
Dashticz Documentation

Release beta

Rob Geerts

Mar 24, 2022

Contents:

1	Introduction	3
1.1	Why Dashticz	3
1.2	Concept	3
2	Getting started	5
2.1	Installation	5
2.2	Configuring a basic Dashticz Dashboard	8
3	Configuration	13
3.1	Dashticz configuration	13
3.2	Blocks	21
3.3	Columns	174
3.4	Screens	174
4	Customizations	179
4.1	Styling via custom.css	179
4.2	Functionality via custom.js	190
5	Tips and Tricks	197
5.1	Dashticz security	197
5.2	Dashticz security (method 2)	199
5.3	Use of Web Fonts	200
5.4	Changing alert icon colors dynamically	200
6	Release Notes	203
6.1	Upgrade instructions	203
6.2	Release Notes	208
7	Troubleshooting	231
7.1	Connection	231
7.2	Docker	232
7.3	Domoticz blocks	233
8	For Developers	235
8.1	Code	235
8.2	Design	239
8.3	Translations	241

8.4 Documentation	242
9 Indices and tables	245

This documentation describes how to install and configure Dashticz.

For Dashticz's **beta** version documentation go to: <https://dashticz.readthedocs.io/en/beta/>

For Dashticz's **master** version documentation go to: <https://dashticz.readthedocs.io/en/master/>



1.1 Why Dashticz

The Dashboard of Domoticz is quite powerful. The disadvantage is that it's only possible to show information known in Domoticz. There is where Dashticz steps in. Dashticz is able to show (almost) all Domoticz information. In addition to that it's possible to show information from all kind of other sources.

1.2 Concept

The Dashticz dashboard may consist of several screens.

Every screen consist of one or more columns.

In every column you place one or more blocks.

A block may be a Domoticz device, external content being displayed in a frame, or some special block.

The next sections show how to setup your system, how to configure Dashticz, how to configure a block, how to add blocks to a column, and how to define screens.

2.1 Installation

2.1.1 Automatic install

Note: The installation scripts currently only works on Raspberry (except xbian) and Ubuntu

For the automatic install open a terminal in a folder of choice where Dashticz V3 will get installed:

```
mkdir dev
cd dev
```

Then start the installation script with:

```
. <(wget -qO - https://raw.githubusercontent.com/Dashticz/dashticz/beta/scripts/
↳dashticz_install.sh )
```

The script:

- asks for a folder where to install Dashticz V3
- asks to install the beta or master branch : For now only beta works with the auto install!!
- Clones the Dashticz V3 repository and selected branch into a new folder
- Asks for the IP adress of your Domoticz server.
- Copies CONFIG_DEFAULT.js to CONFIG.js with the correct IP address for Domoticz

Then a Makefile is executed which:

- Installs Docker (if not installed yet)
- Creates a Dashticz V3 container, named dtv3, containing Apache and PHP
- FInd the first free port, 8082 or higher

- Starts the container on the first free port
- Mounts the dashticz folder to the web-root of the container
- Shows the Dashticz url

If you open this url then the default Dashticz dashboard becomes visible.

So no more need to configure Apache and/or PHP! It just works out-of-the box. You need a few 100 MB free space on your system.

The first time the installation may take a while (5 - 15 minutes?): be patient.

Update from a previous version

If the default page is working then you can copy your previous CONFIG.js, custom.css, custom.js from your previous installation to dashticz/custom.

Just refresh your browser, and your new dashboard is shown. No need to rebuild the docker container.

2.1.2 Manual Setup

Manual setup consists of two steps:

- Preparing your system
- Installing Dashticz

System preparation

Since Beta 2.4.6 (October 2018) the installation instruction changed. Main reason is that for most functionality PHP support in the web server is needed. The Domoticz web server doesn't support PHP. That means that Dashticz needs to be installed under a different web server with PHP enabled. The installation instruction consists in two steps:

Installation of a web server

This example shows installation of Apache on Raspberry in it's most basic configuration: running at port 80. Besides Apache also PHP needs to be installed, since it's used by the Calendar and Garbage module in Dashticz.

```
sudo apt-get update
sudo apt-get install apache2 php php-xml php-curl libapache2-mod-php
sudo systemctl restart apache2
```

Note: On some Raspberry configurations the installation of php is failing. In that case you can try the alternative installs under Advanced Installation below

Now check whether Apache is running by browsing to `http://<YOUR_IP>` You should see the Apache demo page.

Alternative installations

For debian/stretch:

- <https://tecadmin.net/install-php-debian-9-stretch/> (PHP installation instructions)].

Functionality that is lost without PHP:

- calendar
- garbage
- savings settings from Dashticz
- version check
- CORS proxy, which is used for TVguide, news, traffic info, frames, images in buttons (You can switch an external CORS proxy as well.)

Quick install for Synology NAS

- Install Apache HTTP Server (Web Station will be installed too) and PHP
- In Web Station configure your HTTP back-end server and PHP (PHP Extensions `curl` and `openssl` needs to be selected)
- Download Dashticz ZIP file from GitHub <https://github.com/Dashticz/dashticz> (choose branch)
- Create Dashticz folder on your Web Server
- Unpack downloaded Dashticz ZIP file to your Dashticz folder
- Copy `CONFIG_DEFAULT.js` to `CONFIG.js`
- Edit `CONFIG.js` to your needs and set write permission

Dashticz installed on ISS

- You have to register a MIME type for tpl extension `mime-type type="text/x-phpbb-template`
- See also <https://bobcares.com/blog/http-error-404-3-not-found/> and <https://forums.iis.net/t/1095097.aspx>

Dashticz Installation

Example for Raspberry PI running Apache web server: Assumption:

- Apache is running at `http://192.168.1.3` on the default port 80 (but this can be any IP:port address)
- Domoticz is running at `http://192.168.1.3:8084`

First clone the dashticz repository to a folder of your choice:

```
cd /home/pi
git clone https://github.com/Dashticz/dashticz
```

If you prefer the development branch (might be less stable, but latest and greatest), then use the following git clone command:

```
git clone https://github.com/Dashticz/dashticz --branch beta
```

After the installation is finished, go to the `/home/pi/Dashticz/custom/` folder, copy the `CONFIG_DEFAULT.js` file to `CONFIG.js` (mind the CAPITALS!), and edit it with the basics:

```
cd dashticz/custom/
cp CONFIG_DEFAULT.js CONFIG.js
nano CONFIG.js
```

Example of `CONFIG.js`:

```
var config = {}
config['language'] = 'nl_NL'; //or: en_US, de_DE, fr_FR, hu_HU, it_IT, pt_PT, sv_SV
config['domoticz_ip'] = 'http://192.168.1.3:8084';
config['domoticz_refresh'] = '5';
config['dashticz_refresh'] = '60';
```

You can read more about the connection configuration *Connection*.

Then create a symbolic link from the root of the www folder of your web server to the previously created Dashticz location:

```
sudo ln -s /home/pi/dashticz/ /var/www/html
```

Set the correct permissions to the files and folders:

```
chmod -R a+rX /home/pi/dashticz
```

If you want to be able to save the settings via Dashticz to CONFIG.js then you have to give write permission to CONFIG.js for root:

```
chmod a+w /home/pi/dashticz/custom/CONFIG.js
```

Now you can browse to the dashboard: <http://192.168.1.3/dashticz/index.html> Replace 192.168.1.3 with the IP Address (and Port number) for your web server, NOT your Domoticz IP!

By default, Dashticz will show all your Domoticz favorites on the dashboard.

Updating

Option 1 - From terminal you can add the following command in Dashticz folder:

```
git pull
```

Option 2 - Download zip file from GitHub, copy and extract in your Dashticz folder. Create backup of your custom folder(s) first.

Troubleshooting

After updating my Ubuntu version I had to manually enable php7.3 on Apache:

```
sudo a2enmod php7.3
sudo systemctl restart apache2
```

Dashticz can be installed in two ways:

- Automatic install via the installation script
- Manual install

2.2 Configuring a basic Dashticz Dashboard

If you followed the steps as described in *Installation* then you are prepared for creating your own Dashboard.

2.2.1 Step 1: Default Dashboard

Let's start with a minimal config. Create the config file `custom/CONFIG.js` with the following content:

```
var config = {}
config['language'] = 'nl_NL'; //or: en_US, de_DE, fr_FR, hu_HU, it_IT, pt_PT, sv_SV
config['domoticz_ip'] = 'http://192.168.1.3:8084';
config['domoticz_refresh'] = '5';
config['dashticz_refresh'] = '60';
```

Replace the `domoticz_ip` config setting with the ip-address of your Domoticz server.

This should give a result like this:



In your situation Dashticz might show less devices, depending of course on how many devices you have defined in Domoticz. Further, with the default settings Dashticz only shows the Domoticz devices that are being used in Domoticz and marked as favorite.

If you don't see any device then probably your Domoticz connection is not working.

Troubleshooting

- Check your Domoticz IP address and port number
- If you've configured a Domoticz username/password then add the following settings to `CONFIG.js`

```
config['user_name'] = '<Domoticz Username>';
config['pass_word'] = '<Domoticz Password>';
```

If your initial dashboard is working then let's continue with step 2.

2.2.2 Step 2: Creating a custom layout

In this example we'll create a simple dashboard, consisting of

- one screen.
- two columns, consisting of 1/3 and 2/3 of the screen width.
- two devices in the second column next to each other.

First we have to add the block definition for the devices. For this you have to know the Domoticz device ID. You can find your device ID in the Settings->Devices overview of Domoticz. Choose two switch devices. In this example we'll use device id 120 and 121.

Add the following to `CONFIG.js`:

```
//Definition of blocks
blocks = {}
blocks[120] = {
  width: 6
}

blocks[121] = {
  width: 6
}
```

This means we want to use device id 120 and device id 121. We give them width of 6.

Explanation of width For setting the width of blocks (and columns) the grid system is used. The total available width is 12. If you set the width of a block to 6 then 50% of the column width will be used for the block width.

So in the previous example both blocks will have a width of 50% of the column we will place them in, meaning they will fit into one row.

Next, we add the column definitions to `CONFIG.js`:

```
//Definition of columns
columns = {}
columns[1] = {
  //In this example: No blocks are defined in this column
  //This column will be empty
  width: 4
}
columns[2] = {
  blocks : [120, 121],
  width: 8
}
```

In this example we've defined two columns. The first one will have width 4 (33% screen width. The second one has width 8, meaning 67% of screen width).

In column 2 we place device 120, as defined by `blocks[120]` and device 121, as defined by `blocks[121]`

As the last step the screens need to be defined:

```
//Definition of screens
screens = {}
screens[1] = {
  columns: [1, 2]
}
```

One screen has been defined, consisting of column 1 and column 2.

Just to be sure, add the following `CONFIG` settings as well:

```
config['use_favorites'] = 0; //Request all Domoticz Devices, not only favorites
config['auto_positioning'] = 0; // Use 0 this if you have defined your own columns
```

Your complete `CONFIG.js` now should look as follows:

```

var config = {}
config['language'] = 'nl_NL'; //or: en_US, de_DE, fr_FR, hu_HU, it_IT, pt_PT, sv_SV
config['domoticz_ip'] = 'http://192.168.178.18:8080';
config['domoticz_refresh'] = '5';
config['dashticz_refresh'] = '60';

config['use_favorites'] = 0; //Request all Domoticz Devices, not only favorites
config['auto_positioning'] = 0; // Use 0 this if you have defined your own columns

//Definition of blocks
blocks = {}
blocks[120] = {
  width: 6
}

blocks[121] = {
  width: 6
}

//Definition of columns
columns = {}
columns[1] = {
  //In this example: No blocks are defined in this column
  //This column will be empty
  width: 4
}
columns[2] = {
  blocks : [120, 121],
  width: 8
}

//Definition of screens
screens = {}
screens[1] = {
  columns: [1, 2]
}

```

This should give the following result:



2.2.3 Retrieve status of a device

You can get the status of a specific device with: `http://192.168.1.3:8084/json.htm?type=devices&rid=IDX`

- Replace 192.168.1.3 with the IP Address (and Port number) for your Domoticz!
- IDX = id of your device (This number can be found in the Domoticz' devices tab in the column "IDX")

First you have to install Dashticz, then you can configure your Dashboard.

The Dashticz configuration is stored in the file `[dashticz folder]/custom/CONFIG.js` and consists of several parts.

The first part is used to configure all kind of global Dashticz settings, including the Domoticz connection, language settings, etc. See *Dashticz configuration*.

The second part is used to define all blocks.

In the third part the columns are defined, followed by the screens.

These four parts will be discussed in the following sections.

3.1 Dashticz configuration

Dashticz can be configured by editing the `CONFIG.js` file. This file you can find in the subfolder `[dashticz]/custom`.

Note: TIP! If CUSTOM POSITIONING is not working check if you have uncomment all lines from the blocks/columns/screens you want.

In the following part, the `CONFIG.js` is divided in sections. For each section there will be an explanation how to use.

First part describes setting up the configuration of the Domoticz connection. After that an overview of all configuration parameters can be found.

3.1.1 Connection

Below the basic configuration to make the connection with Domoticz work.

```
var config = {}
config['language'] = 'nl_NL'; //or: en_US, de_DE, fr_FR, hu_HU, it_IT, pt_PT, sv_SE
config['domoticz_ip'] = 'http://192.168.1.3:8084';
config['domoticz_refresh'] = '5';
config['dashticz_refresh'] = '60';
```

Parameter	Description
config['language']	can be used to select the language, Dutch (nl_NL), English (en_US), German (de_DE),French (fr_FR), Hungarian (hu_HU), Italian (it_IT), Portuguese (pt_PT), or Swedish (sv_SV)
config['domoticz_ip']	is the URL to your Domoticz installation (with the correct PORT address)
config['domoticz_refresh']	the refresh rate of Dashticz to get information from Domoticz
config['dashticz_refresh']	the refresh rate of the Dashticz Dashboard

3.1.2 Config parameters

Parameter	Description
domoticz_ip	IP Address and Portnumber of your Domoticz installation 'http://192.168.1.3:8084'
user_name	Domoticz username ' ' = No username (default) 'john' = Use 'john' as Domoticz username
pass_word	Domoticz password ' ' = No password (default) 'secret' = Use 'secret' as Domoticz password
loginEnabled	Enable if you want a login form to dashticz false = No login form (default) true = Show login form
login_timeout	Time to keep Dashticz logged in 60 = Time in minutes
enable_websocket	Enable Domoticz websocket connection. See <i>Websocket connection</i> false = Dashticz will use a http connection. true = Dashticz will switch to a websocket connection if the Domoticz version is above 4.11000.

Continued on next page

Table 1 – continued from previous page

Parameter	Description
domoticz_timeout	Time Dashticz is fetching for Domoticz devices during the initial request. After this time Dashticz falls back from websocket to HTTP. 2000 = Time in <value> milliseconds (default=2000).
app_title	Name of the Dashboard - Title to show in the <i>Topbar</i> 'Dashticz' = Show 'Dashticz' in the top bar
domoticz_refresh	Number of seconds to get the information from Domoticz 5 = Refresh of Domoticz data every 5 seconds
dashticz_refresh	Number of minutes to refresh the Dashticz dashboard 60 = Refresh of the Dashticz dashboard every 60 minutes
default_news_url	URL of the default news feed 'http://www.nu.nl/rss/algemeen' = Example for nu.nl
news_scroll_after	Enter the ammount in seconds (delay) 5 = Scroll the news message every 5 seconds
standby_after	Enter the amount of minutes 0 = No standby mode(default) 1 . . 1000 = Switch to standby after <value> minutes
start_page	Page to show after starting Dashticz 1 . . 100 = Page number
vertical_scroll	Enable vertical scroll in case Dashticz dashboard height is more than the screen height 0 = Disable vertical scroll 1 = Enable vertical scroll if swiper is disabled 2 = Enable vertical scroll (=default)
enable_swiper	Enable horizontal swiping in case multiple screens have been defined. 0 = Swiper disabled 1 = Swiper enabled if screen width > 760px 2 = Swiper enabled (=default)

Continued on next page

Table 1 – continued from previous page

Parameter	Description
swiper_touch_move	Swipe the screen by touch. Only applicable in case swiper is enabled. 0 = Swipe by touch disabled 1 = Swipe by touch enabled
auto_swipe_back_to	when no activity, swipe back to the selected page. Also see <i>Auto swipe, auto slide</i> 1..100 = page number (default: 1)
auto_swipe_back_after	The amount of seconds after which Dashticz will start with auto swipe/auto slide. Also see <i>Auto swipe, auto slide</i> 0 = No auto swiping (default) 1..9999 = Start auto swipe back after <value> seconds the last screen touch/mouse activity.
auto_slide_pages	Loop all pages and change page every x seconds, set config['auto_swipe_back_to'] = 0. Also see <i>Auto swipe, auto slide</i> 0 = Auto slide is disabled. (=default) 1..9999 = Auto slide to the next page every <value> second
slide_effect	Control which Screenslider effect you prefer 'slide', 'fade', 'cube', 'coverflow', 'flip'
standard_graph	Default Graph shown on the Dashticz Dashboard 'HOUR', 'MONTH', 'DAY'
security_panel_lock	If set the Security Panel in Domoticz or Dashticz to 'Arm - Away', then Dashticz will automatically load a full screen panel. 0 = Loading Security Panel full screen disabled (default) 1 = Loading Security Panel full screen enabled
language	Default language of Dashticz. See the lang folder for all supported languages. 'en_us' = default 'nl_NL', 'de_DE', '...'
timeformat	Configure the TimeFormat 'DD-MM-YY HH:mm' = default
calendarformat	Configure the Calendar Date/Time format. 'dd DD.MM HH:mm' = default

Continued on next page

Table 1 – continued from previous page

Parameter	Description
calendarlanguage	Controls the weather dates and garbage pickup dates language '<LANGUAGE>'
calendarurl	'<url>' = Configure your Calendar URL if only 1 Calendar (ICS)
boss_stationclock	Configure your type of clock 'NoBoss', 'BlackBoss', 'RedBoss' = Default, 'ViennaBoss'
gm_api	[API KEY] = API Key to use with the Google Maps functionality
gm_latitude	[LATITUDE] = Enter the Latitude to use within Google Maps
gm_longitude	[LONGITUDE] = Enter the Longitude to use within Google Maps
gm_zoomlevel	Enter the Google Maps zoom level 1 = Whole world 2..14 15 = Most detail
wu_api	' [API KEY] ' Your Wunderground Weather API key. You can get a API key at https://www.wunderground.com/weather/api/d/pricing.html . ‘Edit: You can no longer get a free API key from Wunderground’”
wu_city	' [CITY] ' Put here your weather city.
wu_country	' [COUNTRY] ' Put here your weather country
wu_name	' [CITY] ' Alternative display name of your city
switch_horizon	'<url>' (Only Dutch users) If you have a Ziggo Horizon box, you can set the url of the Horizon box here
host_nzbget	' [IP ADDRESS :PORT NUMBER] ' Configure the IP Address and Portnumber of your NZB Host
spot_clientid	' [CLIENTID] ' Configure your Spotify Client ID (see also <i>Spotify</i>)
selector_instead_of_buttons	Choose how to show your selector switches 0 As buttons 1 As dropdown menu
auto_positioning	Configure the ability to define your own positioning for the buttons (in combination with <code>config['use_favorites']</code>) 0 Use this if you have defined your own columns 1 Default
use_favorites	0 Show all domoticz devices (default) 1 Only show Domoticz devices marked as favorite in Domoticz If use auto positioning, then this item should be 1

Continued on next page

Table 1 – continued from previous page

Parameter	Description
use_cors	0 Don't use CORS proxy for OpenWeatherMap (default) 1 Use CORS proxy for OpenWeatherMap. Needed on Android 4.4.2.
last_update	0 / 1 To show the time when the device was updated for the last time
hide_topbar	0 / 1 Hide or Show <i>Topbar</i>
hide_seconds	0 / 1 Show the seconds of the clock
hide_seconds_stationclock	0 / 1 Configure if you like to show seconds in the StationClock
use_fahrenheit	0 / 1 Select temperature in Celcius (Default) of Fahrenheit
use_beaufort	0 / 1 Use Bft instead of m/s for windspeed
translate_windspeed	0 / 1 For windspeed use north northwest instead of NNW
static_weathericons'	0 / 1 Use Static or 'moving' weather icons
hide_mediaplayer'	0 / 1 When you have a mediaplayer connected, hide it when nothing is playing
selector_instead_of_buttons'	0 / 1 Use buttons for the selector switch in stead of the dropdown menu
settings_icons	["settings", "fullscreen"] Show the given icons if the settings block is selected. Available: "settings", "fullscreen"
shortdate'	'D MMM' Short format for dates, see https://momentjs.com/ for all options.

Continued on next page

Table 1 – continued from previous page

Parameter	Description
longdate	‘D MMMM YYYY’ Long format for dates, see https://momentjs.com/ for all options.
shorttime	HH:mm Short format for time, see https://momentjs.com/ for all options.
longtime	HH:mm:ss Long format for time, see https://momentjs.com/ for all options.
weekday	‘dddd’ Format to show the weekday, see https://momentjs.com/ for all options.
no_rgb	0 / 1 Hide or show RGB button on switch
colorpicker	0: No RGB colorpicker, 1: Old style RGB colorpicker, 2: New style RGB colorpicker Choose the default RGB colorpicker for RGB devices
standby_call_url	[URL] Enter the url for adjusting the brightness when entering stand-by mode
standby_call_url_on_end	[URL] Enter the url for adjusting the brightness when exiting stand-by mode
hide_off_button	0 / 1 Hide off button of selector switch
speak_lang	Text to speech language ' <LANGUAGE> ' Language options: de-DE, en-US, es-ES, fr-FR, it-IT, nl-NL, pl-PL, ru-RU
longfonds_zipcode	Longfonds (Dutch air quality check) zipcode. ' 1234AZ ' Language options
longfonds_housenumber	Longfonds (Dutch air quality check) housenumber ' 123 ' Language options

3.1.3 Dashticz URL parameters

Dashticz parameters can be configured via the Dashticz URL as well. For instance, to start Dashticz with a custom title and set the dashticz refresh period to 300 minutes, use the following URL:

```
http://<dashticz-ip:port>/?app_title=Living room&dashticz_refresh=300
```

Besides the Dashticz configuration parameters, the following additional parameters can be used in the Dashticz URL:

Parameter	Description
cfg	The Dashticz configuration file to be used instead of CONFIG.js. CONFIG.room2.js
cfg2	The name of a second configuration file, which will be loaded after CONFIG.js. CONFIG.extra.js
css	The CSS file to be used instead of custom.css. custom.mobile.css
folder	The folder containing the configuration files. (Default: custom) custom/room1

So, to load Dashticz with a different config file, use the following URL:

```
http://<dashticz ip:port>/?cfg=CONFIG.room2.js
```

The URL parameters can be accessed in CONFIG.js or custom.js via the global variable `_PARAMS`:

```
if (_PARAMS['myownparam']=='2') {  
    screen[2].columns = [2,4,5]  
}
```

3.1.4 Usage

PHP based CORS proxy

To be able to load resources from other domains, like tvguide data and news updates, we need a CORS proxy (Cross Origin Resource Sharing). Public CORS proxies exist on the internet, like `cors-anywhere.herokuapp.com`. These public CORS proxies might be slow or not available at all.

A basic PHP based CORS proxy has been integrated into Dashticz.

For normal use just remove the `config['default_cors_url']` from your `CONFIG.js` and the internal CORS proxy will be used.

If you prefer to use a different CORS proxy you can define it in `CONFIG.js` as usual:

```
config['default_cors_url'] = 'http://cors-anywhere.herokuapp.com'
```

In case PHP is not installed it falls back to the defined CORS proxy in `config['default_cors_url']`. In case `config['default_cors_url']` is not set, it will use `cors-anywhere.herokuapp.com` by default.

Websocket connection

From v4.11000 Domoticz supports a so called websocket connection next to the standard http(s) connection.

Note: Currently the Domoticz stable version is 4.10717 which doesn't support the websocket connection. That means you have to update to the Domoticz develop branch.

Warning: The Domoticz develop branch may give issues, especially on Raspberry PI. Make a backup of Domoticz first! If you switch to the develop branch, the Domoticz database will be upgraded as well, and cannot be downgraded.

The advantages of a websocket connection:

- instant updates in Dashticz in case of a changing Domoticz device
- Faster and less overhead

To prevent switching to a websocket connection add the following setting to `CONFIG.js`:

```
config['enable_websocket'] = false;
```

In the <gif> below you see the instant updates in action. On the right you see the Domoticz dashboard. On the left Dashticz.

