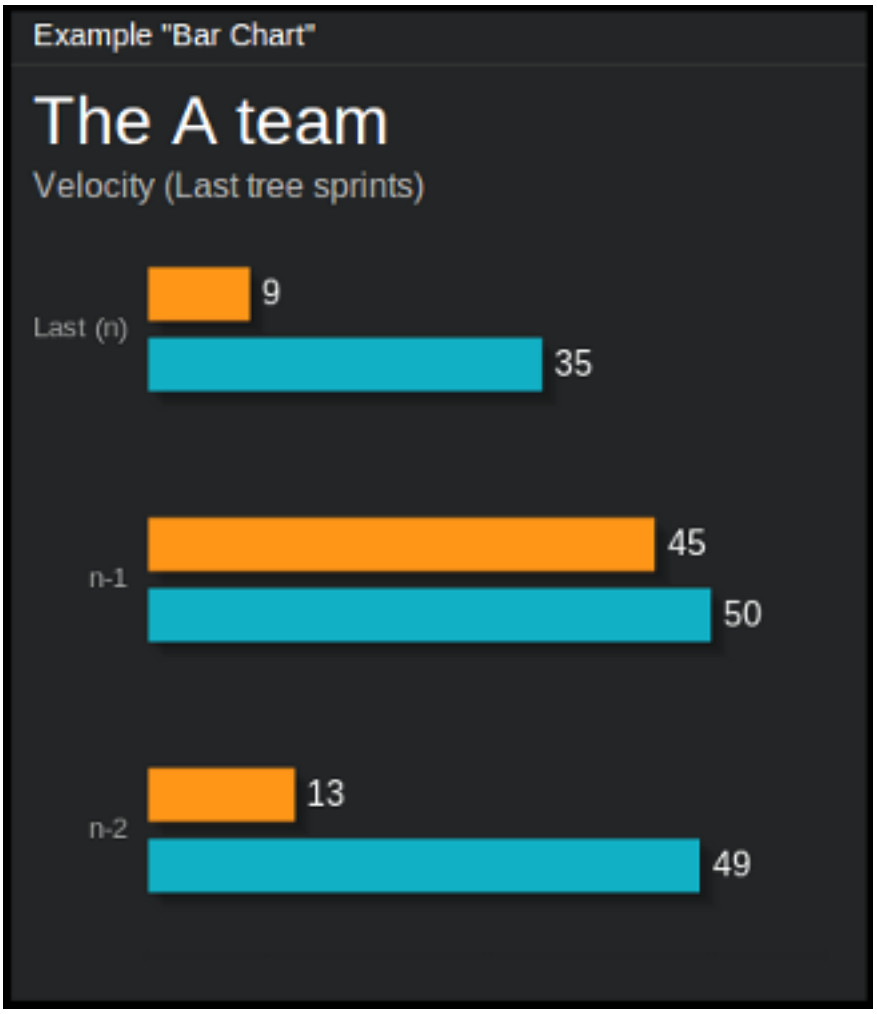
Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=listing"
     -d "key=<tile_id>"
     -d "data={"items": ["Leader: 5", "Product Owner: 0", "Scrum Master: 3", "Developer: 0"]}"
```

Configuration

This tile does not offer any configuration options.

bar_chart



Description

Tile displaying data in the form of horizontal bar charts. Each cart may consist of one or more series of data - in the latter case, such bars are grouped respectively. There are no "hard" limits when it comes to the number of charts/series, although we suggest to keep them down to 3 charts of 2 series each.

Content

```
data = {
    "title": "<title>",
    "subtitle": "<subtitle>",
    "ticks": ["<label1>", "<label2>", "<label3>" ...],
    "series_list": [[<val1>, <val2>, <val3>, ...],
                    [<val1>, <val2>, <val3>, ...]]
}
```

where:

title, subtitle
Title and subtitle displayed on the top of the tile.

ticks
Labels to be displayed on the left side of the charts.

series_list
List of series, as name suggests. `[[1, 2]]` will give one chart of two bars, `[[3, 4, 5], [6, 7, 8]]` will give two charts of three bars each.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=bar_chart"
     -d "key=<tile_id>"
     -d 'data={"title": "The A-Team",
               "subtitle": "Velocity (Last tree sprints)",
               "ticks": ["n-2", "n-1", "Last (n)"],
               "series_list": [[49, 50, 35], [13, 45, 9]]}'
```