3.2 Blocks

The visible elements on the Dashticz dashboard are called blocks. Several block types are supported:

3.2.1 Domoticz blocks

Several types of Domoticz blocks can be defined:

- Devices
- Scenes
- Groups
- Variables
- Texts

Devices

Almost all Domoticz devices can be shown on the Dashticz dashboard. Before you can use a device in a column you must make it known in your CONFIG.js as follows:

```
blocks = {}           //only once
blocks[123] = {
  title: 'My device',
  width: 12
}
```

If you use anything other than a number you have to put it between quotes: ['s1'] ['v3'] ['123_1']

The number 123 is the Domoticz device id. The example above also shows the use of two parameters: `title` and `width`. For a full list of parameters see [Block parameters](#).

For most devices containing a value, like temperature, power, etc, it's possible to show the data as a graph. See [Graphs](#).

You can also use custom names for the block identifier. In that case you have to add the `idx` parameter to indicate which Domoticz device you want to use:

```
blocks['my device'] = {
  idx: 123
}
```

Grouped devices

To use grouped devices in a column you must make it known in your CONFIG.js as follows:

```
blocks['lights'] = {
  blocks: [
    'light_livingroom',
    'light_kitchen',
    'light_bathroom'
  ]
}
```

Now you can add all 3 light blocks to a column with the following code:

```
columns[1] = {}
columns[1]['blocks'] = [
  'lights'
]
```

Scenes and Groups

To select a Domoticz Group or Scene add 's' in front of the Scene/Group ID.

Example:

```
blocks['s12'] = {      //Select group/scene with Domoticz index 12
  title: 'My group 12'
}
```

Variables

To select a Domoticz variable add 'v' in front of the Domoticz variable ID.

Example:

```
blocks['v3'] = { //Select variable with Domoticz index 3
  title: 'My variable 3'
}
```

After that you can use 'v3' in your column definitions in CONFIG.js as usual.

A list of all Domoticz variables can be obtained via:

```
http://[DomoticzIP:Port]/json.htm?type=command&param=getuservariables
```

Block parameters

Parameter	Description
width	1..12: The width of the block relative to the column width
title	'<string>': Custom title for the block
idx	Index of the Domoticz device id, group/scene id, or variable id you want to use. <idx> or '<idx>': Device idx to use '<idx>_<subidx>': To select subdevice from Domoticz device, like temperature/humidity. 's<idx>': Select group or scene with id <idx> 'v<idx>': Select variable with id <idx>
icon	Defines alternative icon of the device instead of the default, choose from: https://fontawesome.com/icons?d=gallery&m=free 'fas fa-eye'
image	If you want to show an image instead of an icon, place image in img/ folder 'bulb_off.png'
iconOn	Icon to show in case the device state is on. 'fas fa-eye'
iconOff	Icon to show in case the device state is off. 'fas fa-eye'
imageOn	Image to show in case the device state is on. Place image in img/ folder 'bulb_off.png'

Continued on next page

Table 2 – continued from previous page

Parameter	Description
imageOff	Image to show in case the device state is off. Place image in <code>img/</code> folder 'bulb_off.png'
textOn	Text to display in case the device is on.
textOff	Text to display in case the device is off.
switch	<code>true</code> Switch title and data <code>false</code> (default)
hide_data	<code>true</code> Don't show data <code>false</code> (default) Show data field
last_update	<code>true</code> (default) Show the time when this block was updated for the last time <code>false</code> Don't show the last update time for this block
flash	Controls the flashing of the block when it's value changes. <code>0</code> : No flashing (=default) <code>1..1000</code> : Duration (in ms) of the flashing effect
hide_stop	<code>true</code> Hide stop button for applicable devices, like blinds <code>false</code> (Default) Show stop button
playsound	Play a sound when a device changes 'sounds/ping.mp3'
playsoundOn	Play a sound when a device changes to On 'sounds/ping.mp3'
playsoundOff	Play a sound when a device changes to Off 'sounds/ping.mp3'
speak	Speaks text when a device changes 'Device status has changed'
speakOn	Speaks text when a device changes to on 'Device is on'

Continued on next page

Table 2 – continued from previous page

Parameter	Description
speakOff	Speaks text when a device changes to off 'Device is off'
protected	true Protect switching manually in Dashticz (not in Domoticz) false (Default) Switch state can be changed in Dashticz
confirmation	0 No confirmation (default) 1 Dashticz asks the user for confirmation before changing a switch-device
password	Password protect switches, buttons, thermostats, sliders, blinds 'secret ': Password to use
gotoslide	Goto screen when a device changes 1 .. 99
gotoslideOn	Goto screen when a device changes to on 1 .. 99
gotoslideOff	Goto screen when a device changes to off 1 .. 99
popup	This allows the popup to use all the block parameters that a graph block does, allowing users to style the graph. popup: 'popup_your_graph'
graph	Popup graphs enabled by default for most block types. graph: false will disable a popup graph.
openpopup	Open a popup when a device changes. See <i>Usage of openpopup(On)(Off)</i>
openpopupOn	Open a popup when a device changes to on. See <i>Usage of openpopup(On)(Off)</i>
openpopupOff	Open a popup when a device changes to off. See <i>Usage of openpopup(On)(Off)</i>
addClass	The CSS class name, that will be added to the block. 'myclassname': Define 'myclassname' in custom.css
unit	String that will be placed behind the device value to indicate the unit. 'kilowatt ': The string will replace the default unit.
url	'<url>': URL of the page to open in a popup frame or new window on click. For text blocks.

Continued on next page

Table 2 – continued from previous page

Parameter	Description
newwindow	<p>0: open in current window 1: open in new window 2: open in new frame (default, to prevent a breaking change in default behavior) 3: no new window/frame (for intent handling, api calls). HTTP get request. 4: no new window/frame (for intent handling, api calls). HTTP post request. (forcerefresh not supported)</p>
colorpicker	<p>Choose the RGB colorpicker for RGB devices. See <i>RGB Color picker</i></p> <p>0: No RGB colorpicker 1: Old style RGB colorpicker 2: New style RGB colorpicker</p>
mode	<p>Parameter for specific functionality</p> <p>1: Set mode: 1 for Hue RGBWW devices having colorpicker: 2</p>
batteryThreshold	<p>If the battery level is below <code>batteryThreshold</code> then a battery icon will be displayed. See <i>Battery level</i></p> <p>Default value is defined by <code>config['batteryThreshold'] (=30)</code></p> <p>15: Only show the battery icon when the battery level is below 15%.</p>

There are several additional parameters for Graphs. See *Graphs*

Usage

Example of a block definition:

```
var blocks = {}

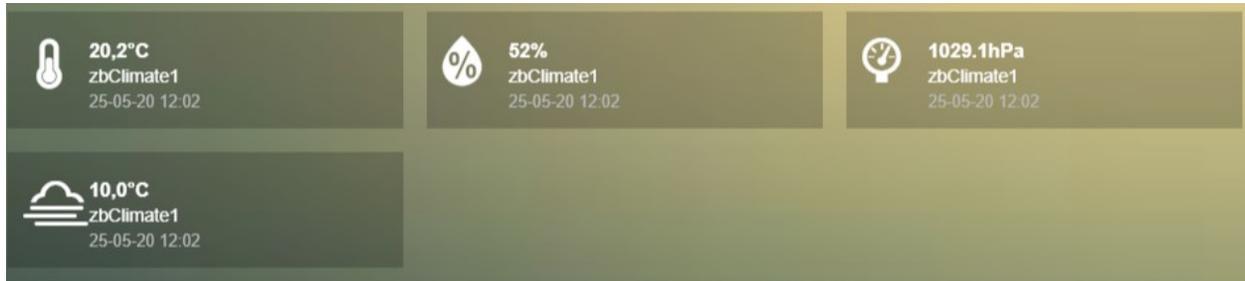
blocks[1] = {
  width: 4, //1 to 12, remove this line if you want to use the default
  ↪(4)
  title : 'Living room', //if you want change the name of switch different then
  ↪domoticz
  icon : 'fa-eye', //if you want an other icon instead of the default, choose
  ↪from: https://fontawesome.com/icons?d=gallery&m=free
  image : 'bulb_off.png', //if you want to show an image instead if icon, place image
  ↪in img/ folder
  switch : true, //if you want to switch the title and data
  hide_data : true, //if you want to hide the data of this block
  last_update : true, //if you want to show the last update specific for this
  ↪block
  playsound : 'sounds/ping.mp3', //play a sound when a device changes
  protected : true, //protect switching manually in Dashticz
  speak : 'Device status has changed', //speak text when device is changed
  gotoslide: 2 //Goto screen when a device changes
};
```

Device with subdevices

If a device consists of several subdevices, like a TempHumBar device or SmartMeter, then for each subdevice a block will be generated.

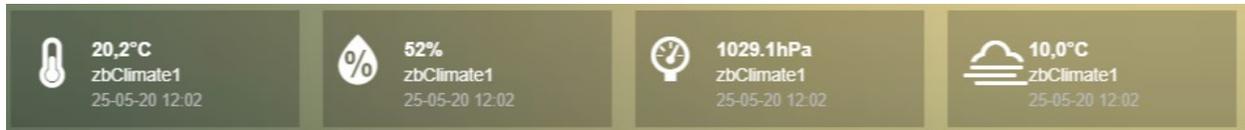
In this example device device 659 is a TempHumBar device:

```
columns[1] = {
  blocks: [659]
}
```



In case I want to show all four subdevices onto one row I've to change the default width from 4 to 3:

```
blocks[659] = {
  width:4
}
columns[1] = {
  blocks: [659]
}
```



Now assume I want to have the first 3 subdevices on one row, and the fourth device on a new row, full width, with some additional customizations:

```
blocks[659] = {
  width:4
}

blocks['659_4'] = {
  width:12,
  title: 'Dew temperature of device 659',
  icon: 'fas fa-bus',
  last_update: 'false',
  switch: true
}

columns[1] = {
  blocks: [659]
}
```

In the previous example first the settings of ``block[659]`` will be applied to all `↳subblocks`, followed by a subblock if it has been defined. (In this case ``blocks['659_4']``)



In case you only want to show subdevice 1, the column definition should be as follows:

```
columns[1] = {
  blocks: [ '659_1' ]
}
```

Don't forget the tick marks around 659_1

As for single device it's also possible to use a custom block key in combination with the idx parameter.

To make this visible I've defined two classes in custom.css:

```
.css_red {
  background-color: red !important;
}

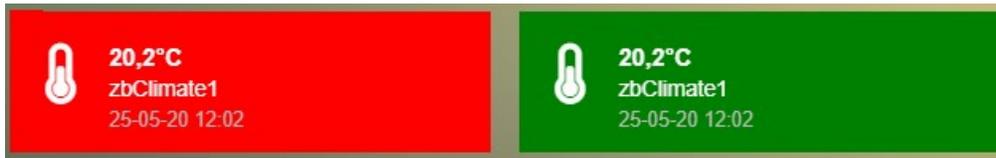
.css_green {
  background-color: green !important;
}
```

Now I'll add the temperature twice, with different backgrounds:

```
blocks['659_1'] = {
  addClass: 'css_red'
}

blocks['another'] = {
  idx: '659_1',
  addClass: 'css_green'
}

columns[1] = {
  blocks: [ '659_1', 'another' ]
}
```



You can also change a subdevice of a block with custom key:

```
blocks['another'] = { //This block will show domoticz device 659
  idx: 659,
  addClass: 'css_red'
}
```

(continues on next page)

(continued from previous page)

```
blocks['another_1'] = { //This block will be applied to subdevice 1 of "another"
  addClass: 'css_green'
}

columns[1] = {
  blocks: [ 'another' ]
}
```



Thermostat devices

For a thermostat IDX, IDX_1 or IDX_2 can be used. If IDX_1 is used the thermostat +/- buttons will not be shown. If IDX_2 is used the icon/image of the block can be changed as in a normal block.

```
blocks['123_2'] = {
  image: 'toon.png'
}
```

Usage of popup graph window

With the popup parameter you can configure to open a popup graph window. Example:

```
blocks[258] = {
  title: 'Consumption',
  flash: 500,
  width: 4,
  popup: 'popup_consumption'
}
```

In this example, the specified popup will use a defined graph called 'popup_consumption' instead of the default popup. This defined graph is then added to the config.js just like a normal graph:

```
blocks['popup_consumption'] = {
  title: 'Energy Consumption Popup',
  devices: [258],
  tooltipStyle: true,
  datasetColors: ['red', 'yellow'],
  graph: 'line',
  legend: {
    'v_258' : 'Usage',
    'c_258' : 'Total'
  }
}
```

Usage of popup multi block window

With the popup parameter you can also configure to open a popup multi block window. Example:

```
blocks['your_block'] = {
  popup: 'container',
  ...
}
```

```
blocks['container'] = {
  blocks: [ 'one1', 'two2'] // where 'one1' and 'two2' are other blocks
}
```

Usage of openpopup(On)(Off)

With the openpopup, openpopupOn and openpopupOff parameter you can configure to open a popup window when the device changes. Example:

```
blocks[123]['openpopup'] = {
  url: 'http://www.urltocamera.nl/image.jpg', //Open a popup window with this url
  ↪when the device changes
  framewidth:500, //specific width of the frame
  frameheight:400, //specific height of the frame
  auto_close: 5 //auto close the popup window after
  ↪5 seconds.
}

blocks[123]['openpopupOn'] = {
  url: 'http://www.urltocamera.nl/image.jpg', //Open a popup window with this url
  ↪when the device changes to On
  framewidth:500, //specific width of the frame
  frameheight:400, //specific height of the frame
  auto_close: 5 //auto close the popup window after
  ↪5 seconds.
}

blocks[123]['openpopupOff'] = {
  url: 'http://www.urltocamera.nl/image.jpg', //Open a popup window with this url
  ↪when the device changes to Off
  framewidth:500, //specific width of the frame
  frameheight:400, //specific height of the frame
  auto_close: 5 //auto close the popup window after
  ↪5 seconds.
}
```

To remove the close button of the block-popup add the following text to custom.css:

```
.frameclose { display: none; }
```

Flash on change

To control the flashing of the block when it's value change you can set the flash parameter. Via the style blockchange in custom.css you can set the class-style that needs to be applied.

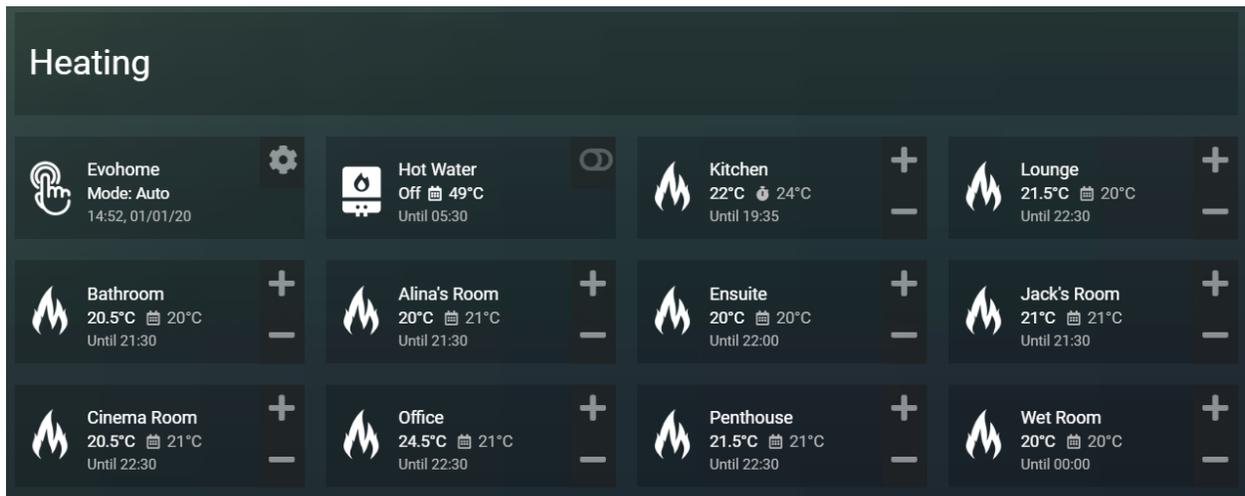
Example CONFIG.js:

```
blocks[123] = {
    //123 is the Domoticz device ID
    title: 'My new device',
    flash: 500 //flash effect of 500 ms
}
```

Example `custom.css` (only needed in case you want to change the default flash effect):

```
.blockchange {
    background-color: #0f0 !important;
}
```

Evohome



The following config parameters from `CONFIG.js` are applicable:

Parameter	Description
<code>evohome_status</code>	'Auto':
<code>evohome_boost_<zone number></code>	: Zone boost temporary override time in minutes. Default: 60
<code>evohome_boost_<hw number></code>	: Hot water boost temporary override time in minutes. Default: 15

The EvoHome devices can be represented as dial by adding `type: 'dial'` to the block definition. See [Dial](#)



Formatting

You can define the default unit text and number of decimals to show for some (most?) blocks by adding the following to CONFIG.js:

```
config['units'] = {
  names: {
    kwh: 'kWh',
    watt: 'W',
    gas: 'm3',
    water: 'l',
    time: ''
  },
  decimals: {
    kwh: 1,
    watt: 0,
    gas: 1,
    water: 0,
    time: 0
  }
}
```

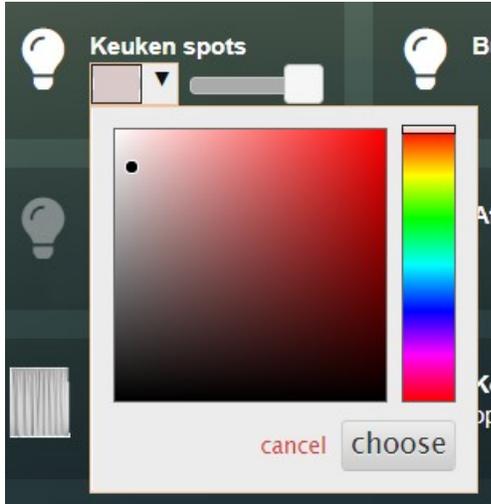
You can also define the unit parameter on block level by setting the `unit` parameter:

```
blocks[123] = {
  unit: 'Watt'
}
```

RGB Color picker

By setting the block parameter `colorpicker` to a non-zero value a color picker dropdown button will be added to a RGB device.

With `colorpicker:1` the old style colorpicker will be added:



With `colorpicker:2` the enhanced colorpicker will be selected. The colorpicker configuration depends on the RGB type. The behavior is the same as in Domoticz.

The following Domoticz RGB devices are supported:

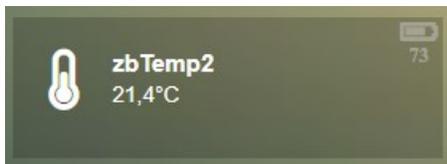
- RGB: Plain RGB dimmer
- RGBW: RGB dimmer with white modus
- RGBWW: RGB dimmer with white modus and adjustable white color temperature
- RGBWZ: Dimmer with seperate adjustable levels for RGB and White leds
- RGBWWZ: Adjustable levels for RGB and White, adjustable white color temperature

For Hue RGBWW device add the following block parameter for correct functioning:

```
mode: 1
```

Battery level

A battery level indicator will be displayed when the battery level is below a certain threshold.



For battery powered devices the minimum level is 0, and the maximum level 100. For devices without a battery the battery level will always be 255.

To display the battery indicator for all battery powered devices set the `batteryThreshold` to 100:

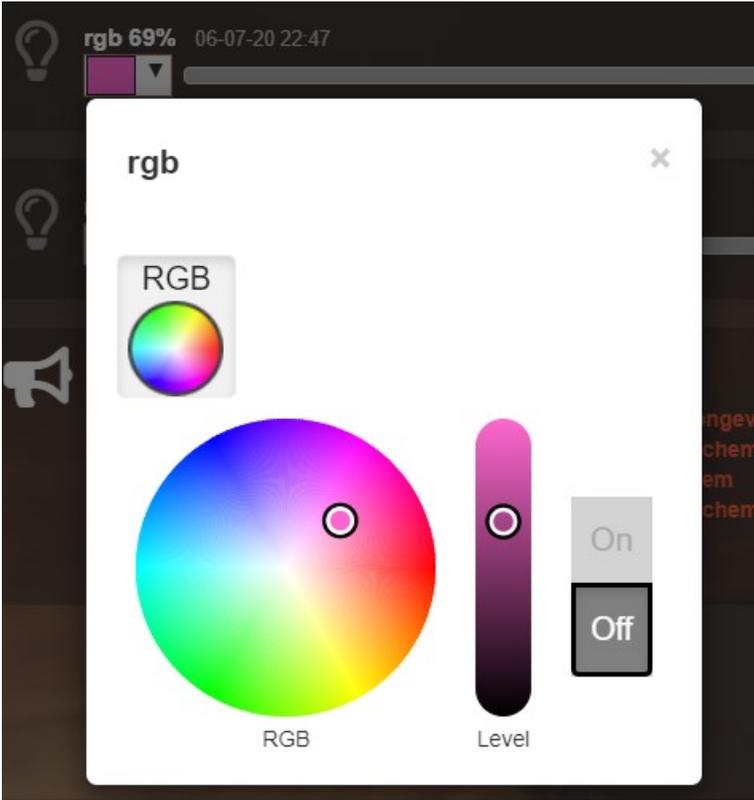


Fig. 1: RGB device



Fig. 2: RGBW device in white modus.

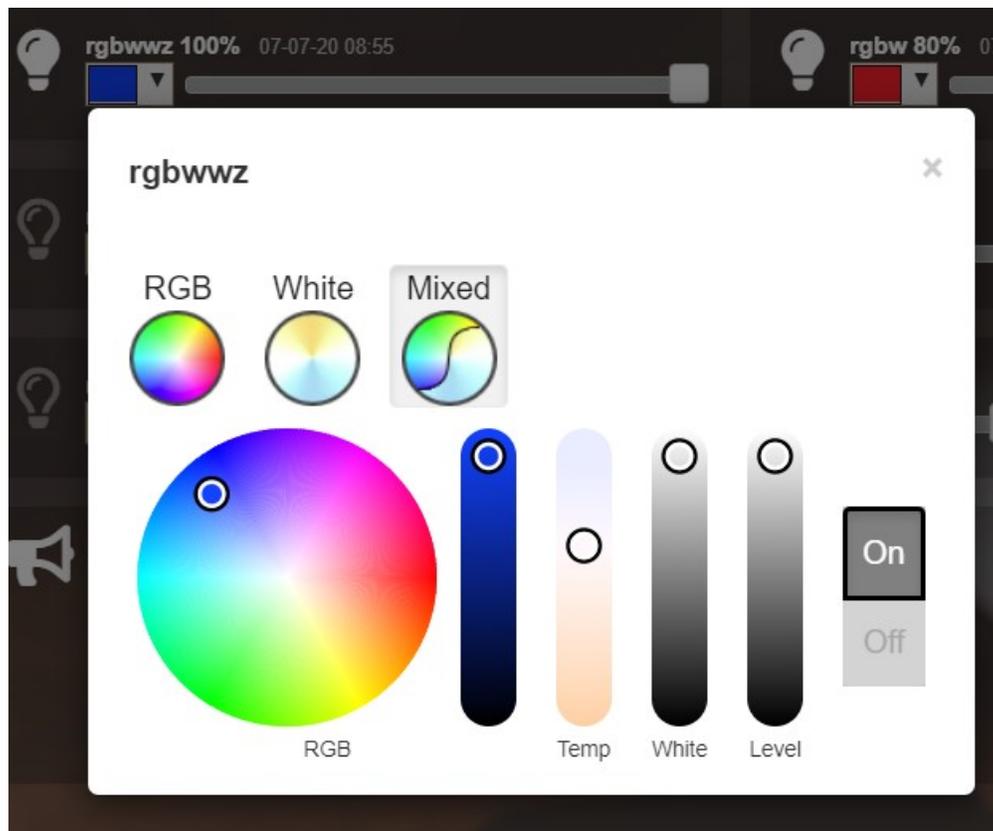


Fig. 3: RGBWWZ device in Mixed modus.

In this last example you see from left to right the RGB color picker, the RGB color level, the white color temperature, the white level and the master level.

```
config['batteryThreshold'] = 100;
```

or configure it for a specific block:

```
blocks[123] = {
  batteryThreshold: 100
}
```

The following indicators will be displayed:

Min battery level	Max battery level	icon
0	10%	fa-battery-empty
10%	35%	fa-battery-quarter
35%	60%	fa-battery-half
60%	90%	fa-battery-three-quarters
90%	100%	fa-battery-full

The battery symbols can be styled in custom.css. As an example the default styling for battery empty:

```
.battery-level.fa-battery-empty {
  color:red;
  z-index: 15;
}
```

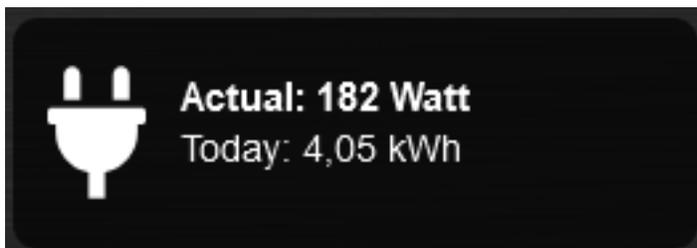
To hide the number, add the following to custom.css:

```
.battery-percentage {
  display:none
}
```

Multiple Values Block

Assuming your device is a P1 smart meter, you can use the following block definition:

```
blocks['customblock'] = {
  idx: '43_1',
  title:"Actual: <Usage>",
  value: "Today: <NettCounterToday> kWh",
  format:true,
  decimals: 1
}
```



Instead of 43 use the Domoticz device ID of your own device.

For as well the title field as the value field of a Domoticz block you can indicate to fill in the value of a Domoticz device parameter by using the '<' and '>' symbol.

For a smart meter device you can use the following device parameters:

- Usage (=Actual power entering your house)
- UsageDeliv (=Actual power leaving your house)
- NettUsage (=Usage - UsageDeliv)
- CounterToday (=Energy entered your house today)
- CounterDelivToday (=Energy left your house today)
- NettCounterToday (=CounterToday - CounterDelivToday)
- Counter (=Total counter of energy that entered your house)
- CounterDeliv (=Total counter of energy that left your house)
- NettCounter (=Counter - CounterDeliv)

All parameter names are case sensitive.

For more fancy computations you can create your custom `getStatus` function in `custom.js`

Styling

The following CSS classes will be attached to a Domoticz block automatically:

- `on`. In case a Domoticz switch is in the On (or closed) state
- `off`. In case a Domoticz switch is not in the On (or closed) state.
- `timeout`. In case the Domoticz device is in timeout state.

To give Domoticz blocks with a device in timeout state a red background, add the following to `custom.css`:

```
.mh.timeout {
  background-color: rgba(255, 0, 0, 0.3);
}
```

3.2.2 Graphs

If your Domoticz device contains a value (temperature, humidity, power, etc.) then when you click on the block a popup window will appear showing a graph of the values of the device. These popups can be customized by using the 'popup' parameter.

Besides popup graphs it's also possible to show the graph directly on the dashboard itself.

If you want to show the (undefined) graph of the data of a single device, you can add the `graph-id` to a column definition as follows:

```
//Adding a graph of device 691 to column 2
columns[2]['blocks'] = [
  ...,
  'graph_691',      //691 is the device id for which you want to show the graph
  ...
]
```

For a defined graph you have to add a block definition to `CONFIG.js` and add the `devices-parameter`:

```
blocks['my_graph'] = { //my_graph can be any name you want, as long as you add
↳'devices' to the block
    ...,
    devices: [691], //691 is the device id
    ...
}

columns[2]['blocks'] = [
    ..., 'my_graph', ...
]
```

It's also possible to combine the data from several devices into one graph. In that case you have to add a block definition to CONFIG.js and add the devices-parameter:

```
blocks['my_multi_graph'] = { //my_multi_graph can be any name you want, as long as
↳you add 'devices' to the block
    devices: [691, 692] //691 and 692 are the device id's you want to have combined
↳into one graph
}

columns[2]['blocks'] = [
    ...,
    'my_multi_graph',
    ...
]
```

The following block parameters can be used to configure the graph:

Parameter	Description
devices	an array of the device ids that you want to report on, e.g. for single graph [22] or multigraph [17, 18, 189]
graph	Sets the graph type 'line' Line graph (default) 'bar' Bar graph
graphTypes	Array of values you want to show in the graph. Can be used for Domoticz devices having several values. ['te']: Temperature ['hu']: Humidity ['ba']: Barometer ['gu', 'sp']: wind guts and speed ['uvi'], ['lux'], ['lux_avg'], ['mm'], ['v_max'] ['v2'], ['mm'], ['eu'], ['u'], ['u_max'], ['co2']

Continued on next page

Table 3 – continued from previous page

Parameter	Description
groupBy	<p>This allows users to group their data by hour, day, week or month, where applicable ranges are used. See groupBy.</p> <p>The GroupBy function will either:</p> <ul style="list-style-type: none"> - The <i>Sum</i> of all values together for that group - Provide the <i>Average</i> of all values for that group <p>It identifies what type of sensor it is to apply to appropriate calculation:</p> <ul style="list-style-type: none"> - Counter, Rain – uses the <i>Add</i> calculation - Temperature, Custom Sensor and Percentage – uses the <i>Average</i> calculation
aggregate	<p>In a graph block you can add the parameter ‘aggregate’ which can have the value <i>sum</i> or <i>avg</i> to define how to compute the aggregation.</p> <p>sum: show the sum. avg: shows the average.</p>
groupByDevice	<p>allowing user to show the status of several devices in a single graph. Instead of the data being group by time intervals, it is grouped by the devices. See groupByDevice.</p> <p>false: disables the feature (default) true: enables the feature and display a vertical bar chart, grouped by device 'vertical': is the same as true 'horizontal': enables the feature and display a horizontal bar chart, grouped by device</p> <ul style="list-style-type: none"> - Setpoint devices will be displayed as a line in front of the bar graph - For Evohome devices: The tooltip info will display the status and schedule
labels	<p>An array to rename the device names (groupByDevice charts only) ['Device A', 'Device B'] The first device will be labeled as Device A, the second as Device B</p>
stacked	<p>true: Show stacked bar charts. See stacked.</p>
beginAtZero	<p>This forces the Y axis to begin at 0 (zero). The beginAtZero setting can accomodate multiple Y axes. For example, for a graph with 3 Y axes, you can use: beginAtZero: [true, false, true] For a graph with a single Y axis, you can use: beginAtZero: true</p>
height	<p>'300px': Height of the graph in the graph block</p>
width	<p>6: The width of the block relative to the column width.</p>
displayFormats	<p>Object to set the time display format on the x-axis. See Time format on the x-axis.</p>
ylabels	<p>To define the y-axes for a custom graph. See Y-axis for custom graphs.</p>
custom	<p>Customized graph. See Custom graphs.</p>
interval	<p>a time based limiter, to limit time data, e.g. 2 will show 1/2 the time labels, 5 will show 20% of the time labels (default is 1)</p>
maxTicksLimit	<p>specifies how many labels (ticks) to display on the X axis, this does not limit the data in the graph, e.g. 10 (default is all)</p>
cartesian	<p>scales the graph with standard ‘linear’ scale, or ‘logarithmic’, an algorithm to ensure all data can be seen (default is linear)</p>

Continued on next page

Table 3 – continued from previous page

Parameter	Description
datasetColors	datasetColors: ['Blue', '#D3D3D3', 'rgb(44,130,201)', 'rgba(44,130,201,1)'] Use custom colors for the graph lines/bars. Must be <i>html colors</i> , <i>hex code</i> , <i>rgb</i> or <i>rgba string</i> . See <i>Custom colors</i> .
iconColour	colours the graph's title icons (default is grey)
fontColor	font color for the axis ticks and labels (default is white)
lineFill	if line graph, this fills the graph, it is an array for each dataset, e.g. ['true', 'false', 'true'] (default is false)
axisRight	if true the first Y axis will be shown at the right. (default is false, meaning left)
axisAlternate	if true then in case of multiple Y axes they will be shown alternating left/right
borderWidth	this is actually the width of the line (default is 2)
borderDash	use if you want a dashed line, it takes an array of two values; length of the line and the space, e.g. [10, 10] (default is off)
borderColors	handy for bar graphs, takes an array of colours like datasetColors, e.g. ['red', 'green', 'blue'] (default uses datasetColors)
pointRadius	the size of each data point, e.g. 3 (default is 1)
pointStyle	an array of the shape of each point, such as circlecrossldashllinelectstarltriangle, e.g. ['star', 'triangle'] (default is circle)
pointFillColor	an array containing the colour of each point, e.g. ['red', 'green', 'blue'] (default uses datasetColors)
pointBorderColor	an array containing the border colour of each point, e.g. ['red', 'green', 'blue'] (default is light grey)
pointBorderWidth	the thickness of the point border, e.g. 2 (default is 0)
barWidth	if a bar graph, this is the width of each bar, 0-1, e.g. 0.5 is half bar, half gap (default is 0.9)
reverseTime	use this if you want to reverse your X axis, i.e. setting 'true' would mean the time will be reversed (default is false)
lineTension	sets the bezier curve the line is, 0 is straight, 1 is extremely curved! e.g. 0.4 gives a nice bendy line (default is 0.1)
drawOrderLast	an array stating the order in which each dataset should be added to the graph for "last hours", e.g. ['v_idx2', 'v_idx1']
drawOrderDay	an array stating the order in which each dataset should be added to the graph for "today", e.g. ['v_idx3', 'v_idx1', 'v_idx2']
drawOrderMonth	an array stating the order in which each dataset should be added to the graph for "last month", e.g. ['v_idx1', 'v_idx2', 'c_idx1', 'c_idx2']
buttonsBorder	color of the buttons border, e.g. 'red', default is 'white'
buttonsColor	color of the buttons text, e.g. '#fff' or 'white', default is 'black'
buttonsFill	color of the buttons background colour, e.g. '#000' or 'black', default is 'white'
buttonsIcon	color of the buttons icon, e.g. 'blue', default is 'grey'
buttonsMarginX	gap (or margin) between the buttons (left and right), e.g. 5, default is 2
buttonsMarginY	gap (or margin) above and below the buttons, e.g. 5, default is 0
buttonsPadX	padding inside the buttons (left and right), e.g. 10, default is 6
buttonsPadY	padding inside the buttons, top and bottom, e.g. 5, default is 2
buttonsRadius	the curvature of the corners of the buttons, e.g. 10, default is 4
buttonsShadow	the shadow below the button in RGBA format (last number is opacity), e.g. 'rgba(0,0,0,0.5)', default is off
buttonsSize	the size of the button, e.g. 12, default is 14
buttonsText	change the text displayed on each button in an array, e.g. ['Now', 'Today', 'Month'], default is what you see today
gradients	an array of arrays, e.g. gradients: [['white', 'blue'], ['orange', 'powderblue']], default disabled
gradientHeight	a number showing the height of the gradient split, e.g. 0.8, default 1

Continued on next page

Table 3 – continued from previous page

Parameter	Description
spanGaps	If true, lines will be drawn between points with no or null data. If false, points with NaN data will create a break in the line.
sortDevices	the code automatically calculate if any devices' time data is longer than others. It then use that device's time data, then match all of the devices non-time data to that. This setting allows users to choose to enable or disable that feature (true or false)
steppedLine	defines the interpolation method. It can be a single value 'before', or an array of values ['before', false, false] false (default) No stepped line but interpolation true The line steps just before the new value 'before' The line steps just before the new value (same as true) 'after' The line steps just after the new value 'middle' The line steps between the old and the new value
customHeader	customHeader: { ... } Customized graph header. See <i>customHeader</i> .
format	false (default). Show the value in the graph header as reported by Dashticz. true. Format the graph header value using the decimals parameter and the config settings _THOUSAND_SEPARATOR and _DECIMAL_POINT. See <i>Number format</i>
decimals	<number>. Number of decimals to use in tooltip value, and header value (in case format is true)
popup	popup: 'popup_graph' Defined Popups. See <i>Defined Popups</i> .
tooltiptotal	Display graph tooltiptotal instead of the standard one. true, false or an array, e.g. tooltiptotal: ['Office (Temp)', 'Lounge (Temp)'], See <i>tooltiptotal</i> .
zoom	Allows graph zoom controls and orientation. See <i>Zoom</i> . 'x': allow zooming on the x axis (left to right) 'y': allow zooming on the y axis (top to bottom) 'xy': allow zooming in any direction 'false': disable zooming, do not show zoom button
debugButton: true	Users can now debug their graph by setting their graph's block config, e.g. debugButton: true. See <i>Debug</i> .

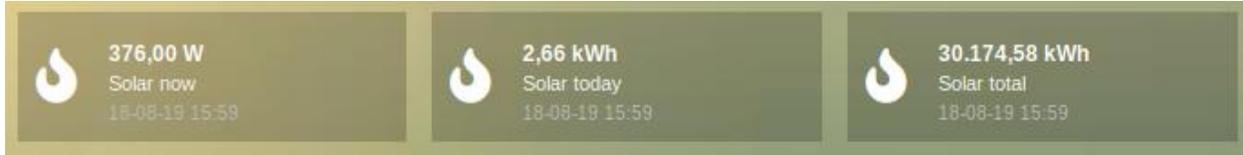
We will show the possibilities by showing a:

- Simple energy device (Solar panel)
- Climate device (temperature, humidity, barometer)
- P1 Smart Meter

Simple energy device

The solar panel device has device id 6. First we add it to a column without any additional configuration parameters, to show the default result:

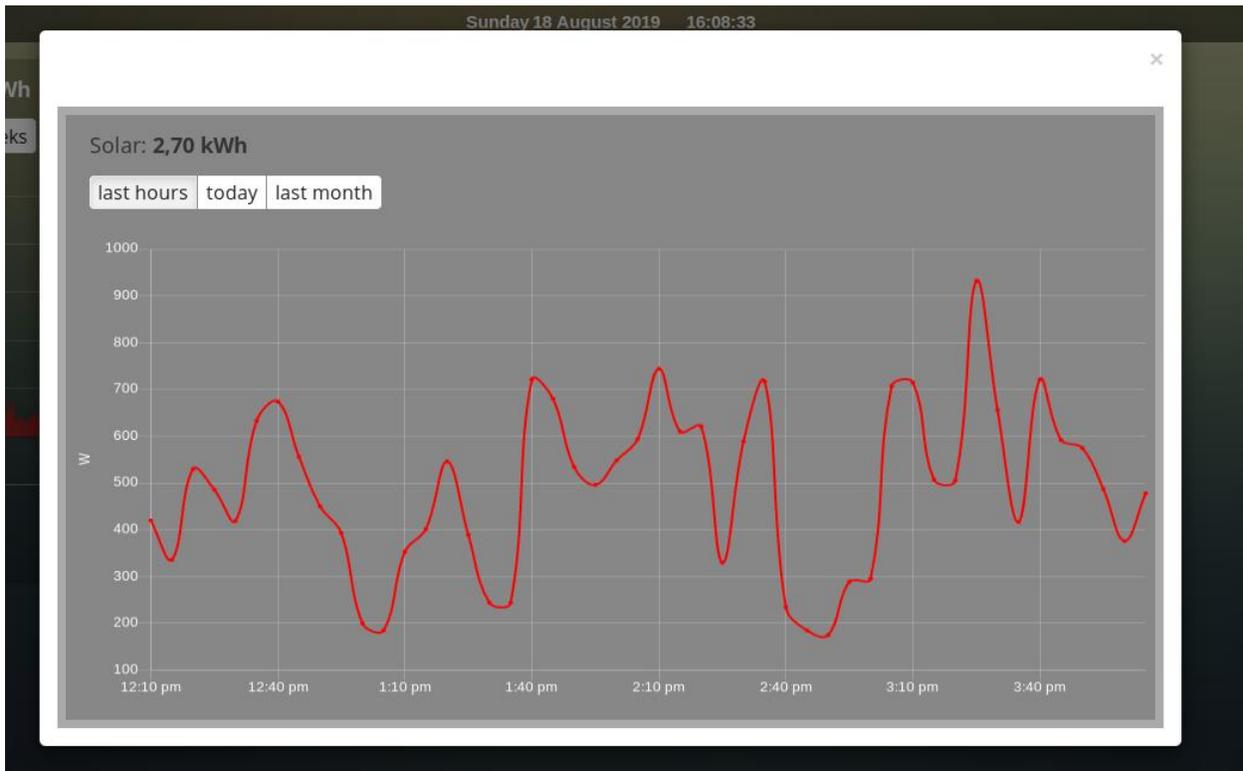
```
columns[2]['blocks'] = [
  6
]
```



As you see three buttons are generated (actual power, energy today, total energy). I only want to have one button, so I change my column definition to:

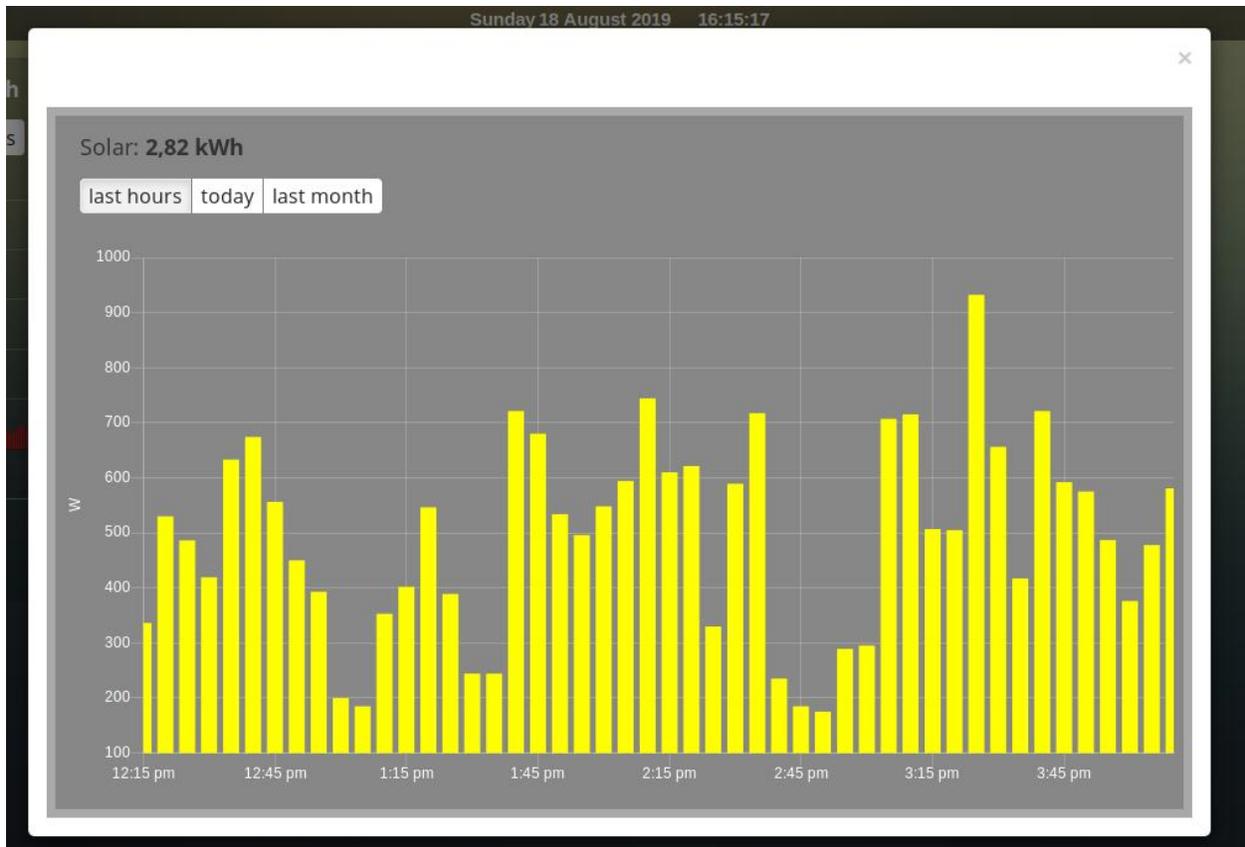
```
columns[2]['blocks'] = [
  '6_1'
]
```

By pressing the button the following graphs pops up:



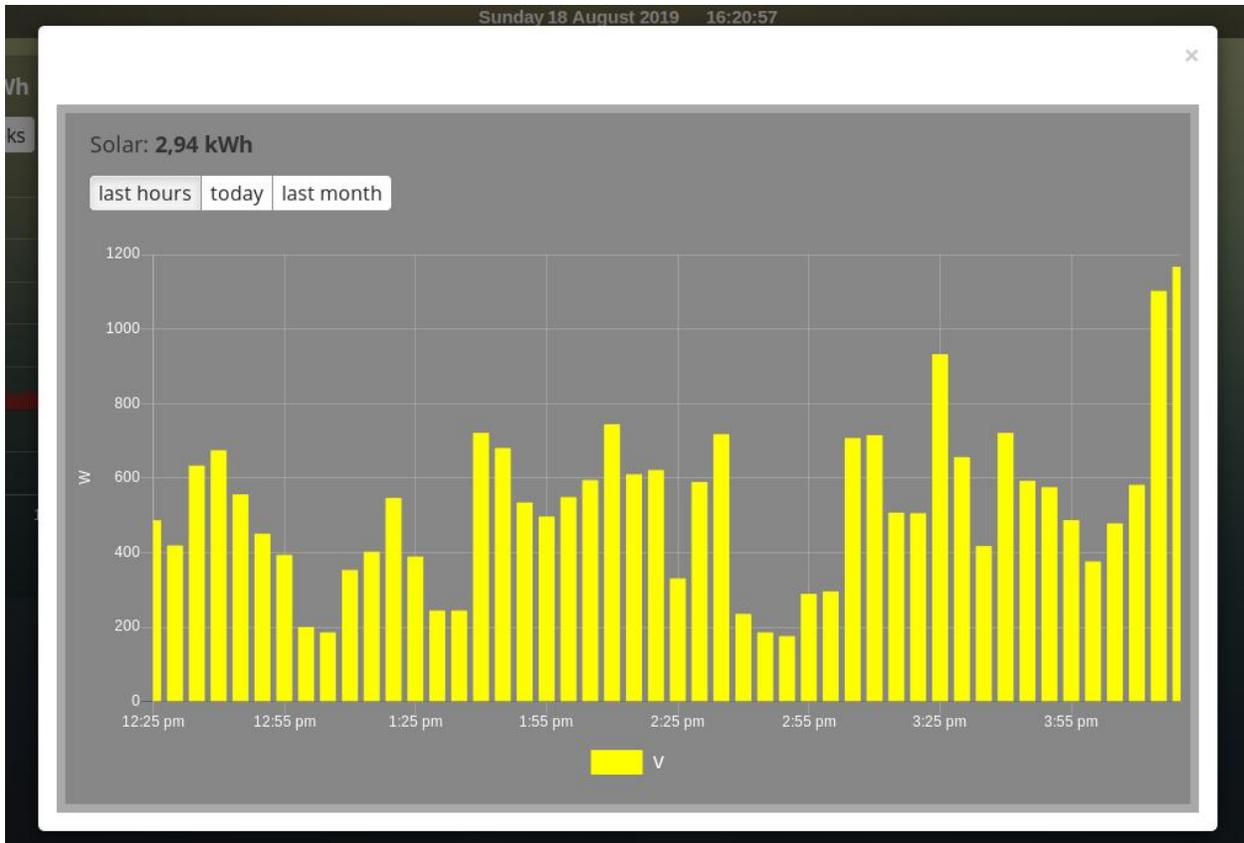
So, nothing special. Only the red line color is maybe a bit too much. Let's change it into a yellow bar graph. We have to add a block definition:

```
blocks['graph_6'] = {
  devices: [6],
  graph: 'bar',
  datasetColors: ['yellow']
}
```



Now I want to add a legend at the bottom:

```
blocks['graph_6'] = {
  devices: [6],
  graph: 'bar',
  datasetColors: ['yellow'],
  legend: true
}
```

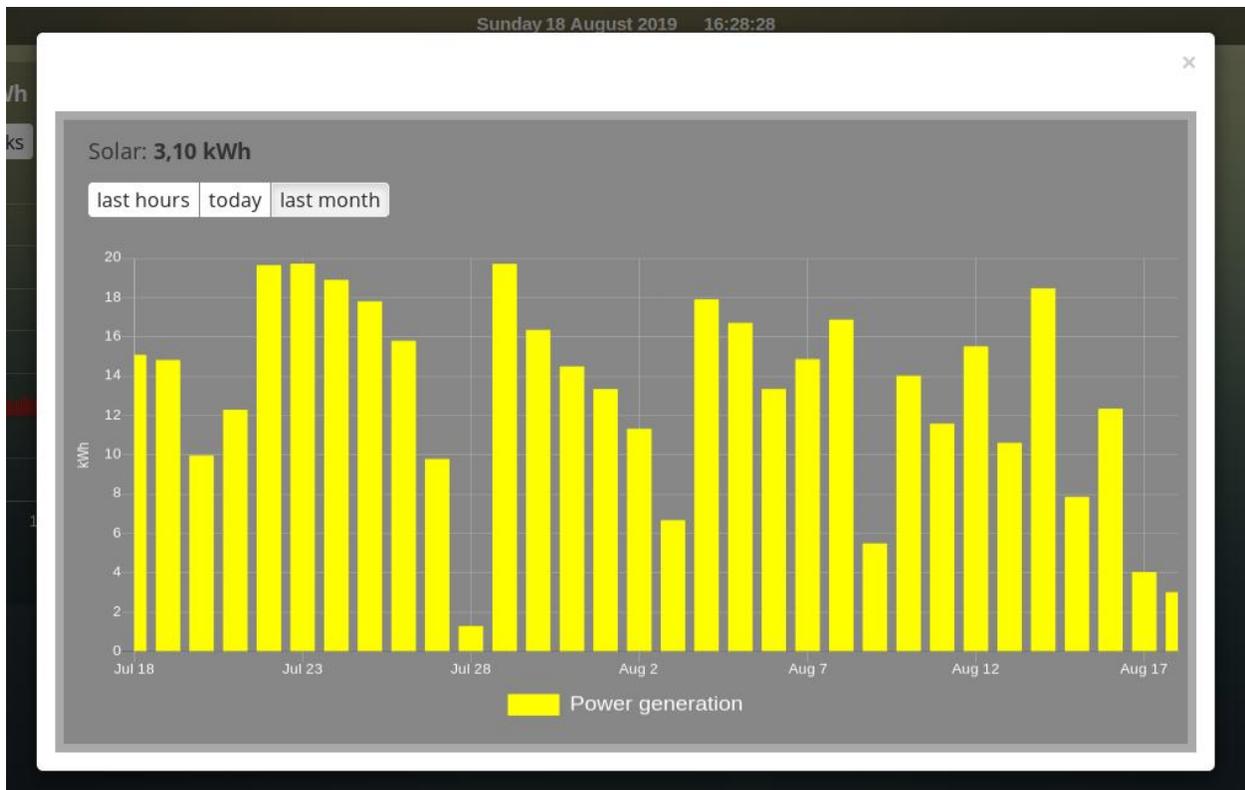


As you can see the data points are labeled as 'V'. This name is generated by Domoticz. We can translate the Domoticz name in something else, by extending the legend parameter

```
blocks['graph_6'] = {
  devices: [6],
  graph: 'bar',
  datasetColors: ['yellow'],
  legend: {
    'v': 'Power generation'
  }
}
```

legend is an object consisting of key-value pairs for the translation from Domoticz names to custom names.