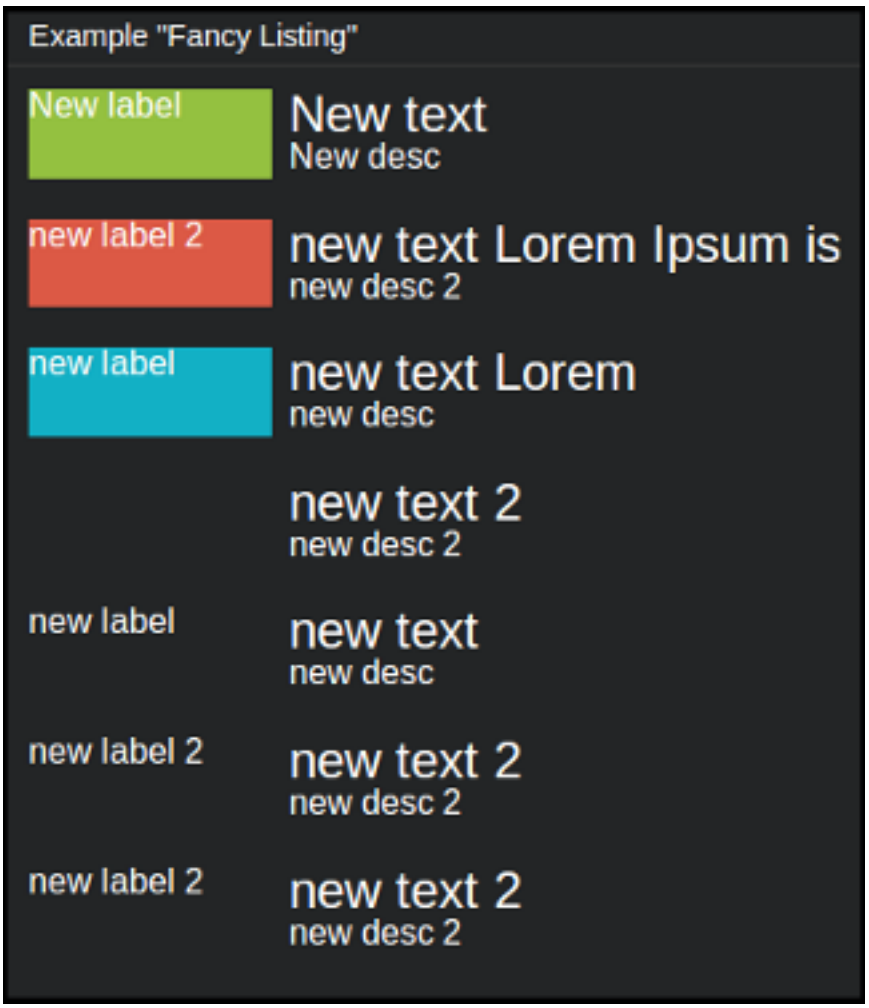
Configuration

This tile does not offer any configuration options.

Note: In case of displaying more than one charts on the same tile, the number of values in `series_list` for every chart should be the same (and they should be equal to the number of `ticks`.

---

fancy_listing



Description

This tile is a more sophisticated version of `listing` tile offering colored labels and centering options. Each entry is an object containing three keys: `label`, `text` and `description`. Therefore, `data` (i.e. content) is just a list of such objects.

Content

```
"data" = [
    {"label": "<label1>", "text": "<entry1>", "description": "<desc1>" },
    {"label": "<label2>", "text": "<entry2>", "description": "<desc2>" }
]
```

where:

`label`
: Smaller label displayed on the left which can be colored.

`text`
: A textual entry to be displayed next to the label.

`description`
: Subtitle displayed below `text` element.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=fancy_listing"
     -d "key=<tile_id>"
     -d 'data=[{"label": "My label 1", "text": "Lorem ipsum", "description": "such description" },
               {"label": "My label 2", "text": "Dolor sit", "description": "yet another" },
               {"label": "My label 3", "text": "Amet", "description": "" }]'
```

Configuration

```
value = {
    "vertical_center": <BOOLEAN>,
    "<position>": {
        "label_color": "<color>",
        "center": <BOOLEAN>
    },
}
```

where:

vertical_center
: Centers vertically all the entries (along with their labels).

  New in version 1.3.0.

position
: Tells which entry (starting from 1) should be a subject to `label_color` and `center` (specified as subkeys of `position`).

  label_color
  : Sets the color of label for the entry given with `position`. Color can be specified in a hexadecimal form (`#RRGGBB`) or by name (e.g. `green`).

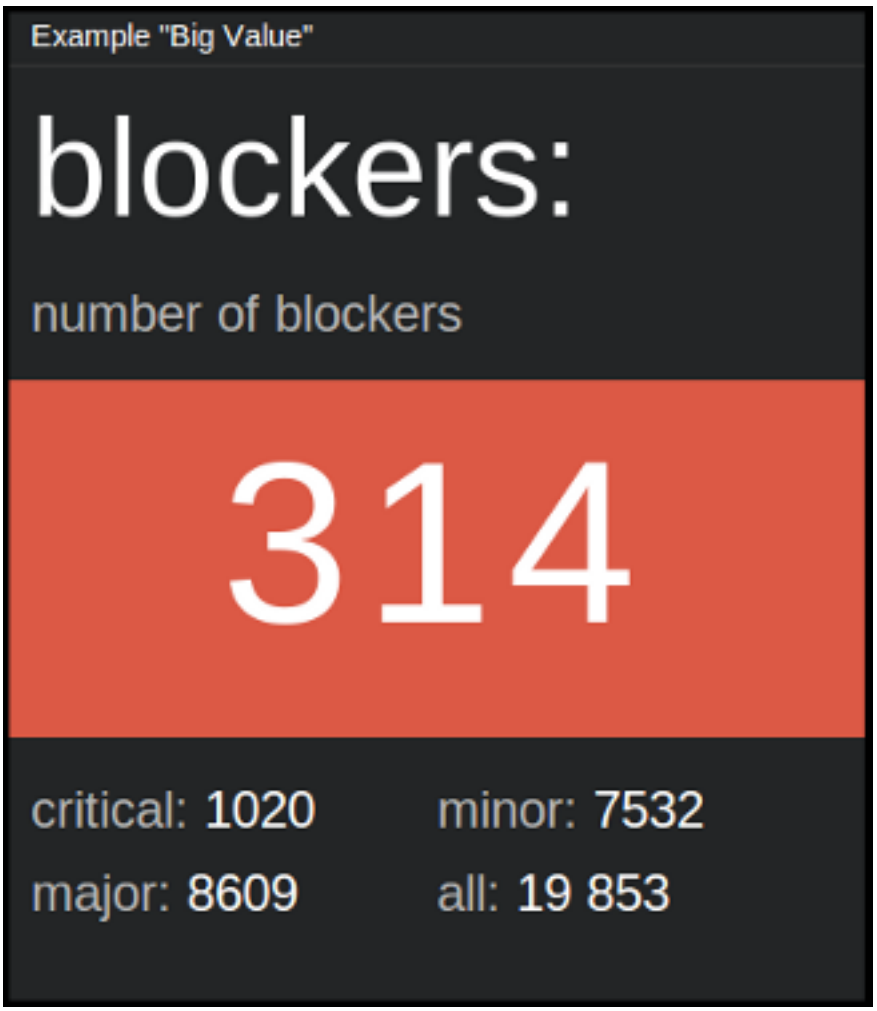
  center
  : Centers horizontally entry's `text` and `description` (it does not affect label's position).

    New in version 1.3.0.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/<tile_id>
     -X POST
     -d 'value={"vertical_center": true,
                "1": {"label_color": "red", "center": true},
                "3": {"label_color": "green", "center": true }}'
```

big_value



Description

This is a variation of `simple_percentage` tile. It has slightly different footer (four possible values instead of just two; values are displayed on the right side of the label instead of below), and the main value (`big_value`) is little bit bigger.

Content

```
data = {
    "title": "<title>",
    "description": "<description>",
    "big-value": "<value>",
    "upper-left-label": "<label>",
    "upper-left-value": "<value>",
    "lower-left-label": "<label>",
    "lower-left-value": "<value>",
    "upper-right-label": "<label>",
    "upper-right-value": "<value>",
    "lower-right-label": "<label>",
    "lower-right-value": "<value>"
}
```

where:

title, description
Title and description (subtitle) for the tile.

big_value
Main value, which treated as a string, so it can contain symbols like `%` etc.

upper-left-value, lower-left-value, upper-right-value, lower-right-value
Smaller, bottom-left and bottom-right values.

upper-left-label, lower-left-label, upper-right-label, lower-right-label
Labels for above values.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=big_value"
     -d "key=<tile_id>"
     -d 'data={"title": "Tickets",
               "description": "number of blockers",
               "big-value": "314",
               "upper-left-label": "critical:",
               "upper-left-value": "1020",
               "lower-left-label": "major:",
               "lower-left-value": "8609",
               "upper-right-label": "minor:",
               "upper-right-value": "7532",
               "lower-right-label": "all:",
               "lower-right-value": "19 853"}'
```

Configuration