After pressing the 'Month' button on the popup graph:

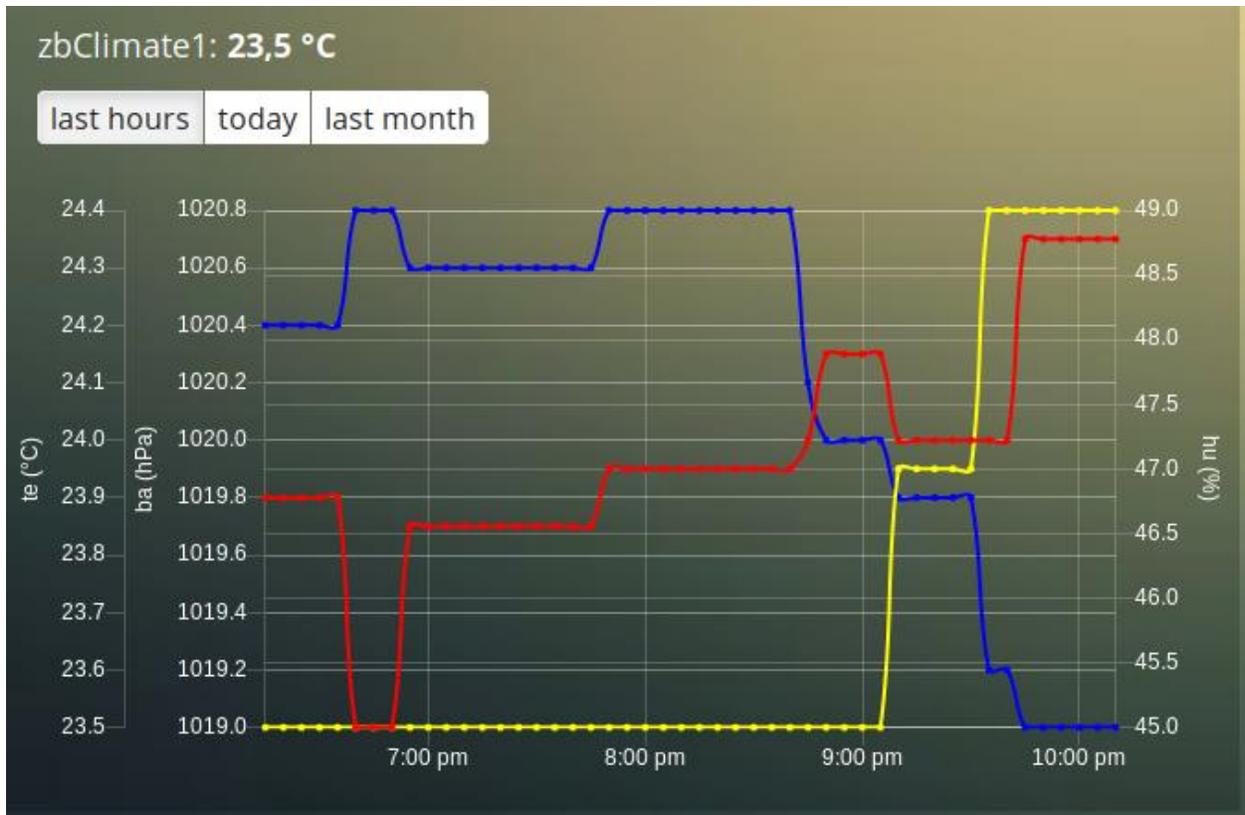


Climate device

First let's add a climate device with Domoticz ID 659 to a column:

```
columns[3]['blocks'] = [
    'graph_659'
]
```

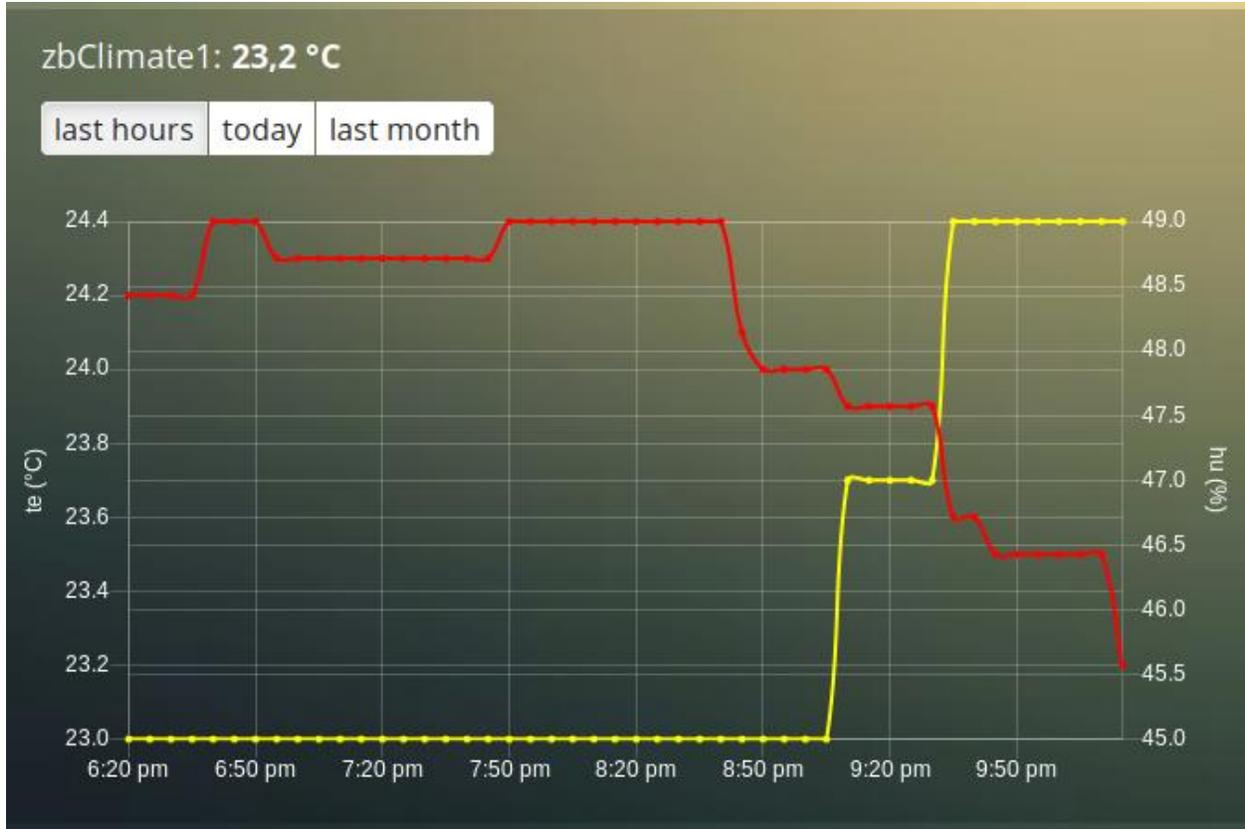
This will show the graph directly on the Dashticz dashboard:



As you can see the climate device has three subdevices (temperature, humidity, pressure). Since these are different properties three Y axes are being created.

If you prefer to only see the temperature and humidity add a block definition:

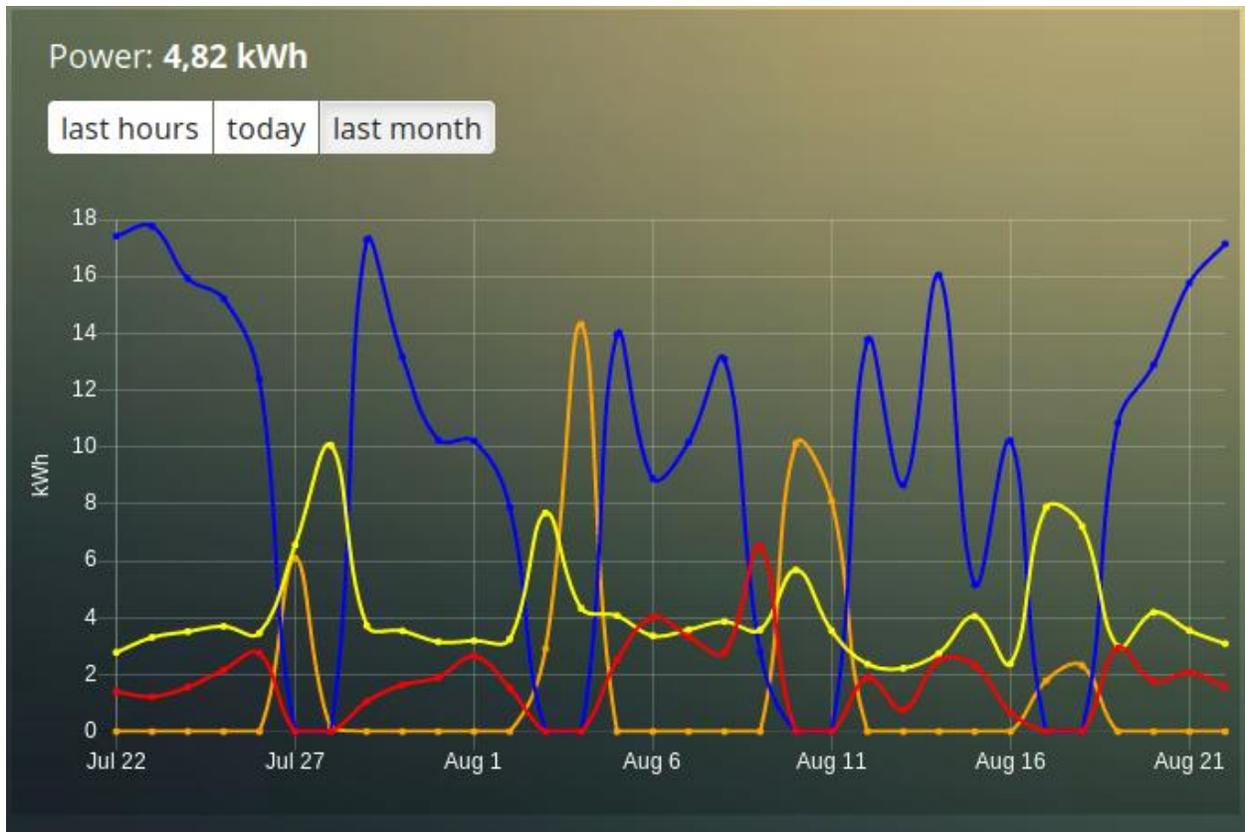
```
blocks['graph_659'] = {
  devices: [659],
  graphTypes : ['te', 'hu'],
  legend: true
}
```



Of course you can add a legend as well. See the previous section for an example.

P1 smart meter

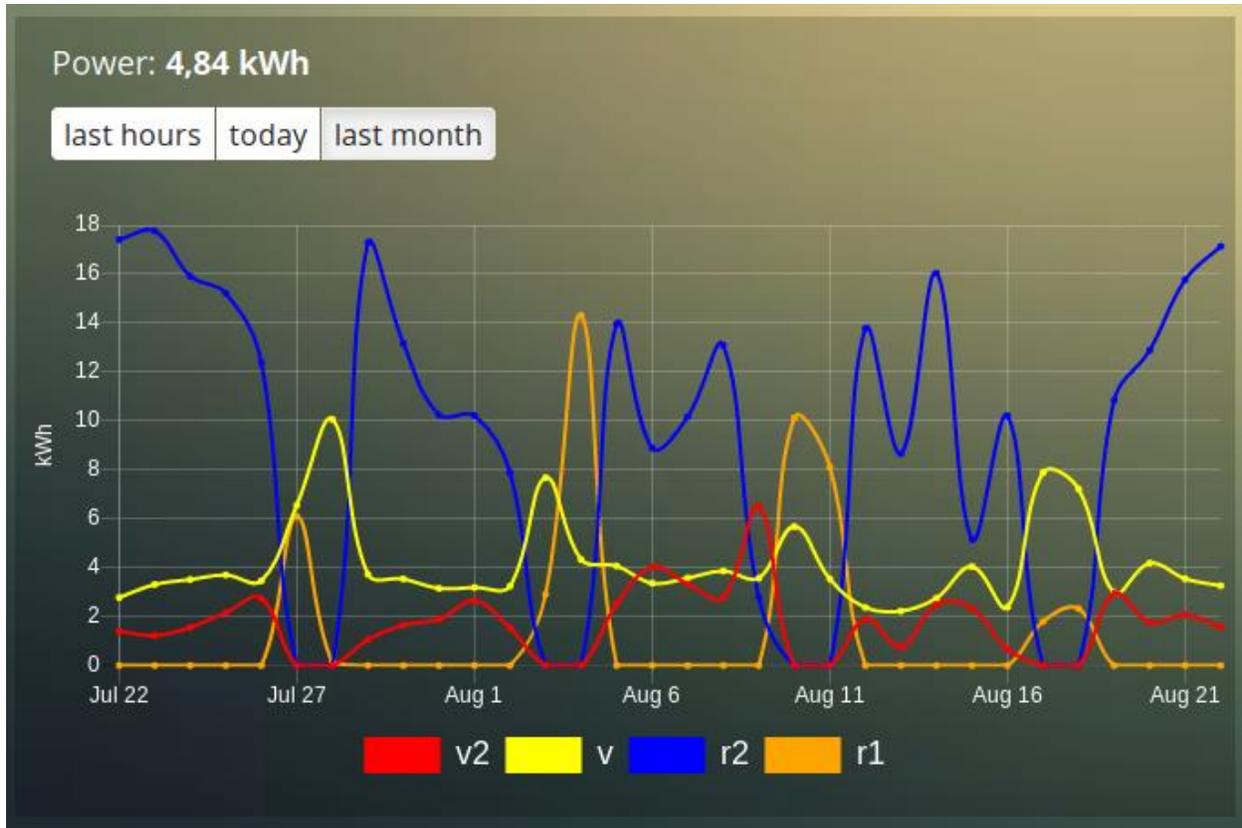
First let's show the default P1 smart meter graph:



You see a lot of lines. What do they mean? Let's add a legend

```
blocks['graph_43'] = {  
    devices: [43],  
    legend: true  
}
```

This gives:



That doesn't tell too much. However, this are the value names as provided by Domoticz. Now you have to know that a P1 power meter has 4 values:

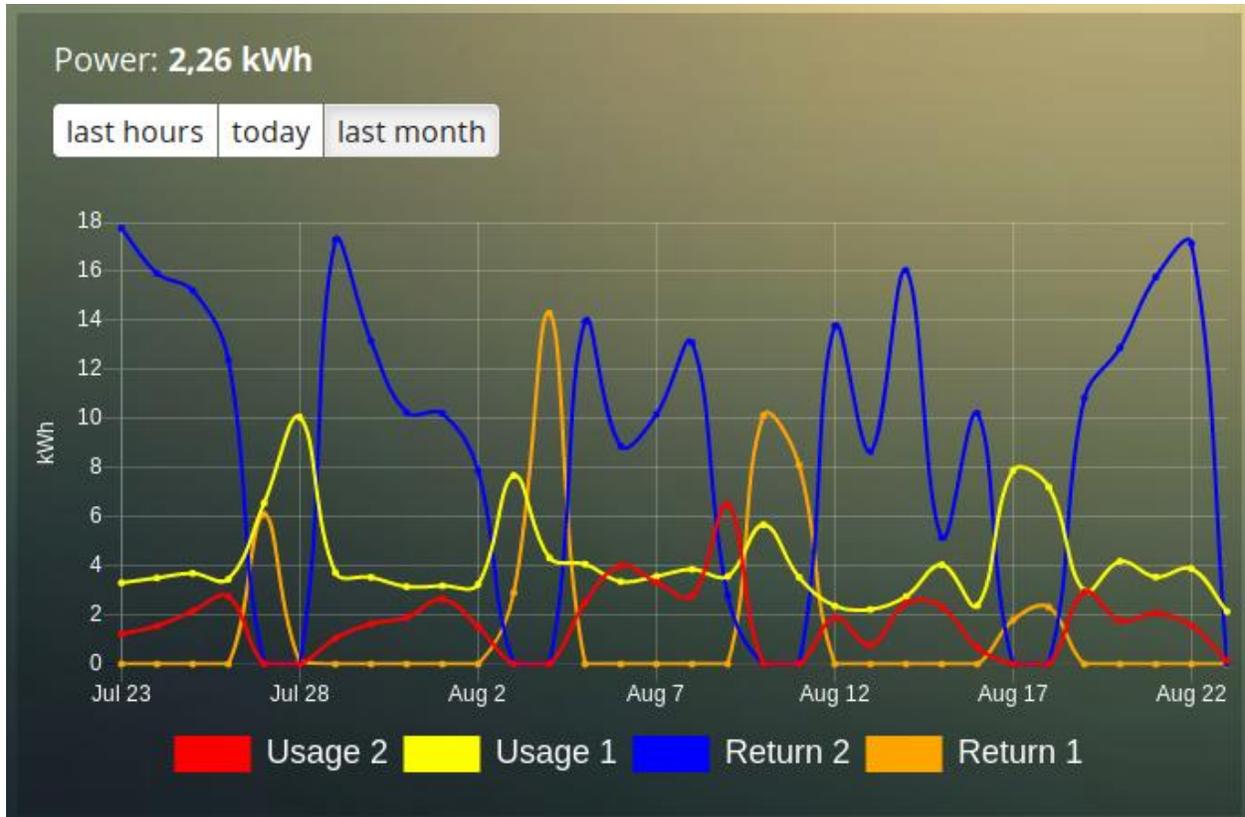
- Power usage tariff 1
- Power usage tariff 2
- Power delivery tariff 1
- Power delivery tariff 2

The first two represent the energy that flows into your house. The last two represent the energy that your house delivers back to the grid.

So we can add a more meaningful legend as follows:

```
blocks['graph_43'] = {
  devices: [43],
  legend: {
    v_43: "Usage 1",
    v2_43: "Usage 2",
    r1_43: "Return 1",
    r2_43: "Return 2"
  }
}
```

Resulting in:



However what I would like to see is:

- The sum of Usage 1 and Usage 2
- The sum of Return 1 and Return 2, but then negative
- A line to show the nett energy usage: Usage 1 + Usage 2 - Return 1 - Return 2
- The usage and return data should be presented as bars. The nett energy as a line.

Can we do that? Yes, with custom graphs!

Custom graphs

I use the P1 smart meter as an example again to demonstrate how to create custom graphs. First the code and result. The explanation will follow after that:

```
blocks['graph_43'] = {
  title: 'My Power',
  devices: [43],
  graph: ['line', 'bar', 'bar'],
  custom : {
    "last day": {
      range: 'day',
      filter: '24 hours',
      data: {
        nett: 'd.v_43+d.v2_43-d.r1_43-d.r2_43',
        usage: 'd.v_43+d.v2_43',
        generation: '-d.r1_43-d.r2_43'
      }
    }
  }
}
```

(continues on next page)

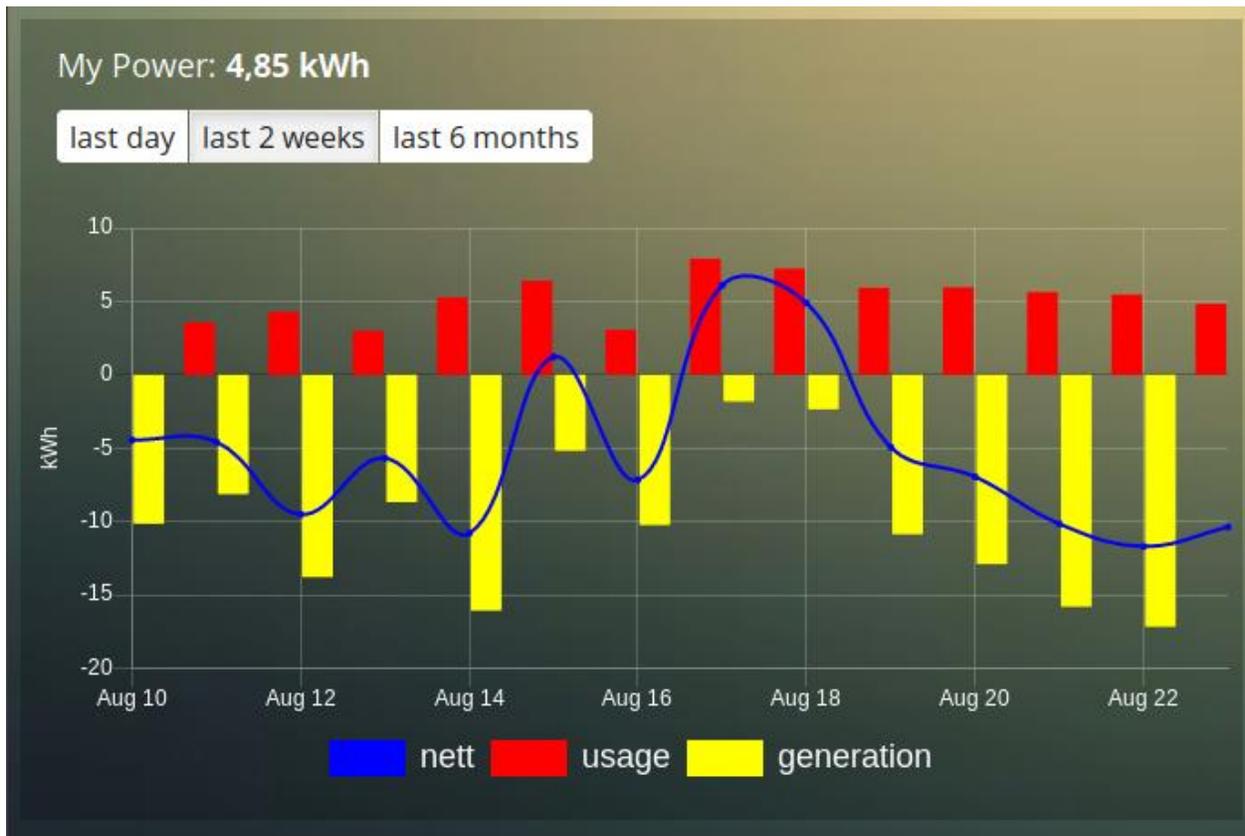
(continued from previous page)

```

    },
    "last 2 weeks": {
      range: 'month',
      filter: '14 days',
      data: {
        nett: 'd.v_43+d.v2_43-d.r1_43-d.r2_43',
        usage: 'd.v_43+d.v2_43',
        generation: '-d.r1_43-d.r2_43'
      }
    },
    "last 6 months": {
      range: 'year',
      filter: '6 months',
      data: {
        nett: 'd.v_43+d.v2_43-d.r1_43-d.r2_43',
        usage: 'd.v_43+d.v2_43',
        generation: '-d.r1_43-d.r2_43'
      }
    }
  },
  legend: true,
  datasetColors:['blue', 'red', 'yellow']
}

```

This will give:



As you can see, the graph has

- title, set via the `title` parameter
- devices, set via the `devices` parameter
- custom colors, defined by the parameter `datasetColors`
- The `graph` parameter is used to define the graph types. This time it's an array, because we want to select the graph type per value.
- legend set to `true`, to show a default legend
- custom buttons, defined by the `custom` parameter

A custom object start with the name of the button. The button should contain the following three parameters:

- `range`. This is the name of the range as requested from Domoticz, and can be `'day'`, `'today'`, `'month'` or `'year'`. The range `'today'` filters the data to today, independent of the setting in Domoticz, and sets the graph x-axis to the full day.
- `filter` (optional). This limits the amount of data to the period as defined by this parameter. Examples: `'2 hours'`, `'4 days'`, `'3 months'`
- `data`. This is an object that defines the values to use for the graph.
- `buttonIcon` (optional). The Fontawesome icon to use for the button. Example `'fas fa-bus'`

As you can see in the example the first value will have the name `'nett'`. The formula to compute the value is:

```
'd.v_idx+d.v2_idx-d.r1_idx-d.r2_idx'
```

The `d` object contains the data as delivered by Domoticz. As you maybe remember from a previous example Domoticz provides the two power usage values (`v_idx` and `v2_idx`) and the two power return values (`r1_idx` and `r2_idx`).