```
value = {
    "big_value_color": "<color>",
    "fading_background": <BOOLEAN>
}
```

where:

big_value_color
Background color for `big_value` in a hexadecimal form or color name (e.g. `#94C140` or `green`).

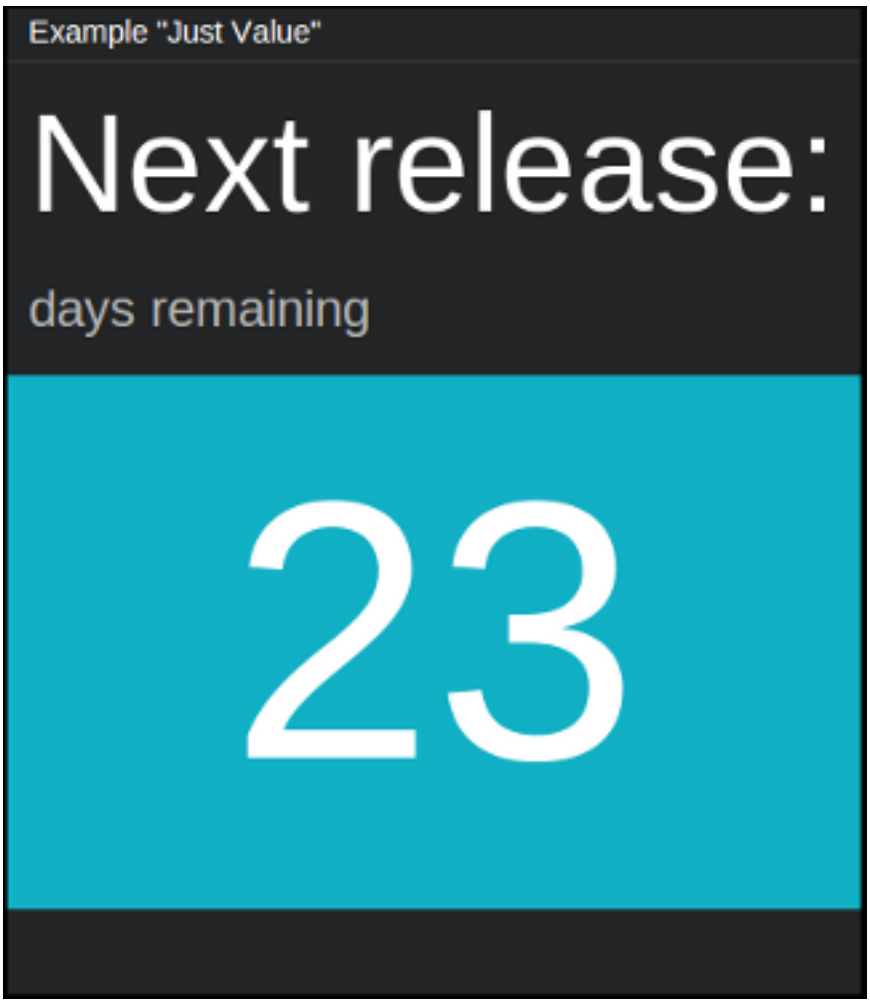
fading_background
Turns on/off background pulsation for `big_value` (may be useful for alerts etc.).

New in version 1.3.0.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/<tile_id>
     -X POST
     -d 'value={"big_value_color": "green", "fading_background": true}'
```

just_value



Description

Tile for displaying single, short information with a possibility to change its background color.

Content

```
"data" = {
    "title": "<title>",
    "description": "<description>",
    "just-value": "<value>"
}
```

where:

title, description
: Title and description (subtitle) for the tile.

just-value
: Value to be displayed on the tile, with optionally colored background.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=just_value"
     -d "key=<tile_id>"
     -d 'data={"title": "Next release:", "description": "(days remaining)", "just-value": "23"}'
```

Configuration