So the first part sums the two power usage values (`d.v_idx+d.v2_idx`) and the last parts subtracts the two return values (`-d.r1_idx-d.r2_idx`),

The two other value-names in the data object (`usage` and `generation`) will compute the sum of the power usage values and the power return values respectively.

Maybe a bit complex in the beginning, but the Dashticz forum is not far away.

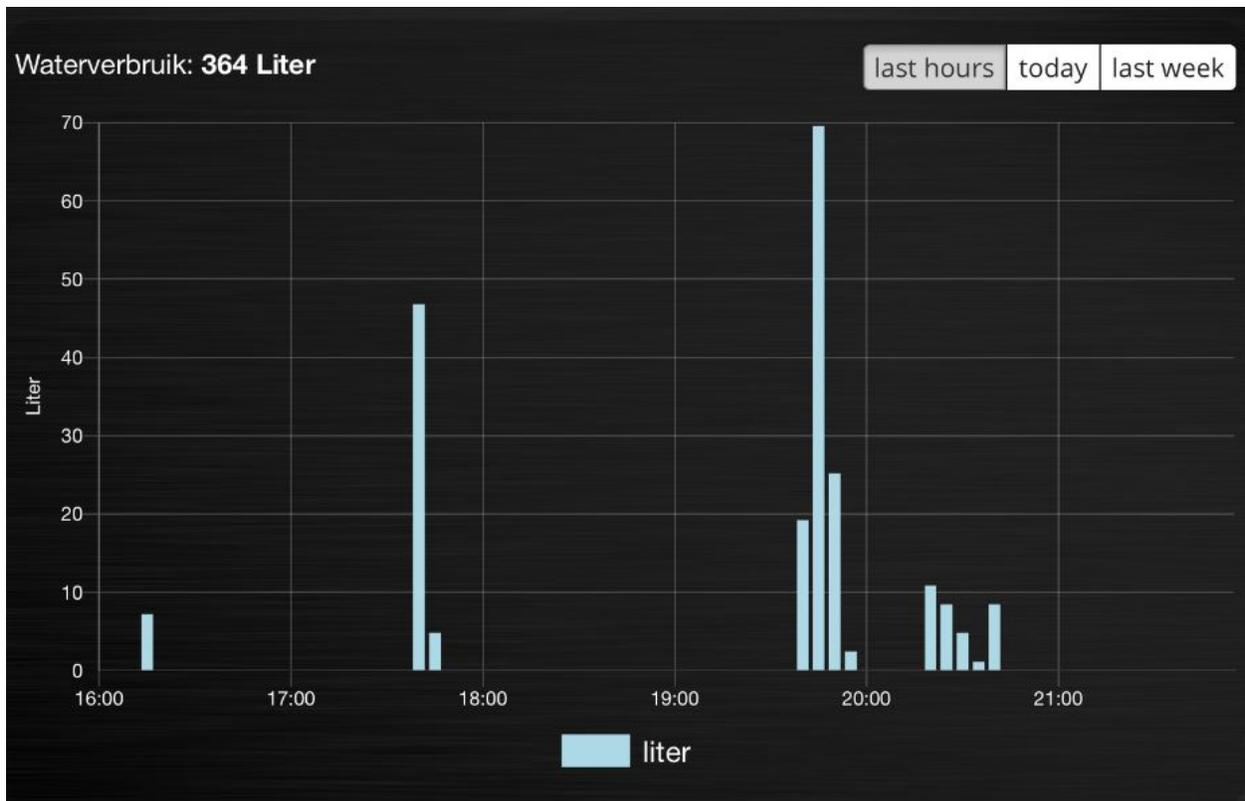
Below another example to adapt the reported values of a watermeter to liters:

```
blocks['graph_903'] = {
  graph: 'bar',
  devices: [903],
  datasetColors: ['lightblue'],
  legend: true,
  custom : {
    "last hours": {
      range: 'day',
      filter: '6 hours',
      data: {
        liter: 'd.v_903*100'
      }
    },
    "today": {
      range: 'day',
      filter: '12 hours',
      data: {
        liter: 'd.v_903*100'
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```
"last week": {
  range: 'month',
  filter: '7 days',
  data: {
    liter: 'd.v_903*1000'
  }
}
}
```



Time format on the x-axis

The chart module uses moments.js for displaying the times and dates. The locale will be set via the Domoticz setting for the calendar language:

```
config['calendarlanguage'] = 'nl_NL';
```

To set the time (or date) format for the x-axis add the `displayFormats` parameter to the block definition:

```
blocks['graph_6'] = {
  devices: [6],
  displayFormats: {
    minute: 'h:mm a',
    hour: 'hA',
    day: 'MMM D',
    week: 'll',
  }
}
```

(continues on next page)

(continued from previous page)

```

    month: 'MMM D',
  },
}

```

The previous example sets the time formats to UK style. See <https://www.chartjs.org/docs/latest/axes/cartesian/time.html#display-formats> for time/date formats.

Modifying the y-axes

You can modify the y-axes by setting the options parameter. Below you see an example how to define the min and max values of two y-axes:

```

blocks['graph_659'] = {
  devices: [659],
  graph: 'line',
  graphTypes: ['te', 'hu'],
  options: {
    scales: {
      yAxes: [{
        ticks: {
          min: 0,
          max: 30
        }
      }, {
        ticks: {
          min: 50,
          max: 100
        }
      }
    ]
  }
}

```

The `yAxes` parameter in the `options` block is an array, with an entry for each y-axis.

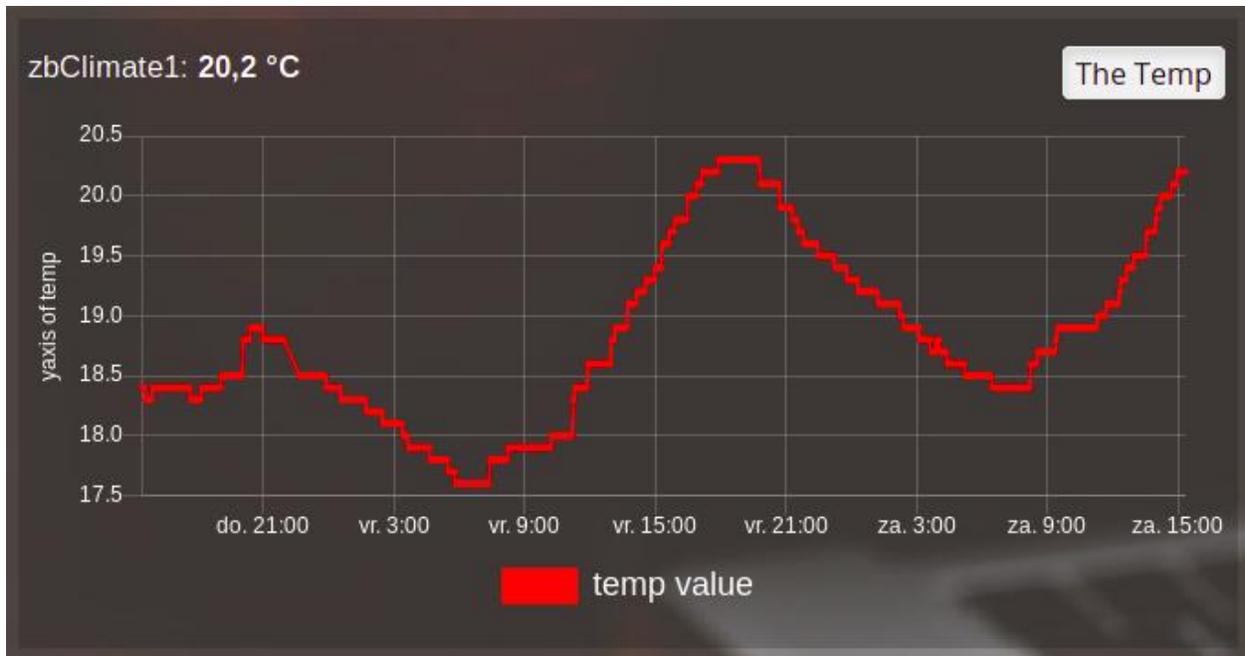
Y-axis for custom graphs

To define the y-axes for a custom graph you can add the `ylabels` parameter as follows:

```

blocks['graph_659'] = {
  devices: [659],
  custom: {
    'The Temp': {
      ylabels: ['yaxis of temp'],
      data: {
        'temp value': 'd.te_659'
      },
      range: 'day',
      filter: '2 days',
      legend: true
    }
  },
  width: 6
}

```



The parameter `ylabels` is an array. You can add a string for each value of the data object.

Custom colors

Custom colors can be defined by the parameter `datasetColors`:

```
datasetColors: ['red', 'yellow', 'blue', 'orange', 'green', 'purple']
```

Optional: If you want to use *custom color names* you have to set the variable `dataset colors` to *html colors*, *hex code*, *rgb* or *rgba string*:

```
datasetColors: [colourBlueLight, colourLightGrey, colourBlue]
```

```
var colourBlueLight= 'rgba(44, 130, 201, 1)'; // rgba
var colourLightGrey= '#D3D3D3'; // hex code
var colourBlue= 'Blue'; // html color
```

Custom button styling

```
blocks['multigraph_1'] = {
  ...
  buttonsPadX: 10,
  buttonsPadY: 10,
  buttonsBorder: 'red',
  buttonsColor: '#fff',
  buttonsFill: '#000',
  buttonsIcon: 'red',
  buttonsMarginX: 5,
  buttonsMarginY: 5,
  buttonsRadius: 20,
  buttonsShadow: 'rgba(255, 255, 255, 0.1)',
```

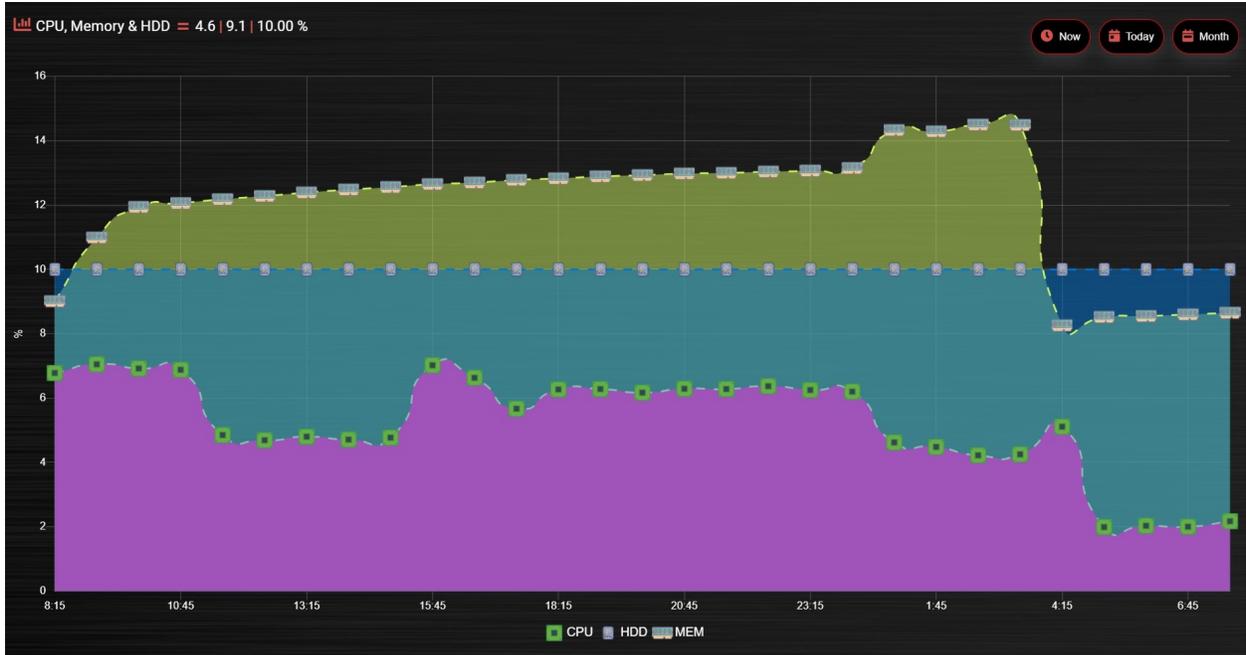
(continues on next page)

(continued from previous page)

```

    buttonsSize: 12,
    ...
}

```



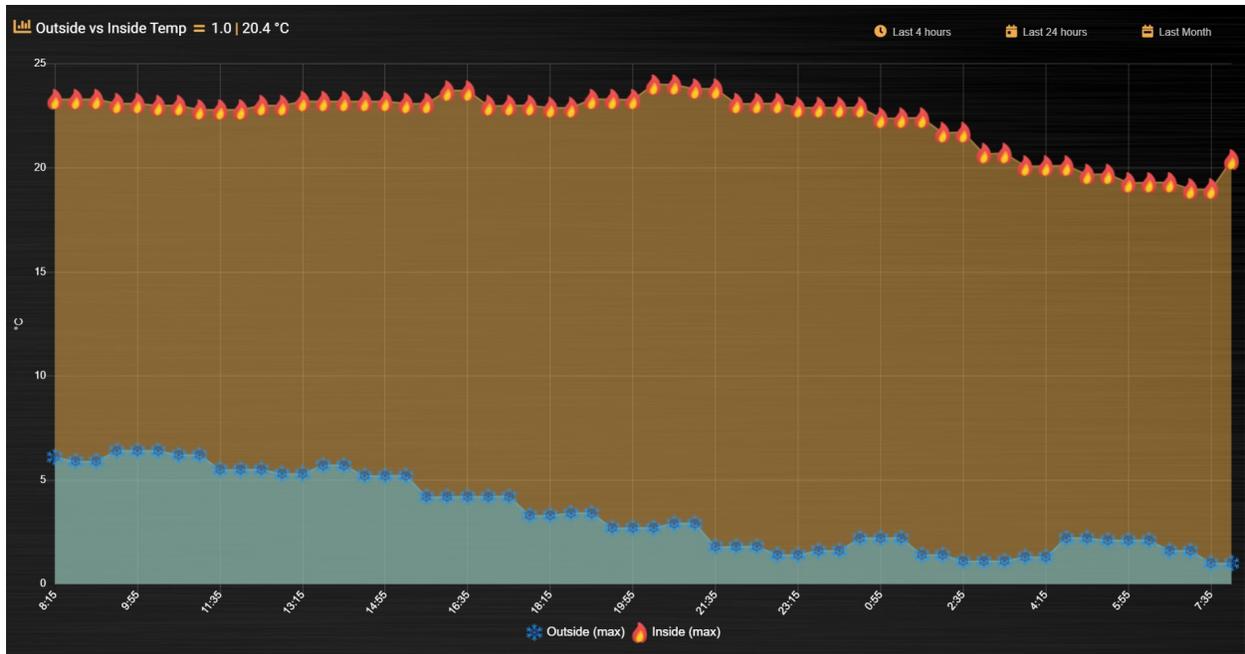
Custom point styling

```

var hot = new Image();
hot.src = "img/hot.png"
var cold = new Image();
cold.src = "img/cold.png"

blocks['multigraph_2'] = {
  ...
  pointStyle: [cold, hot ],
  ...
}

```



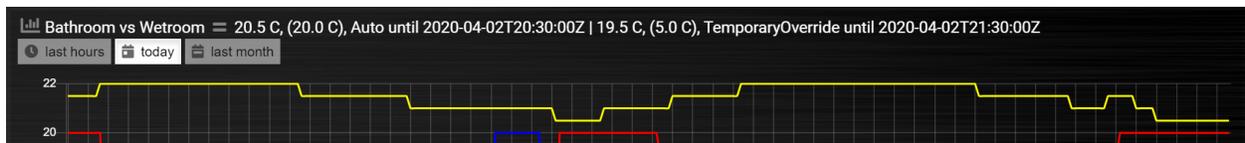
customHeader

The parameter `customheader` can be a:

- string
- function
- object

Examples for each type are presented below.

Example of graph showing long data values in header:



We can now update the block with the `customHeader` object as shown below:

```
blocks['bathroom_wetroom'] = {
  title: 'Bathroom vs Wetroom',
  devices: [12, 13],
  customHeader: {
    12: 'data.split(",").slice(0,2)',           <---- update/
    ↪format the data for idx 12
    13: 'data.split(",").slice(0,2)',           <---- update/
    ↪format the data for idx 13
    x: '(data.12-data.13).toFixed(2) + " C"',   <---- append_
    ↪custom data based on idx 12 and idx 13
  },
  tooltiptotal: true,
  graph: "line",
}
```

(continues on next page)

(continued from previous page)

```

    legend: true
  }

```

The object accepts key value pairs. Standard Javascript or JQuery code can be used, where 'data' is the data you want to change.

- To format/update the data being displayed by a device, use the idx (**number**) as the key.
- To add a new 'custom value' that is a calculation of existing device data, use any **letter** as the key.

Using the updated block (above), the graph now displays like this (below). The unwanted data has been removed, and a new value (the delta between the 2 devices), "1.0 C", has been added:



In case customHeader is a string, string will be evaluated, and the result added to the graph title:

```

customHeader: '"Usage: " + devices[6].Usage + "Delivery: " + devices[6].CurrentDeliv'

```

The Domoticz devices can be accessed via devices[idx], as you can see in the previous example.

In case the formatting and/or computation is more complex, you can define customHeader as function:

```

customHeader : function(graph) {
    var devices = Domoticz.getAllDevices();
    var solarGeneration = devices[6].Usage;
    var inflow = devices[43].Usage;
    var outflow = devices[43].UsageDeliv;
    var nett = inflow + solarGeneration - outflow;
    return "Nett: " + nett + " Watt";
},

```

Custom data

```

blocks['multigraph_72'] = {
  title: 'Outside vs Inside Temp',
  devices: [ 72, 152],
  graph: 'line',
  buttonsBorder: '#ccc',
  buttonsColor: '#ccc',
  buttonsFill: 'transparent',
  buttonsIcon: 'Blue',
  buttonsPadX: 10,
  buttonsPadY: 5,
  buttonsMarginX: 5,
  buttonsMarginY: 2,
  buttonsRadius: 0,
  buttonsShadow: 'rgba(2, 117, 216, 0.2)',
  buttonsSize: 12,
  custom : {
    "Last hours": {
      range: 'day',
      filter: '6 hours',
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

        data: {
            te_72: 'd.te_72',
            te_152: 'd.te_152',
            delta: 'd.te_152-d.te_72'
        },
    },
    "Last 2 weeks": {
        range: 'month',
        filter: '14 days',
        data: {
            te_72: 'd.te_72',
            te_152: 'd.te_152',
            delta: 'd.te_152-d.te_72'
        }
    },
    "Last 6 months": {
        range: 'year',
        filter: '6 months',
        data: {
            te_72: 'd.te_72',
            te_152: 'd.te_152',
            delta: 'd.te_152-d.te_72'
        }
    }
},
legend: {
    'te_72': 'Outside',
    'te_152': 'Inside',
    'delta': 'Difference'
}
}

```

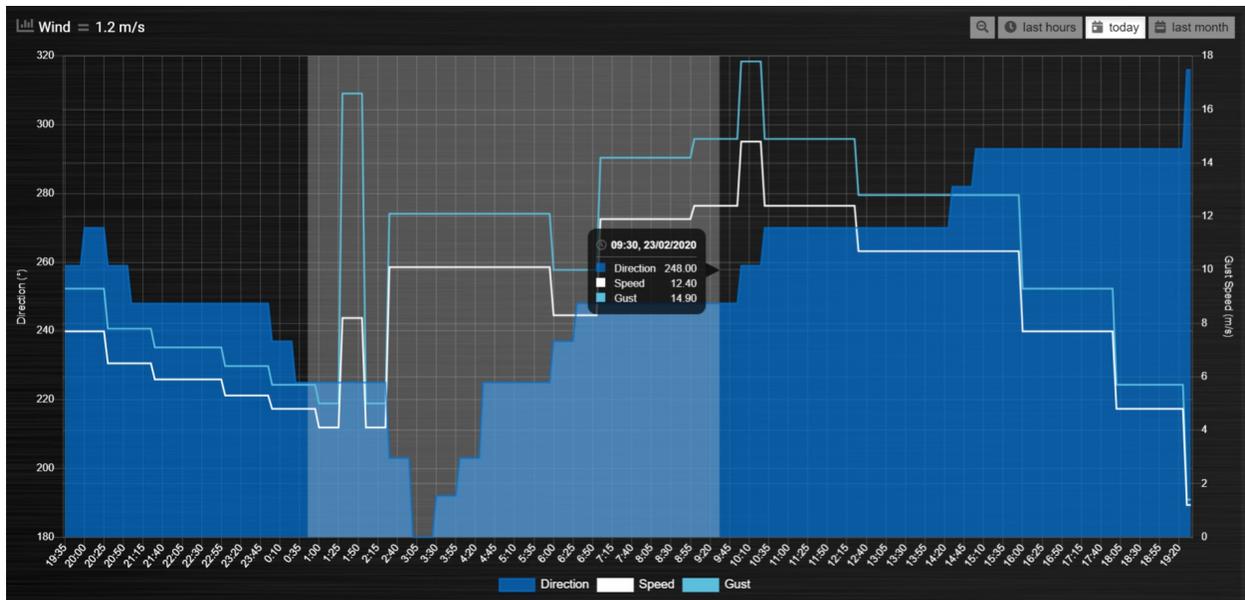


Zoom

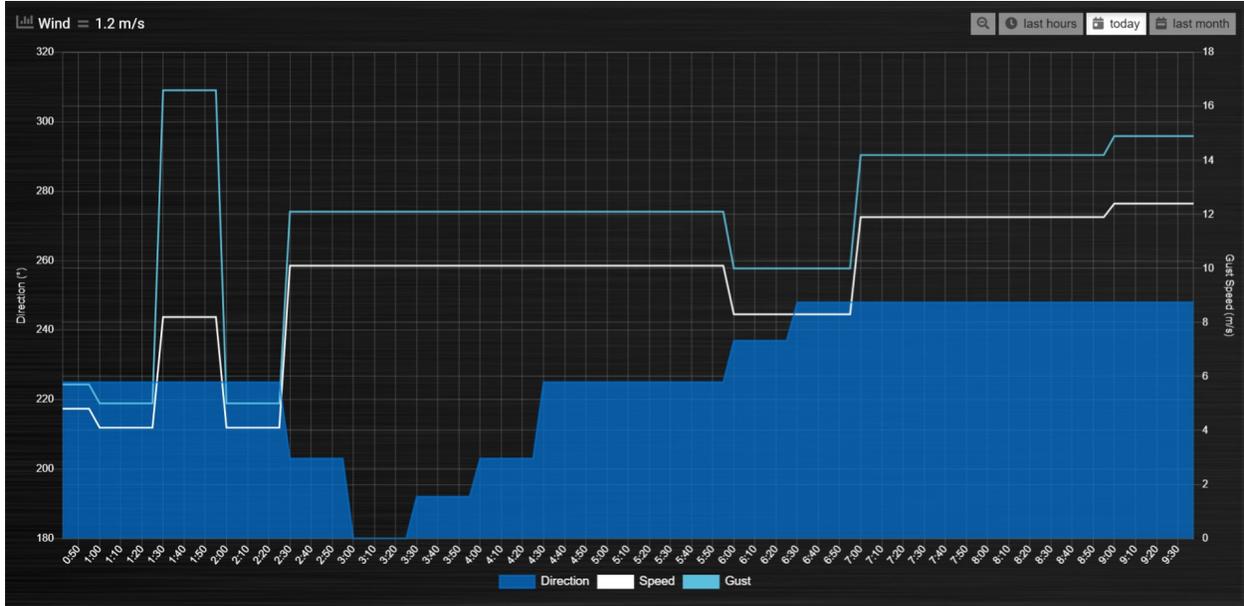
The *zoom* parameter can be set on the graph block as follows:

```
blocks['wind'] = {
  title: 'Wind',
  devices: [73],
  graph: 'line',
  zoom: 'xy',
  legend: {
    'di_73' : 'Direction',
    'sp_73' : 'Speed',
    'gu_73' : 'Gust'
  }
}
```

The “Wind” graph before zoom “x”:



The “Wind” graph after zoom “x”:



GroupBy

The *GroupBy* parameter can be set on the graph block as follows:

```
blocks['group_by_solar'] = {
  title: 'Solar',
  devices: [1],
  graph: ['bar'],
  graphTypes: ['v'],
  groupBy: 'week',
  legend: true
}
```

Alternatively, the param can be applied to custom data as follows:

```
blocks['group_by_solar'] = {
  title: 'Grouped: Solar',
  devices: [1],
  graph: ['bar'],
  graphTypes: ['v'],
  custom : {
    "Day by Hour": {
      range: 'last',
      groupBy: 'hour',
      filter: '24 hours',
      data: {
        Solar: 'd.v_1'
      },
    },
    "Week by Day": {
      range: 'month',
      groupBy: 'day',
      filter: '7 days',
      data: {
        Solar: 'd.v_1',
      },
    },
  },
}
```

(continues on next page)

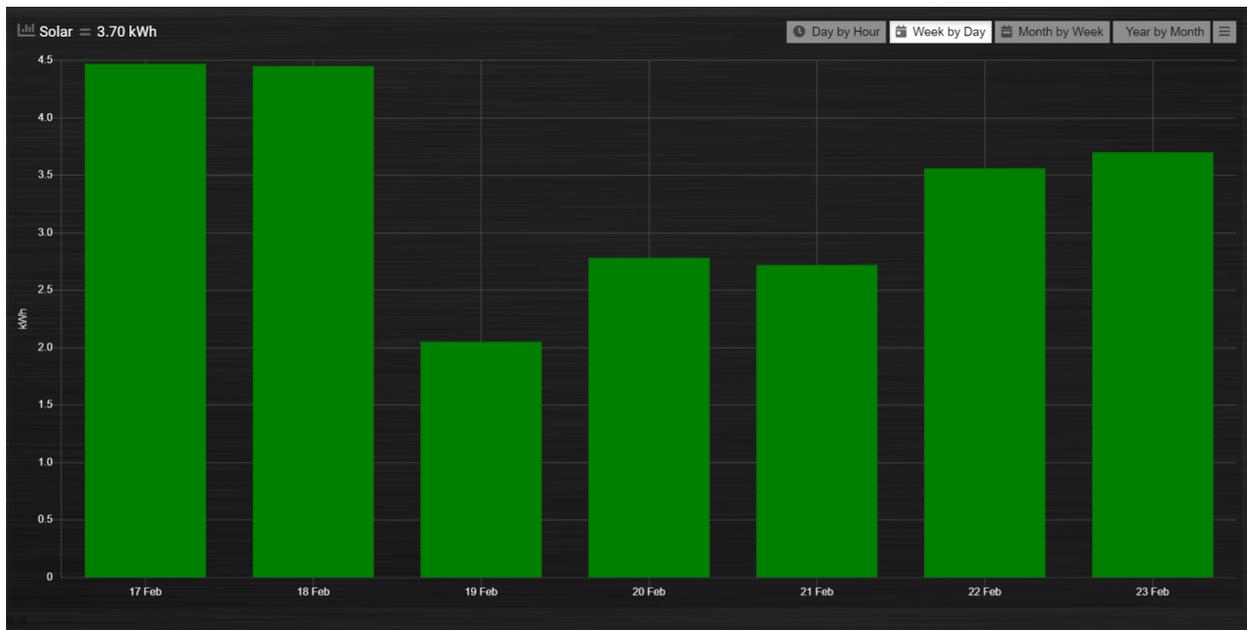
(continued from previous page)

```

    },
    "Month by Week": {
      range: 'month',
      groupBy: 'week',
      data: {
        Solar: 'd.v_1',
      }
    },
    "Year by Month": {
      range: 'year',
      groupBy: 'month',
      data: {
        Solar: 'd.v_1',
      }
    }
  },
  datasetColors: ['green'],
  legend: true
}

```

This results in the “Solar” graph grouping its data by hour, day, week or month - *Week by Day* is shown in the image below:



You can use the block parameter ‘aggregate’ to set the aggregation method as ‘sum’ or ‘avg’ (for average).

Instead of a single string the parameter ‘aggregate’ can also be an array of strings, like

```

aggregate: ['sum', 'avg'],

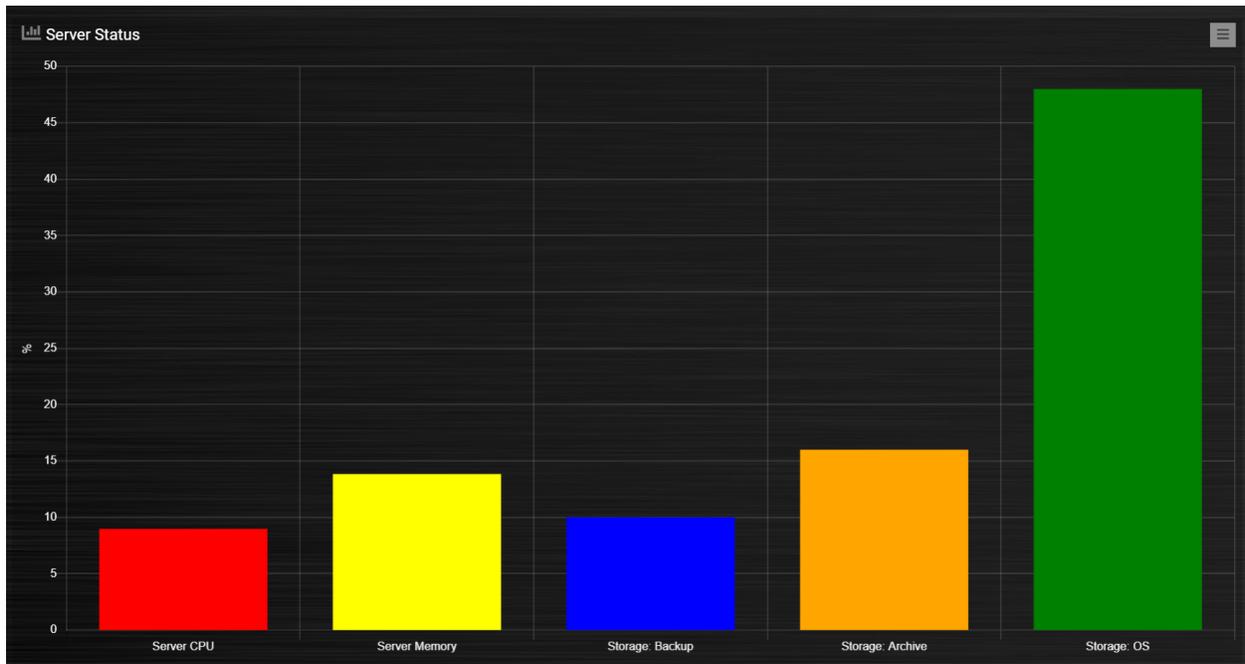
```

This can be useful in case of a custom graph, showing gas usage and temperature in one graph, because in that situation two different aggregation methods are required.

groupByDevice

The block parameter `groupByDevice` is showing the **live** status of several devices in a single graph. Instead of the data being grouped by *time* intervals, it is grouped by the *devices*. Note, unlike other graphs, this type of graph does not report on historic data. I.e. there are no 'last', 'today', 'month' buttons.

```
blocks['server_status'] = {
  title: 'Server Status',
  devices: [17, 18, 189, 190, 192],
  groupByDevice: true,
  beginAtZero: true
}
```

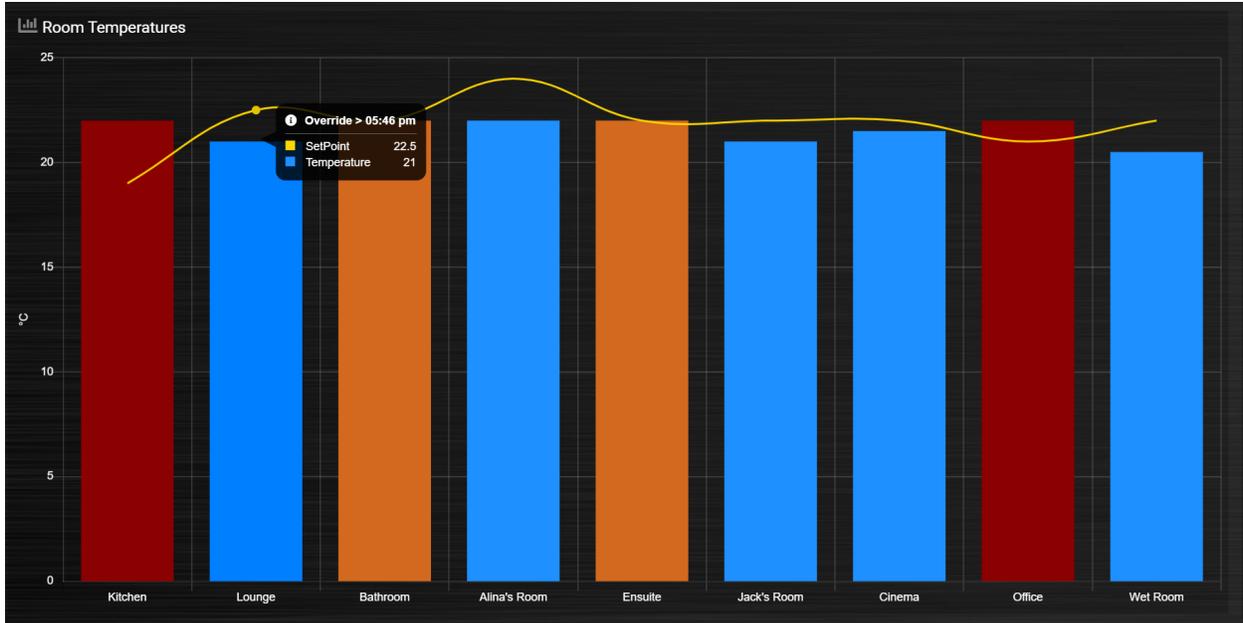


The feature works with device sensors such as counter, percentage and temperature.

With temperature sensors that have setpoints, it calculates whether the device is:

- Cold - blue
- At setpoint - orange
- Hot - red
- It will show a line displaying the SetPoint values in front of the bar graph for thermostat devices that have SetPoint data.
- The tooltip will show the status and schedule with EvoHome devices.
- The block `datasetColors` parameter can now be used to set the colors for 'below temp', 'at temp', 'above temp' and 'setpoint' (in that order).

The office and kitchen rooms are showing red, as the temperature is above the setpoint ...



Setting `groupByDevice` to `'horizontal'` shows like this ...

```
blocks['all_zones'] = {
  title: 'Room Temperatures',
  devices: [6, 11, 12, 8, 14, 9, 15, 235, 10, 13],
  groupByDevice: 'horizontal',
  beginAtZero: true
}
```



stacked

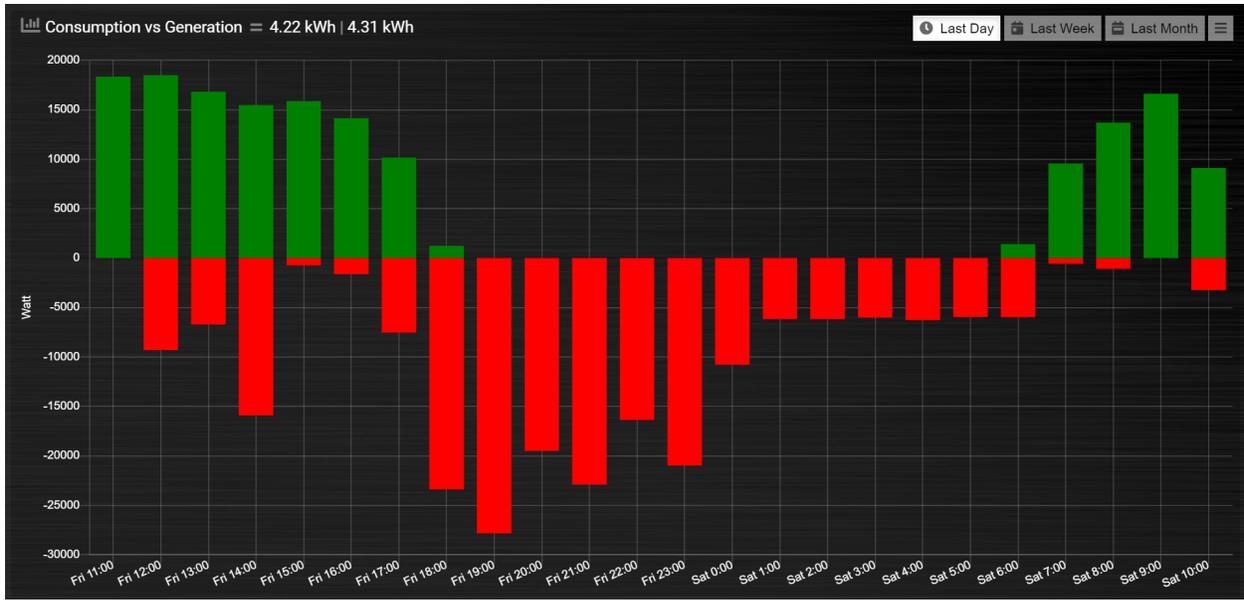
With *stacked: true* parameter graph bars will be stacked. To show the total value of the stacked bars on the tooltip you have to add *tooltiptotal: true* to the graph block.

```
blocks['group_by_solar_vs_grid'] = {
  title: 'Consumption vs Generation',
  devices: [258,1],
  graph: 'bar',
  stacked: true,
  graphTypes: ['v'],
  tooltiptotal: true,
  debugButton: true,
  custom : {
    "Last Day": {
      range: 'last',
      groupBy: 'hour',
      filter: '24 hours',

      data: {
        Generation: 'd.v_1',
        Consumption: 'd.v_258*-1'
      },
    },
    "Last Week": {
      range: 'month',
      groupBy: 'day',
      filter: '7 days',

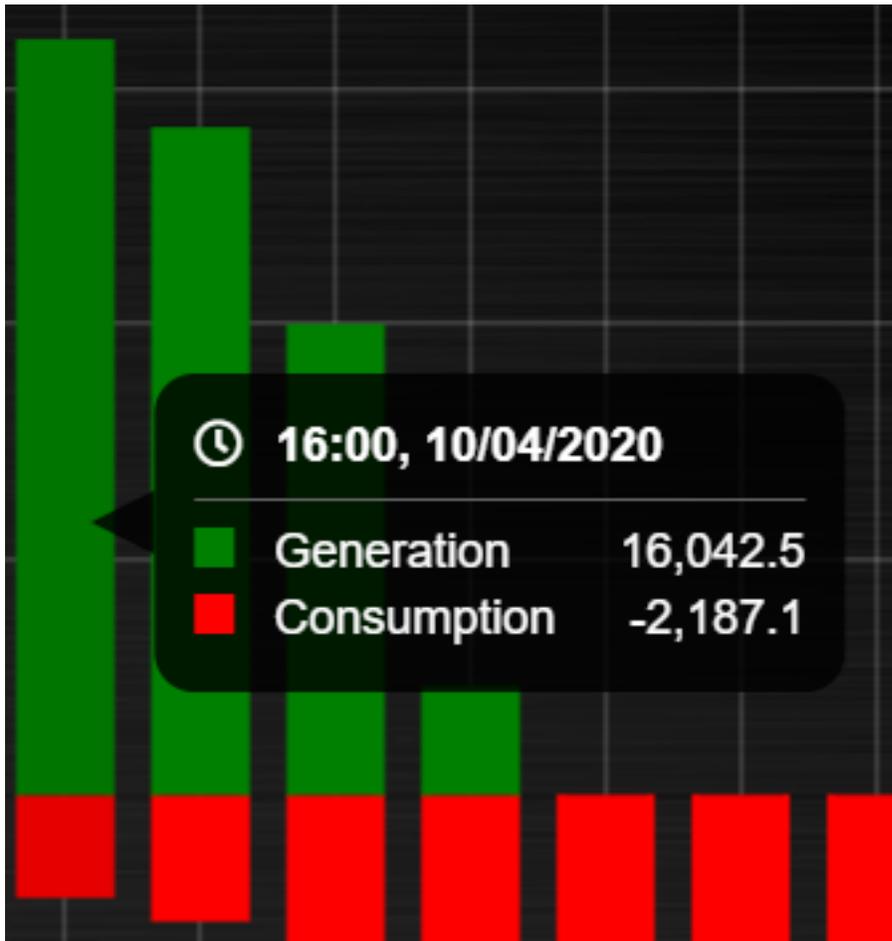
      data: {
        Generation: 'd.v_1',
        Consumption: 'd.v_258*-1'
      },
    },
    "Last Month": {
      range: 'month',
      groupBy: 'week',

      data: {
        Generation: 'd.v_1',
        Consumption: 'd.v_258*-1'
      },
    },
  },
  lineTension: 0.5,
  datasetColors: ['green', 'red']
}
```

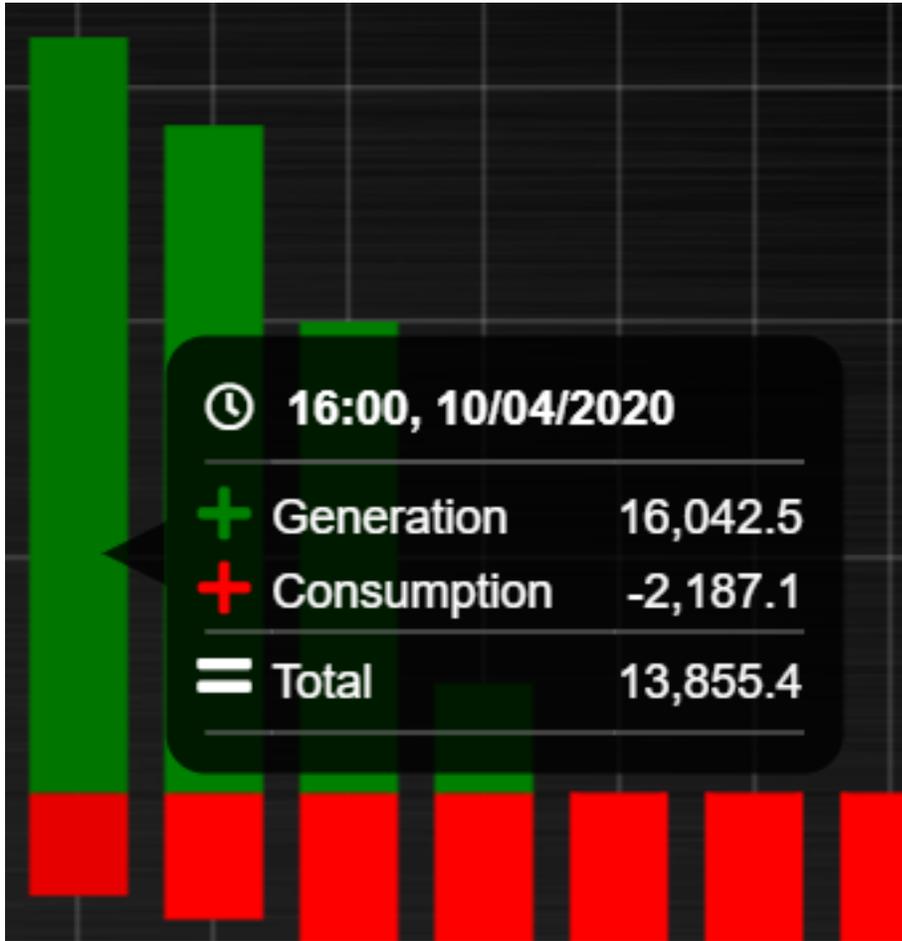


tooltiptotal

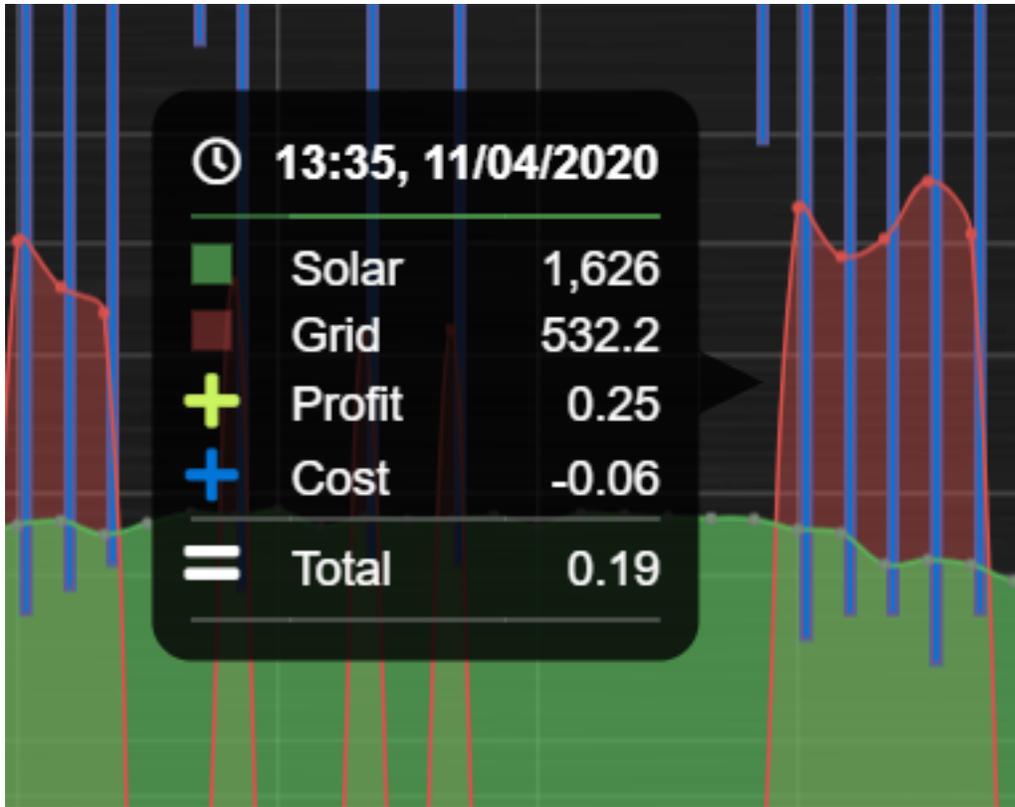
tooltiptotal: false



```
tooltiptotal: true
```



```
tooltiptotal: ['Confirmed (Total)', 'Deaths (Total)']
```



Basically, if you specify an array, it will only total those datasets, and ignore the other ones. Anything that is being totalled will show a “+” icon.