```
value = {
    "just-value-color": "<color>",
    "fading_background": <BOOLEAN>
}
```

where:

`just-value-color`
: Background color for `just-value` in a hexadecimal form or color name (e.g. `#94C140` or `green`).

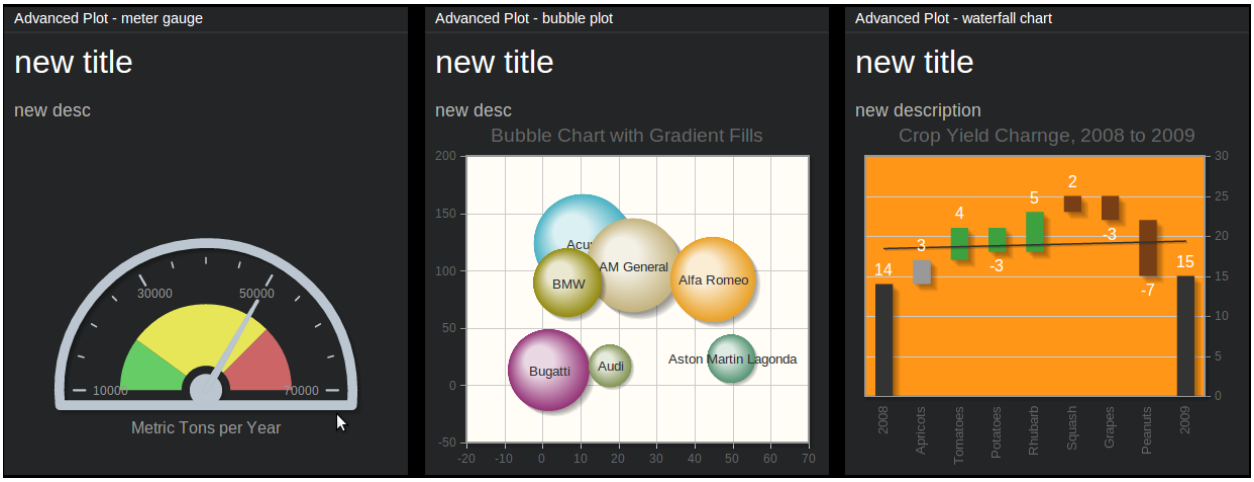
`fading_background`
: Turns on/off background pulsation for `just-value` (may be useful for alerts etc.).

  New in version 1.3.0.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/<tile_id>
     -X POST
     -d 'value={"just-value-color": "green", "fading_background": true}'
```

advanced_plot



Description

This tile is for more demanding users. It basically allows to display arbitrary type of chart/plot from the jqPlot library, along with the title and description (both are optional).

Before you start experimenting with jqPlot library, we suggest to familiarize yourself with this manual. After that you should check out options tutorial and options summary.

Here you will find some examples.

Content

```
"data" = {
    "title": "<tile>",
    "description": "<description>",
    "plotData": "<data>"
}
```

where:

title, description
: Title and description (subtitle) for the tile.

plotData
: Data that will be fed directly to your plot. Its form depends on the specific type of plot that you are going to use - see jqPlot's documentation for the details.

Example (using horizontal Bar Chart - third example from the top):

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=advanced_plot"
     -d "key=<tile_id>"
     -d 'data={"title": "Metric Tons per Year", "description": "",
               "plotData": [[[2,1], [4,2], [6,3], [3,4]],
                            [[5,1], [1,2], [3,3], [4,4]],
                            [[4,1], [7,2], [1,3], [2,4]]]}'
```

Note: Keep in mind that `advanced_plot` can display arbitrary charts from jqPlot library, and more than often they are quite different when it comes to the parameters required etc.

Configuration