Defined Popups

Popups can be defined by adding a new block parameter, “popup”, to the block that popup is for. This allows the popup to use all the block parameters that a graph block does, allowing them to style the graph however they want. It also means the legend and tooltips can display custom names (instead of the key names). For example, the user has an Energy meter block defined as follows:

```
blocks[258] = {
  title: 'Consumption',
  flash: 500,
  width: 4,
  popup: 'popup_consumption'
}
```

In this example, they have specified that the popup will use a defined graph called ‘popup_consumption’ instead of the default popup. This defined graph is then added to the config.js just like a normal graph:

```
blocks['popup_consumption'] = {
  title: 'Energy Consumption Popup',
  devices: [258],
  tooltipStyle: true,
  datasetColors: ['red', 'yellow'],
  graph: 'line',
  legend: {
    'v_258' : 'Usage',
  }
}
```

(continues on next page)

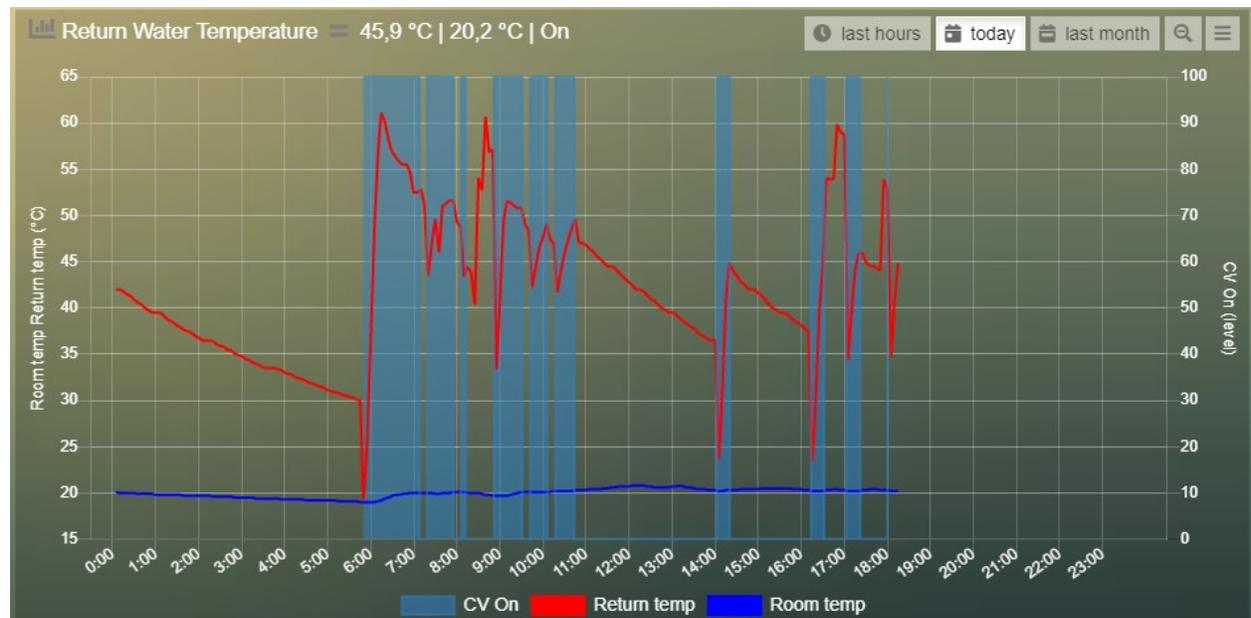
(continued from previous page)

```
'c_258' : 'Total'
}
}
```

Examples

Combine two temperature devices with the switch info indicating central heating is on

```
blocks['switchgraph'] = {
  devices: [
    31,
    27,
    17,
  ],
  debugButton: true,
  zoom: true,
  graph: 'line',
  legend : {
    l_17: 'CV On',
    te_31: 'Return temp',
    te_27: 'Room temp'
  },
  lineFill: [true, false, false],
  datasetColors: ['rgba(44,130,201,0.5)', 'red', 'blue'],
}
```



CPU, Memory & HDD

```
blocks['multigraph_17'] = {
  title: 'CPU, Memory & HDD',
  devices: [ 17, 18, 189 ],
  datasetColors: ['Red', 'Orange', 'Blue', 'Green', 'LightBlue', 'Aqua', 'Yellow
↔', 'Purple', 'Pink'],
}
```

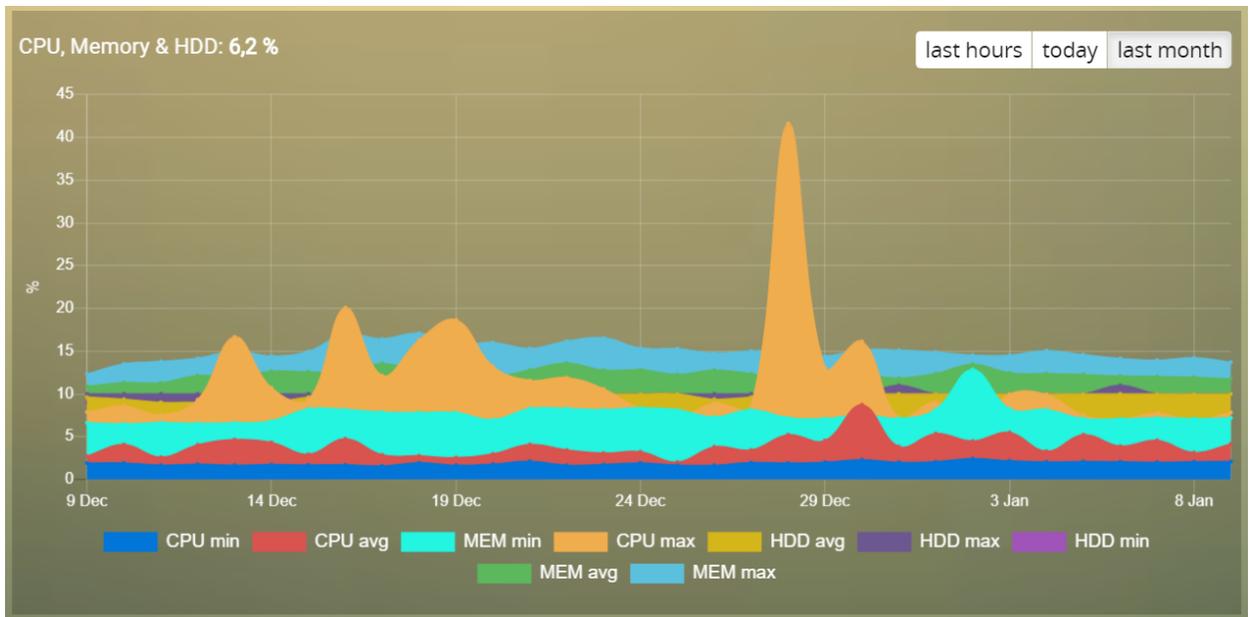
(continues on next page)

(continued from previous page)

```

legend: true,
cartesian : 'linear',
graph: 'line',
lineFill: true,
drawOrderDay: ['v_17', 'v_189', 'v_18'],
drawOrderMonth: ['v_min_17', 'v_avg_17', 'v_min_18', 'v_max_17', 'v_avg_189',
↪ 'v_max_189', 'v_min_189', 'v_avg_18', 'v_max_18'],
legend: {
  'v_17'           : 'CPU',
  'v_avg_17'      : 'CPU avg',
  'v_max_17'      : 'CPU max',
  'v_min_17'      : 'CPU min',
  'v_18'          : 'MEM',
  'v_avg_18'      : 'MEM avg',
  'v_max_18'      : 'MEM max',
  'v_min_18'      : 'MEM min',
  'v_189'         : 'HDD',
  'v_avg_189'     : 'HDD avg',
  'v_max_189'     : 'HDD max',
  'v_min_189'     : 'HDD min'
}
}

```



Grid vs Solar

Due to the low solar output in winter months, comparing solar to grid was often hard to read. The graph needed to be updated to use a logarithmic scale, i.e. a nonlinear scale useful when analysing data with large ranges. The solar device stops recording data at the usual 5 minute intervals when it gets dark. The code inserts intervals (with a value of 0.00) when no data is recorded. In the updated multigraph block below, the *cartesian* property is used, and three *drawOrder* properties.

```

blocks['multigraph_1'] = {
  title: 'Grid vs Solar',
  devices: [ 162, 1],
  datasetColors: ['Red', 'Green'],
}

```

(continues on next page)

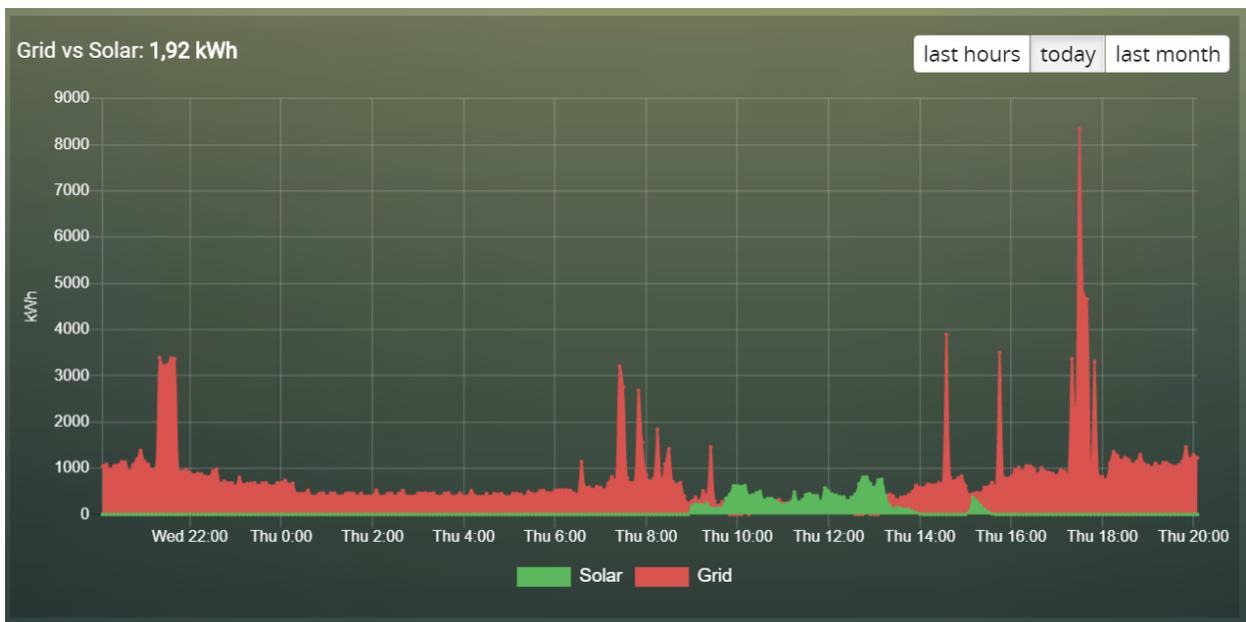
(continued from previous page)

```

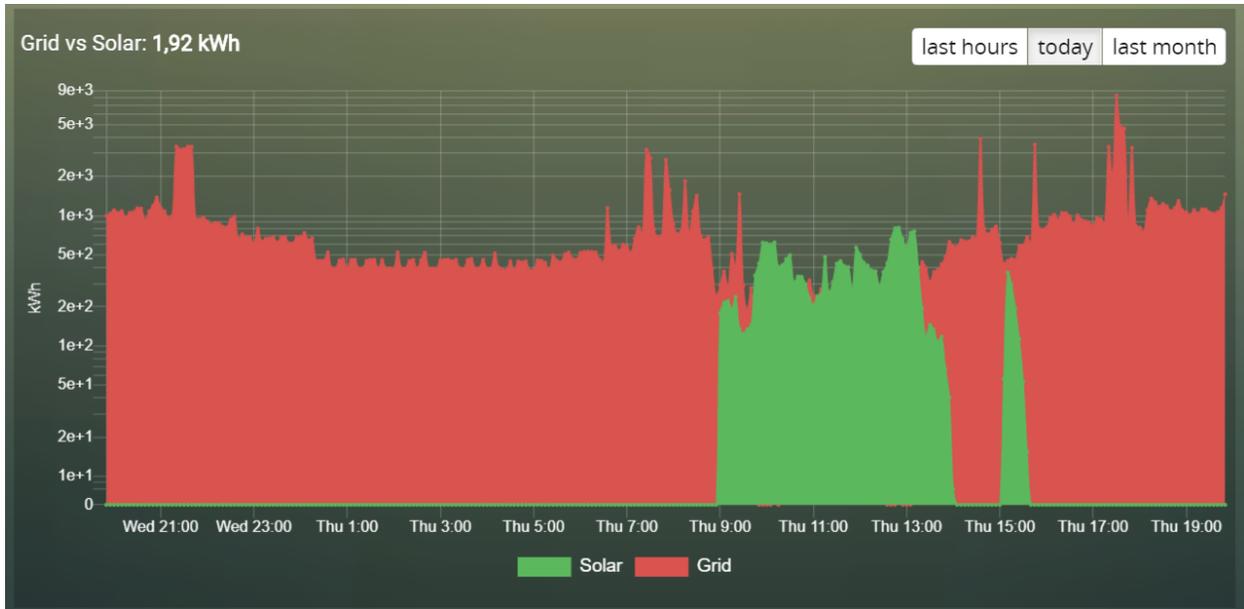
lineFill: [true, true],
graph: 'line',
cartesian: 'logarithmic',
drawOrderLast: ['v_1', 'v_162'],
drawOrderDay: ['v_1', 'v_162'],
drawOrderMonth: ['v_162', 'v_1', 'c_162', 'c_1'],
legend: {
  'v_162': 'Grid',
  'v_1': 'Solar',
  'c_162': 'Solar Cumulative',
  'c_1': 'Solar Cumulative'
}
}

```

This is using the standard *linear* scale (i.e. cartesian = linear):



This is using the new *logarithmic* scale (i.e. cartesian = logarithmic). Note the y axis labelling on the left:



Outside vs Inside Temp

The indoor temp sensor also includes barometric pressure (ba) and humidity (hu), but the outside one is only temperature. In the graph below, the *graphTypes* property is used to remove the extra unwanted data. Now only the temperature is directly compared.

```
blocks['multigraph_72'] = {
  title: 'Outside vs Inside Temp',
  devices: [ 72, 152],
  datasetColors: ['LightBlue', 'LightGrey', 'Blue', 'Orange', 'Red', 'Yellow'],
  graphTypes: ['te', 'ta', 'tm'],
  graph: 'line',
  legend: {
    'te_72': 'Outside (max)',
    'ta_72': 'Outside (avg)',
    'tm_72': 'Outside (min)',
    'te_152': 'Inside (max)',
    'ta_152': 'Inside (avg)',
    'tm_152': 'Inside (min)'
  }
}
```



Temperature and Setpoint

Three thermostat devices (Evohome TRVs), each showing their temperature and setpoint.:

```
blocks['evohome_graphs'] = {
  title: 'Lounge, Kitchen, Hall Thermostats',
  devices: [ 11, 12, 152],
  interval: 2,
  maxTicksLimit: 12,
  datasetColors: ['LightGrey', 'Red', 'Green', 'DarkGreen', 'Blue'],
  buttonsIcon: 'Purple',
  graph: 'line',
  lineTension: 0,
  borderWidth: 2,
  spanGaps: false,
  graphTypes: ['te', 'se'],
  buttonsBorder: '#ccc',
  buttonsColor: '#ccc',
  buttonsFill: 'transparent',
  buttonsIcon: 'Blue',
  buttonsPadX: 10,
  buttonsPadY: 5,
  buttonsMarginX: 5,
  buttonsMarginY: 2,
  buttonsRadius: 0,
  buttonsShadow: 'rgba(2, 117, 216, 0.2)',
  buttonsSize: 12,
  buttonsText: ['6H', '24H', '1M'],
  legend: {
    'se_11': 'Lounge (SP)',
    'sm_11': 'Lounge (SP Min)',
    'sx_11': 'Lounge (SP Max)',
    'te_11': 'Lounge (TE)',
    'ta_11': 'Lounge (TE Avg)',
    'tm_11': 'Lounge (TE Min)',
    'se_12': 'Kitchen (SP)',
    'sm_12': 'Kitchen (SP Min)',
  }
}
```

(continues on next page)

(continued from previous page)

```

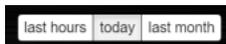
'sx_12': 'Kitchen (SP Max)',
'te_12': 'Kitchen (TE)',
'ta_12': 'Kitchen (TE Avg)',
'tm_12': 'Kitchen (TE Min)',
'se_152': 'Hall (SP)',
'sm_152': 'Hall (SP Min)',
'sx_152': 'Hall (SP Max)',
'te_152': 'Hall (TE)',
'ta_152': 'Hall (TE Avg)',
'tm_152': 'Hall (TE Min)'
}
}

```



Buttons

Standard buttons:

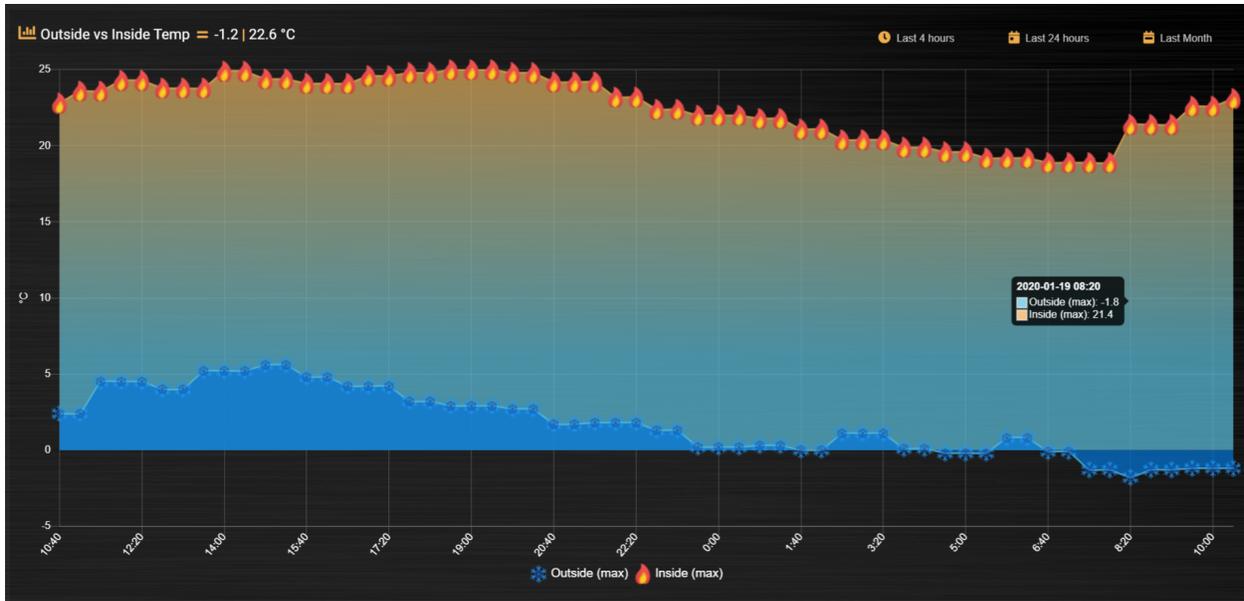


Updated buttons (one of many styles):



More Examples

This graph includes 2 separate *temperature* sensors, with gradients, custom points (images) and button styling:



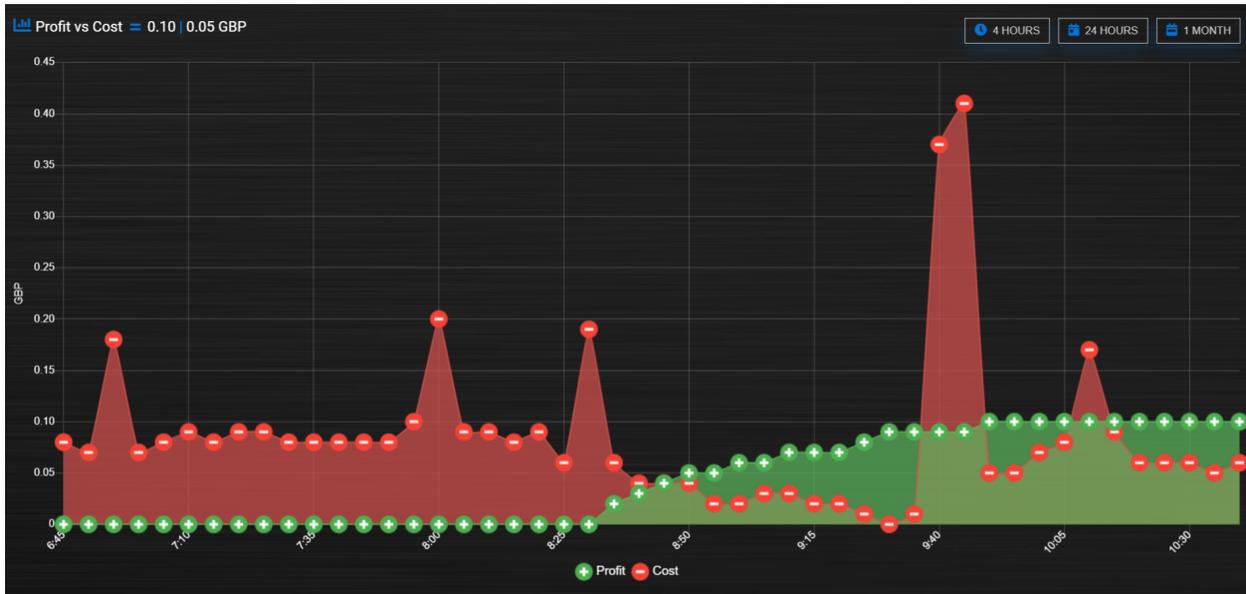
This graph includes 3 separate *percentage* sensors, custom points (images) and button styling:



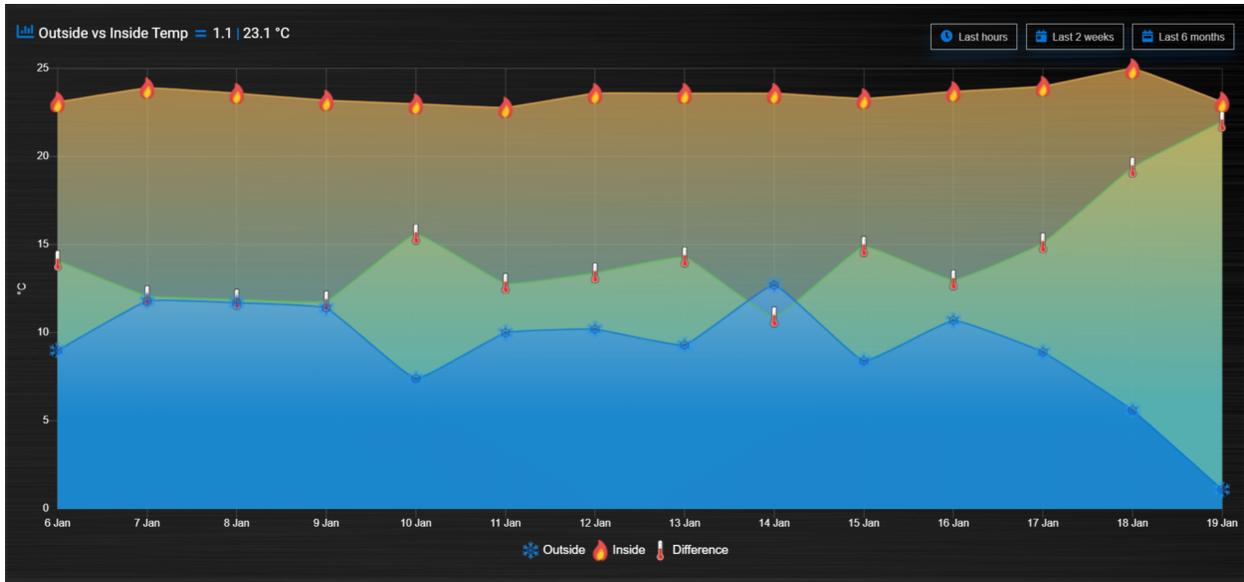
This graph includes 2 separate *energy* sensors, subtle gradients, no points and uses the *logarithmic* scale:



This graph includes 2 separate *counter* sensors, without gradients, but with custom points (images) and button styling:



This graph uses 2 *temperature* sensors **and** *custom data*, calculating a 3rd virtual dataset, showing the difference between the outside temperature and the inside temperature:



Number format

Number formatting is applied to tooltip values and header values.

Number formatting uses two global config parameters:

- `_DECIMAL_POINT`: Default value is `'.'`
- `_THOUSAND_SEPARATOR`: Default value is `','`

You can redefine these two config parameters in `CONFIG.js`:

```
_DECIMAL_POINT = '.';
_THOUSAND_SEPARATOR = ',';
```

For header values the formatting is only applied in case the block parameter `format` is set to `true`, and the device is a single value device. For instance, for a `TempHumBar` device no formatting will be applied to header values. For header values the default number of decimals is derived from the device type. You can overrule the number of decimals with the `decimals` block parameter.

For tooltip values the default number of decimals is the `decimals` value of the first device, which may be overruled by the `decimals` parameter.

The values in a tooltip will always have the same number of decimals.