```
value = {
  "value": "<jqplot_config>"
}
```

where:

value
: Raw configuration that will be passed directly to jqPlot and which should obey the rules defined by the jqPlot library. Internally, this config will be passed as `$.jqplot(some-container, some-data, our-config)`.

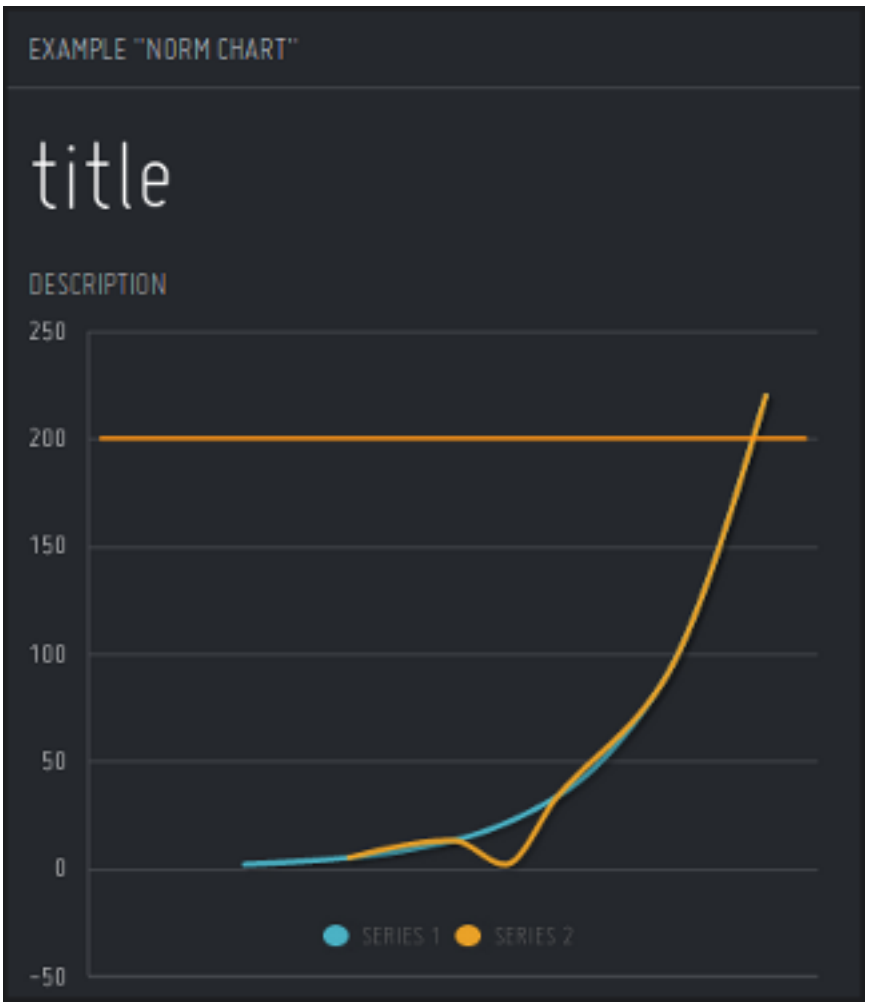
  If such configuration contains one of jqPlot's renderers, its name should be passed as a string, according to the table below:

jqPlot's renderer	string to send
$.jqplot.BarRenderer	"BarRenderer"
$.jqplot.BlockRenderer	"BlockRenderer"
$.jqplot.BubbleRenderer	"BubbleRenderer"
$.jqplot.CanvasAxisLabelRenderer	"CanvasAxisLabelRenderer"
$.jqplot.CanvasAxisTickRenderer	"CanvasAxisTickRenderer"
$.jqplot.CanvasTextRenderer	"CanvasTextRenderer"
$.jqplot.CategoryAxisRenderer	"CategoryAxisRenderer"
$.jqplot.DateAxisRenderer	"DateAxisRenderer"
$.jqplot.DonutRenderer	"DonutRenderer"
$.jqplot.EnhancedLegendRenderer	"EnhancedLegendRenderer"
$.jqplot.FunnelRenderer	"FunnelRenderer"
$.jqplot.LogAxisRenderer	"LogAxisRenderer"
$.jqplot.MekkoAxisRenderer	"MekkoAxisRenderer"
$.jqplot.MekkoRenderer	"MekkoRenderer"
$.jqplot.MeterGaugeRenderer	"MeterGaugeRenderer"
$.jqplot.OhlcRenderer	"OhlcRenderer"
$.jqplot.PieRenderer	"PieRenderer"
$.jqplot.PyramidAxisRenderer	"PyramidAxisRenderer"
$.jqplot.PyramidGridRenderer	"PyramidGridRenderer"
$.jqplot.PyramidRenderer	"PyramidRenderer"

Example (using horizontal Bar Chart - third example from the top):

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/<tile_id>
     -X POST
     -d 'value={
            "seriesDefaults": {
                "trendline": {"show": false},
                "renderer":"BarRenderer",
                "pointLabels": {"show": true, "location": "e", "edgeTolerance": -15},
                "shadowAngle": 135,
                "rendererOptions": {"barDirection": "horizontal"}
            },
            "axes": {"yaxis": { "renderer": "CategoryAxisRenderer"}}}'
```

norm_chart



New in version 1.3.0.

Description

"Curve vs norm" style chart. Suitable for situations, when you want to compare some data with expected value ("norm") or put an emphasis on y-axis values.

Content

```
"data" = {
    "title": "<title>",
    "description": "<description>",
    "plot_data": [ [<series1>], [<series2>], [<series3>], ... ]
}
```

where:

title, description
: Title and description (subtitle) for the tile.

plot_data
: Data for charts in a form of list of series, where each series designates single chart; each element of a given series is a pair `[x_axis_value, y_axis_value]`.

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/push
     -X POST
     -d "tile=norm_chart"
     -d "key=<tile_id>"
     -d 'data={"title": "My title",
               "description": "Some description",
               "plot_data": [[[1, 2], [3, 5.12], [5, 13.1], [7, 33.6], [9, 85.9], [11, 219.9]],
                             [[6, 2], [3, 5.12], [5, 13.1], [7, 33.6], [9, 85.9], [11, 219.9]]]}'
```

Configuration

```
value = {
    "easyNorms": [["<color>", <y-value>, <line_width>], ...]
}
```

where:

easyNorms
: List of norms to be displayed. Each norm consists of three elements:

  color
  : Color which given norm should use - in a hexadecimal form or color name (e.g. `#94C140` or `green`).

  y-value
  : Value for the norm.

  line_width
  : Line thickness for the norm (in pixels).

Example:

```
curl http://localhost:7272/api/v0.1/<api_key>/tileconfig/<tile_id>
     -X POST
     -d 'value={"easyNorms": [["yellow", 200, 2], ["green", 100, 2]]}'
```

Note: In order to keep brevity, all examples presented in specifications above do not include any escape characters. Therefore, it's up to you to insert them where necessary.

And also, remember to set all the elements in angle brackets (e.g. `<api_key>`, `<tile_id>` etc.) to reflect your configuration.

CHAPTER 5

API

One of the advantages of Tipboard is flexibility in feeding tiles with data. We achieve that by providing a simple, REST API - that way, your feeding scripts may be written in any language (Python, Ruby, Bash, Perl, PHP - you name it). The only limitation is the format of input data accepted by a given tile type (see *Library of available tiles* for the details).

To experiment with resources specified below you can use tools like Advanced REST Client (Chrome extension), or cURL, if you prefer working from command line. For Python programmers, there's an excellent Requests library, which we strongly recommend.

API key

To send anything to your tiles, first you have to get your API key. This unique key is generated for you automatically during Tipboard's installation and may be read in the `~/.tipboard/settings-local.py` file - it is a sequence of characters starting with `API_KEY`, e.g.:

```
API_KEY = 'e2c3275d0e1a4bc0da360dd225d74a43'
```

If you can't see any such string, just add the key manually (it doesn't have to be as long and hard to memorise as the one above, though).

Note: Every change in `settings-local.py` file requires restart of the application.

Available resources

Current API version: **v0.1**

Note: In 99% of cases, probably only `push` and `tileconfig` will be of interest to you (and maybe `tiledata` too).

POST /api/(*api_version*)**/**
*api_key***/push** Feeds tiles with data. Input data should be provided in the format that complies with the one used in a desired tile. **Note:** a tile to which data will be sent is defined by the key included in the data sent rather than by *tile_id* as in cases below.

Parameters

- **api_version** – version of API to be used

- **api_key** – your API key

Example request:

```
POST /api/v0.1/my_key/push
Host: localhost:7272
POST data: tile=text key=id_1 data={"text": "Hello world!"}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

Tile's data pushed successfully.
```

POST /api/(*api_version*)**/**
*api_key***/tileconfig/***tile_id* Configures tile specified by *tile_id*. The configuration should comply with the specification of a given tile type.

Parameters

- **api_version** – version of API to be used
- **api_key** – your API key
- **tile_id** – unique tile's ID from your `layout_config.yaml` file

Example request:

```
GET /api/v0.1/my_key/tileconfig/id_1
Host: localhost:7272
POST data: value={"font_color": "#00FF00"}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

Tile's config updated.
```

DELETE /api/(*api_version*)**/**
*api_key***/tileconfig/***tile_id* Resets configuration of the tile specified by *tile_id*.

Parameters

- **api_version** – version of API to be used
- **api_key** – your API key
- **tile_id** – unique tile's ID from your `layout_config.yaml` file

Example request:

```
DELETE /api/v0.1/my_key/tileconfig/id_1
Host: localhost:7272
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

Tile's config deleted.
```

GET /api/(*api_version*)**/**
*api_key***/tiledata/***tile_id* Retrieves data belonging to the tile specified by *tile_id*. May be useful in cases when you need to re-fetch some parts of your data (e.g. when updating your team's stats) or just for diagnostics.

Parameters

- **api_version** – version of API to be used
- **api_key** – your API key
- **tile_id** – unique tile's ID from your `layout_config.yaml` file

Example request:

```
GET /api/v0.1/my_key/tiledata/id_1
Host: localhost:7272
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
    "tile_template": "text",
    "meta": {
        "font_color": "#ff9618",
        "font_size": "45px"
    },
    "data": {
        "text": "Lorem ipsum."
    },
    "id": "id_1"
}
```

DELETE /api/(*api_version*)**/**
*api_key***/tiledata/***tile_id* Removes everything belonging to the tile given by *tile_id* from Redis.

Parameters

- **api_version** – version of API to be used
- **api_key** – your API key
- **tile_id** – unique tile's ID from your `layout_config.yaml` file

Example request:

```
DELETE /api/v0.1/my_key/tiledata/id_1
Host: localhost:7272
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

Tile's data deleted.
```

GET /api/(*api_version*)**/**
*api_key***/info** Provides information on project and user configuration. This resource has been created for debugging purposes.

Parameters

- **api_version** – version of API to be used

- **api_key** – your API key

Example request:

```
GET /api/v0.1/my_key/info
Host: localhost:7272
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
    "tipboard_version": "1.3.0",
    "project_layout_config": "/home/pylabs/.tipboard/layout_config.yaml",
    "redis_db": {
        "host": "localhost",
        "db": 4,
        "port": 6379
    },
    "project_name": "pylabs"
}
```

CHAPTER 6

Extras

Here you will find description of components which are not a part of the Tipboard project *per se*, although they may be useful to some of its users. Assuming standard installation, you can find them here:

```
<path_to_your_virtualenv>/lib/python2.7/site-packages/tipboard/extras
```

Note: If you have developed something of similar nature and you are willing to share it, we encourage you to make a pull request to our repo. Thanks!

jira-ds.py

Script for fetching frequently used data from JIRA issue tracker, in order to present it on your dashboards. Returns requested data to stdout in JSON format. For the list of available options see `jira-ds.py --help`.

This script is basically a wrapper around `jira-ds.js`, so those two files shouldn't be separated. Requires CasperJS and PhantomJS installed somewhere in your path (we suggest using npm for that).

Before you start using them, remember to fill in `JIRA_CREDENTIALS` and `JIRA_BASE_URL` (in `jira-ds.py`) as well as `url_jira` and `url_jira_login` (in `jira-ds.js`) with the your JIRA credentials, location of your JIRA instance and its login page.

Tested with JIRA 6.1.x.

client_code_example.py

Simple Python script targeted to novice users serving as an example how to glue together three steps: fetching data, processing it and then sending it to the tile. See comments in the source code for further explanation.

fabfile.py

Script for quick, automated installations on remote machines.

You need to have fabric and fabtools to use remote install script.

Run:

```
fab -H root@host install
```

– it will install all needed `.deb` packages, create virualenv and set up Tipboard service using master branch from our main repo.

CHAPTER 7

Change Log

1.4.1

Released on November 16, 2016.

- Fixes for tiles: ‘advanced_plot’, ‘simple_percentage’ and ‘text-tile’.
- Fix for tile keys cache.
- Added support for RequireJS.
- Make use of simplify.js to make charts (in e.g. ‘line_chart’ tile) more readable.
- Fix for ‘Target dimension not set’ error when layout ‘row_1_of_1’ is used.
- Minor fixes, improvements, cleanups etc.

1.4.0

Released on August 28, 2014.

- Tipboard got open-sourced!

1.3.1

Released on July 23, 2014.

- Added extensive documentation.
- Numerous fixes in ‘jira-ds’ script (e.g added timeouts).
- Fixed definitions of colors available for tiles.
- Fixed checking for expired data (+ made it timezone aware).
- Added integration with Travis.
- Changed default size of the log files.

1.3.0

Released on February 17, 2014.

New features:

- Fading highlighter (for just_value, big_value and simple_percentage tiles).
- Fancy centering options for fancy_listing tile.
- Notifications on data expiration.
- New tile: norm_chart.
- Possibility to define more than one dashboard per application instance.

Bug fixes:

- Tiles no longer vanish when flipping is enabled.
- Characters like '.' or '-' (and some others) in tiles' ids are no longer causing problems.
- Renderer names (like OHLCRenderer, MarkerRenderer, ShadowRenderer and ShapeRenderer) can now safely be passed to tiles' configs.

Others:

- Error messages displayed on tiles got more emphasis.
- Renderer names (in tiles' configs) are now case insensitive.
- Added frontend tests and selector for tests.

1.2.0

Released on December 19, 2013.

This release brings new features and some minor bugfixes.

- New tiles: big_value, just_value, advanced_plot.
- Rewritten 'jira-ds' script with some new options (e.g. 'maxResults' for JQL).
- Completely new graphic theme - with new colors, fonts etc.
- Fixed existing tests and some new added.
- Exceptions raised by JavaScript are now displayed on the tiles.
- Improved config handling for bar_chart, pie_chart and line_chart.
- Added possibility to specify specialized renderers for almost all plots (except cumulative_flow).

1.1.0

Released on November 20, 2013.

This release contains multiple improvements and bugfixes:

- Tiles are no longer packages (i.e. folders).
- Reorganized files/folders structure.

- Massively reduced app's settings.
- Simplified layout config (no more classes, only one keyword needed to get tile flips working).
- New tiles: bar_chart, fancy_listing.
- Improved scaling of tiles + some cosmetic changes.
- Unique API key is generated automatically for every project.
- Fabric script for administrative installs

1.0.0

Released on November 06, 2013.

This is the first release of Tipboard.

- initial release

CHAPTER 8

License

/api

GET /api/(api_version)/(api_key)/info, 37
GET /api/(api_version)/(api_key)/tiledata/(tile_id), 36
POST /api/(api_version)/(api_key)/push, 35
POST /api/(api_version)/(api_key)/tileconfig/(tile_id), 36
DELETE /api/(api_version)/(api_key)/tileconfig/(tile_id), 36
DELETE /api/(api_version)/(api_key)/tiledata/(tile_id), 37