Styling

For graphs the following css-classes are used:

- `.graphheader`: The graph header, including title and buttons
- `.graphtitle`: The title of the graph, including the current value
- `.graphbuttons`: The buttons for the graph

You can modify the class definition in `custom.css`. If you want to hide the header:

```
.graphheader {
  display: none;
}
```

You can also modify the class for a specific graph only

```
[data-id='mygraph'].graph .graphheader {
  display: none;
}
```

In the previous example only the graph created with key 'mygraph' will be affected.

To change the default size of the graph popup windows add the following style blocks to your custom.css:

```
.graphheight {
  height: 400px;
}

.graphwidth {
  width: 400px;
}
```

To remove the close button of the graph popup add the following text to custom.css:

```
.graphclose { display: none; }
```

To be detailed...

```
.opengraph, .opengraph<idx>p, #opengraph<idx>p //classes attached to the graph_
→popup dialog
.graphcurrent<idx> //class attached to the div with the current value
```

For internal use:

```
block_graph<idx> //The div to which the graph needs to be attached.
#graphoutput<idx> //The canvas for the graph output
```

Styling of X- and Y- axes

(advanced customization...)

By setting the options block parameter most graph properties can be customized. For instance to change the font size of the axes, and reduce the width of the graph legend use the following block definition:

```
blocks['graph_43'] = {
  ...,
  options: {
  scales: {
    xAxes: [{
      ticks: {
        fontSize: 7
      }
    }],
    yAxes: [{
      ticks: {
        fontSize: 7,
```

(continues on next page)

3.2.3 Buttons

Buttons are clickable elements that may show an image. First you have to define the button in `CONFIG.js`:

```
buttons = {}           //only once!!
buttons.yr = {
  width: 12,
  isimage: true,
  refresh: 60,
  btnimage: 'https://www.yr.no/sted/Norge/Oppland/%C3%98ystre_Slidre/Beito/advanced_
↪meteogram.png',
  url: 'https://www.yr.no/sted/Norge/Oppland/%C3%98ystre_Slidre/Beito/
↪langtidsvarsel.html'
};

// Then add the button to a specific column:
var columns = {}      //This line only once!!
columns['3'] = {
  blocks: [
    ...,
    buttons.yr,
    ...
  ],
  width: 1
}
```

Parameters

Parameter	Description
width	1..12: The width of the block relative to the column width
title	'<string>': Custom title for the block
key	'key': unique identifier.
slide	1..99: Slide to specified screen on click.
isimage	Set to true if the image should be shown in the full button width (default false).
icon	'fas fa-icon': icon to show in the button.
image	'image.png': image to show as icon the button. Image path is relative to the <dashticz>/img folder.
btnimage	'<url>': URL of the image to show in the button.
refresh	1..99999: Refresh time of the button image in seconds. There is no maximum. The default is 60 (=1 minute).
url	'<url>': URL of the page to open in a popup window on click.
popup	'mypopup': Opens the 'mypopup' block in a new window on click.
forceheight	Set the height of the image in a button '200px': Set image height to 200px.
framewidth	'<integer>': specific width of the popup window on click.
frameheight	'<integer>': specific height of the popup window on click.
forcerefresh	Control the caching-prevention mechanism of the images and popup frame for a button. 0 : Normal caching behavior (=default) 1, true : Prevent caching by adding t=<timestamp> parameter to the url as second parameter. Try this if you have a (cheap Chinese) webcam. Not all webservers will handle this correctly 2 : The image is loaded via php, preventing caching. (php must be enabled on your Dashticz server) 3 : Prevent caching by adding t=<timestamp> parameter to the end of the url. Not all webservers will handle this correctly.
refreshiframe	0: No automatic refresh of a button popup frame (default) 1..99999: Refresh time of the button popup frame in sec. There is no maximum. The default is 60 (=1 minute).
level	Domoticz log level used by the log-button.
newwindow	0: open in current window 1: open in new window 2: open in new frame (default, to prevent a breaking change in default behavior) 3: no new window/frame (for intent handling, api calls). HTTP get request. 4: no new window/frame (for intent handling, api calls). HTTP post request. (forcerefresh not supported) 5: open in a new browser tab
auto_close	Closes the opened window after a certain period of time (only applicable when newwindow is 1,2 or 5)
82	0: (=Default) No auto close 5: Closes the popup window after 5 seconds.
password	

Usage

Slide button

If you have added the `slide` parameter to a button, then Dashtics will slide to the specific screen if the button is pressed.

If you use a button to slide to specific screen (menu button), then the background color of that button will change if that specific screen is active.

Example: If screen number 2 is the active screen, then a button with parameter `slide:2` will be shown as active.

You can adapt the formatting of the selected button with the class `.selectedbutton` in your `custom.css`.

Example:

```
.selectedbutton {
  background-color: #cba !important;
}
```

Example on how to use menu buttons:

```
//three buttons are defined
buttons.page1 = { width:12, title:'page 1', slide:1};
buttons.page2 = { width:12, title:'page 2', slide:2};
buttons.page3 = { width:12, title:'page 3', slide:3};

//definition of a menu column
var columns = {}           //This line only once!!
columns['menu'] = {
  blocks: [ buttons.page1, buttons.page2, buttons.page3],
  width: 1
}

//Define columns 1 to 6 as well
// ...

//Add the menu column to your screens
var screens = {}          //This line only once!
screens[1] = {
  columns: ['menu', 1,2]
}
screens[2] = {
  columns: ['menu', 3,4]
}
screens[3] = {
  columns: ['menu', 5,6]
}
```

forcerefresh

Control the caching-prevention mechanism of the images for a button.

0 : Normal caching behavior (=default)

1 (or true) : Prevent caching by adding `t=<timestamp>` parameter to the url. Not all webservers will handle this correctly

2 : The image is loaded via php, preventing caching. (php must be enabled on your Dashticz server)

Examples

Additional examples of button definitions:

```
var buttons = {}
buttons.buienradar = {width:12, isimage:true, refresh:60, btnimage: 'https://image.
↳buienradar.nl/2.0/image/animation/RadarMapRainNL?height=300&width=360&extension=gif&
↳renderBackground=True&renderBranding=False&renderText=True&history=3&forecast=6&
↳skip=1', url: 'https://www.buienalarm.nl/amsterdam-noord-holland-nederland/52.3727,
↳4.8936'}
buttons.radio = {width:12, image: 'radio_on.png', title: 'Radio', url: 'http://
↳nederland.fm'}
buttons.nunl = {width:12, icon: 'far fa-newspaper', title: 'Nu.nl', url: 'http://www.
↳nu.nl'}
buttons.webcam = {width:12, isimage:true, refresh:2, btnimage: 'http://ip_url_to_
↳webcam', url: 'http://ip_url_to_webcam', framewidth:500, frameheight:400}
```

To remove the close button of the button-popup add the following text to custom.css:

```
.frameclose { display: none; }
```

3.2.4 Frames

With a frame it's possible to show (internet) content directly on the Dashticz dashboard.

A frame is defined as follows:

```
//////////////////// FRAMES //////////////////////
var frames = {}
frames.weather = {
  frameurl:"//forecast.io/embed/#lat=49.2624&lon=-123.1155&name=Vancouver&color=
↳#00aaff&font=Helvetica&fontColor=#ffffff&units=si&text-color=fff",
  height: 250 //height of the block in pixels
}
```

In the previous example the weather forecast for Vancouver from forecast.io is downloaded.

You can add a frame to a column in the usual way:

```
//Definition of columns
columns = {}

columns[1] = {
  blocks: [frames.weather],
  width: 6
}
```

Frame parameters

Parameter	Description
frameurl	The URL to load in the frame
width	1 . . 12 The width of the frame relative to the column width. 12=100%, 3=25%
height	Height of the frame in pixels
scrollbars	false Scrollbars are never shown (even if they are needed) Default: auto
refresh	Refresh interval (in seconds)
forcerefresh	Control the caching-prevention mechanism of the frame content 0 : Normal caching behavior (=default) 1, true : Prevent caching by adding t=<timestamp> parameter to the url. Not all websites will handle this correctly 2 : The frame content is loaded via php, preventing caching. (php must be enabled on your Dashticz server)
scaletofit	Width of the frame content in pixels, which will be scaled to the block width. 256 : Assuming the width of the frame content is 256 pixels, then the frame will be scaled in such a way that the content will fit the block width.
aspectratio	The height of the block will be adjusted to a certain aspect ratio, taking scaling via scaletofit into account.

Example

```
var config = {}
config['language'] = 'nl_NL'; //or: en_US, de_DE, fr_FR, hu_HU, it_IT, pt_PT, sv_SV
config['domoticz_ip'] = 'http://192.168.178.18:8080'; //Replace with your Domoticz_
↳IP:Portnumber
config['domoticz_refresh'] = '5';
config['dashticz_refresh'] = '60';

config['auto_swipe_back_after'] = '0';

//Definition of blocks
blocks = {}

blocks['weather'] = {
  frameurl:"//forecast.io/embed/#lat=49.2624&lon=-123.1155&name=Vancouver&color=
↳#00aaff&font=Helvetica&fontColor=#ffffff&units=si&text-color=fff&",
  height: 250
}

//Definition of columns
columns = {}

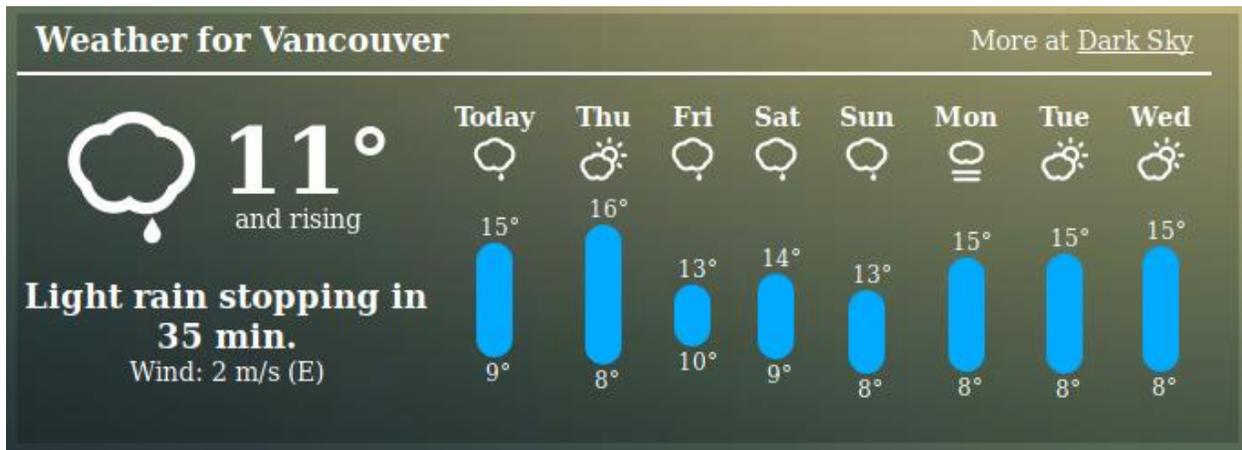
columns[1] = {
  blocks: ['weather'],
  width: 6
}
```

(continues on next page)

(continued from previous page)

```
//Definition of screens
screens = {}
screens[1] = {
  columns: [1]
}
```

This will give the following result:



In some cases the frame content doesn't fit well within the block width. The frame content can be scaled by using the block parameters `scaletofit` and `aspectratio`.

For instance, the `buienradar` widget has a frame width of 256 pixels, and an aspect ratio of 1:1. That means you can define the block as follows:

```
blocks['buien'] = {
  frameurl: 'https://gadgets.buienradar.nl/gadget/zoommap/?lat=52.37403&lng=4.88969&
  ↪overname=2&zoom=11&naam=amsterdam&size=2&voor=0',
  scrollbars: false,
  width:12,
  scaletofit: 256,
  aspectratio: 1,
}
```



In the example above the ‘buien’ block has been added to columns of width 1,2 and 4 respectively.

For other buienradar widgets check the following url: <https://www.buienradar.nl/overbuienradar/gratis-weerdata>

Look for the iframe examples and copy/paste the content of the ‘src’ parameter to the frameurl of your Dashticz block.

3.2.5 Specials

Alarmmeldingen

This module can display (Dutch) 112 Meldingen.

Define your “alarmmeldingen” block (optional):

```
blocks['alarmmeldingen'] = {
  rss: 'https://alarmeringen.nl/feeds/city/venlo.rss',
  filter: 'Venlo', //filter the messages for your town and if you want the
  ↪cities around you
  show_lastupdate: true,
  width: 12,
  refresh: 300, //refresh rate in seconds
  results: 5 //number of recent results to show
}
```

Add “alarmmeldingen” to a column:

```
columns[1]['blocks'] = [
  'alarmmeldingen',
```

Use the site <https://alarmeringen.nl> to find the rss-feed of your choice (region, city)



Parameters

Parameter	Description
title	'<string>': Custom title for the block
rss	The rss url for your City or Region from https://www.alarmeringen.nl/
filter	Add search filters for surrounding cities or special messages mentioned in the alarmmessage like Traumaheli or Brandweer
show_lastupdate	true / false: To display the ime of the last update of the data displayed
width	The block width
refresh	The update interval of this block in seconds
icon	The icon to show in the block, if you dont want to show an icon use ''
image	The image to use instead of an icon. Location is relative to ./img
results	The number of most recent results found by using the filter parameter
timeformat	Time format template. Default is 'ddd d MMM HH:MM' which will display the time as 'Thu 4 Apr 22:03'.

Usage

Note: Alarmmeldingen requires beta 3.4.0 or higher.

If you want to remove the icon, add the following to your block definition:

```
icon: '', //This are two tick-marks
```

To use your own image instead of a FontAwesome Free icon you can add this to your "CONFIG.js":

```
image: '../custom/img/siren.png',
```

For more information on time formats see: <https://momentjscom.readthedocs.io/en/latest/moment/04-displaying/01-format>

Styling

To change the color of the alarmmessages add the following to your `custom.css`:

```
.alarmrow{
font-size: 14px !important;
}
```

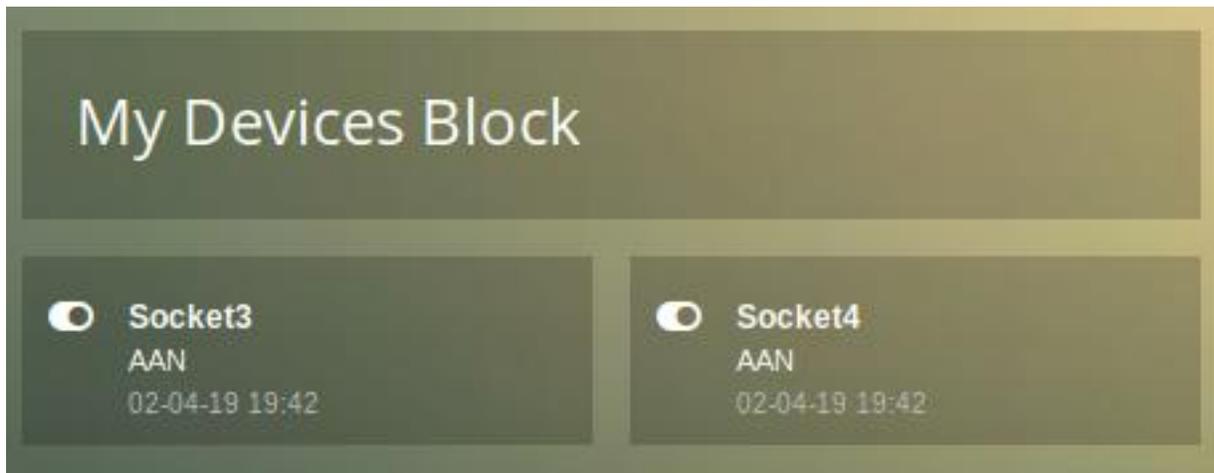
(continues on next page)

(continued from previous page)

```
.alarmrow a {
color: white;
}
.alarmrow strong{
display: inline-block;
}
```

In case no info is available then the CSS class `empty` will be added to block.

Blocktitle



A special block type is a block title. You define a block title as follows:

```
blocks['blocktitle_1'] = { // 'blocktitle_1' must be an unique name
  type: 'blocktitle', //Set type to 'blocktitle' (required for block title)
  title: 'My Devices Block', //The title of the block as shown in the
  ↪ dashboard.
  width: 6, //The width of the block relative to the column width
  icon: 'far fa-lightbulb', //If you want to show an icon, choose from: https://
  ↪ fontawesome.com/icons?d=gallery&m=free
  image: 'lightbulb.png' //If you want to show an image instead if icon, place
  ↪ image in img/ folder
}
```

Block parameters

Parameter	Description
width	1..12: The width of the block relative to the column width
title	'<string>': Custom title for the block
icon	Defines the icon for this block, choose from: https://fontawesome.com/icons?d=gallery&m=free 'fas fa-eye'
image	If you want to show an image instead of an icon, place image in img/ folder 'bulb_off.png'
type	Set this parameter to 'blocktitle'

Note: From 3.7.3 onwards only the 'new calendar' block is supported.

New Calendar

You can add a block with upcoming events from your ical-calendar (gmail, apple etc.).

Note: PHP support is required. See system preparation.

Dashticz config settings

The following config settings are applicable:

Parameter	Description
calendarformat	Configure the Calendar Date/Time format. 'dd DD.MM HH:mm' = default
calendarlanguage	Controls the calendar locale 'nl', 'en', 'hu', etc.

Usage

Google Calendar

You have to know the correct link to your Google Calendar. You can find them as follows:

- Open <https://calendar.google.com/calendar>
- Under 'My calendars' click on the three dots behind your calendar -> settings and sharing
- In the page that opens look for the following links:
 - Public URL to this calendar. It's something like: `https://calendar.google.com/calendar/embed?src=yourname%40gmail.com&ctz=Europe%2FAMsterdam`
Use this public url as url parameter in your calendar block.
 - Secret address in ICAL format. It's something like: `https://calendar.google.com/calendar/ical/yourname%40gmail.com/private-5045b31.....ba/basic.ics`
Use this ical url as icalurl parameter in your calendar block.

The **new calendar** block follows the same look and feel as most other blocks. It uses `type` to tell Dashticz that its a calendar block. The **new calendar** block can be configured as follows:

```
blocks['my_calendar'] = {
  type: 'calendar',
  maxitems: 5,
  layout: 0,
  url: 'https://calendar.google.com/calendar/embed?src=_REDACTED_&ctz=Europe
↪%2FLondon',
  icalurl: 'https://calendar.google.com/calendar/ical/_REDACTED_/private-
↪123456789/basic.ics',
  holidayurl: 'https://www.calendarlabs.com/ical-calendar/ics/75/UK_Holidays.ics
↪',
  weeks: 5,
  lastweek: false,
  isoweek: false,
  width: 12
}
```

Parameter	Description
emptytext	The text to show in case there are no appointments. Default: 'Geen afspraken'
calFormat	The parameter "calFormat" is obsolete. The "calFormat" parameter for the new calendar is called layout
layout	<p>0: Lists an agenda of events in text format</p> <p>1: Lists an agenda of events in table format</p> <p>2: Displays a traditional calendar in table format</p> <p>3: Lists an agenda of all day events in text format (no time is shown)</p> <p>4: Lists an agenda of all day events in table format (no time is shown)</p>
icalurl	This can accept either a single url (string) or one or more calendar objects (example below)
ics	Url (string) of the calendar object
color	Color of the calendar text. Must be <i>html colors</i> , <i>hex code</i> , <i>rgb</i> or <i>rgba string</i>
url	This can be set on the block or in settings['calendarurl']. Whenever you click the calendar block whilst setting layout 0 or 1, an embedded gmail calendar will display in a popup dialog (modal). Alternatively, when layout 2 is set, when clicking on any event, it will display a popup with the event details and provide a link to the calendar (e.g. gmail)
holidayurl	This allows users to add public holidays (or other public events) to their calendar
maxitems	This limits the number of events that you want to display. When setting layout 2, I set it to 100 to allow for 35 days. Adjust to your own preference
weeks	This is how many weeks, or rows of 7 days, you wish to display when layout 2 is selected
lastweek	<p>Show the previous week and any events from that week</p> <p>true: Show the previous week</p> <p>false (=default): Don't show the previous week</p>
isoweek	<p>The week will start on a Sunday or on a Monday</p> <p>true: Week will start on a Monday</p> <p>false (=default): Week will start on a Sunday</p>
icon	<p>Icon name. Example:</p> <p>'fas fa-car'. To display a car icon in the left column</p>
image	<p>If you want to show an image instead of an icon in the left column. Place image in <code>img/</code> folder</p> <p>'calendar.png'</p>
title	<p>A title will be shown above the calendar</p> <p>'<string>': Title for the block</p>
width	1..12: The width of the block relative to the column width
method	<p>0: ical method 0 (recommended. Default when PHP version < 7.1)</p> <p>1: ical method 1 (default when PHP version >= 7.1)</p>

Notes

I did not find an ical library that handles all ical files correctly.

ical method 0 has issues with yearly recurring events.

ical method 1 has issues for recurring events with interval more than 1 (for instance biweekly events)

Example of traditional calendar in table format

```
blocks['gmail_calendars'] = {
  type: 'calendar',
  layout: 2,
  icalurl: {
    Personal: {
      ics: 'https://calendar.google.com/calendar/ical/_REDACTED_/
↳private-123456789/basic.ics',
      color: 'blue'
    },
    Business: {
      ics: 'https://calendar.google.com/calendar/ical/_REDACTED_/
↳private-123456789/basic.ics',
      color: 'purple'
    }
  },
  holidayurl: 'https://www.calendarlabs.com/ical-calendar/ics/75/UK_Holidays.ics
↳',
  maxitems: 100,
  weeks: 5,
  lastweek: true,
  isoweek: false,
  width: 12
}
```

The layout set to 0 will display this:

```
Thu, Apr 09 - Entire day - Testing System
Thu, Apr 09 - 11:00 -11:30 - Cutover Preparation
Thu, Apr 09 - 18:00 -18:30 - Put bins out
Fri, Apr 10 - 11:00 -11:30 - Cutover Preparation
Sun, Apr 12 - 10:00 -11:00 - Timesheet
```

The layout set to 1 will display this:

```
Thu, Apr 09  Entire day  Testing System
              11:00 - 11:30  Cutover Preparation
              18:00 - 18:30  Put bins out
Fri, Apr 10  11:00 - 11:30  Cutover Preparation
Sun, Apr 12  10:00 - 11:00  Timesheet
```

The layout set to 2 will display this:

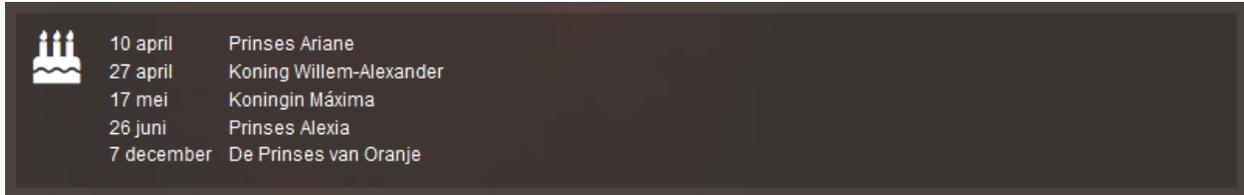
Sun 29 Mar	Mon 30 Mar	Tue 31 Mar	Wed 01 Apr	Thu 02 Apr	Fri 03 Apr	Sat 04 Apr
				18:00 Put bins out		10:00 Fix shower rail 14:00 Teamcenter Drop
Sun 05 Apr 10:00 Teamcenter Drop 17:00 Test Dashticz	Mon 06 Apr 10:30 Sunrise call 13:30 Order shopping 14:00 TMA weekly call	Tue 07 Apr Fix computer 11:00 Cutover Preparation 14:30 System Administration	Wed 08 Apr 11:00 Cutover Preparation 13:00 Status Review Meeting	Thu 09 Apr Testing System 11:00 Cutover Preparation 18:00 Put bins out	Fri 10 Apr Good Friday 11:00 Cutover Preparation	Sat 11 Apr
Sun 12 Apr 10:00 Timesheet	Mon 13 Apr Easter Monday 11:00 Cutover Preparation	Tue 14 Apr 11:00 Cutover Preparation	Wed 15 Apr 11:00 Cutover Preparation	Thu 16 Apr 11:00 Cutover Preparation 18:00 Put bins out	Fri 17 Apr 11:00 Cutover Preparation	Sat 18 Apr 10:00 LVT 4.4 Testing
Sun 19 Apr 10:00 Timesheet	Mon 20 Apr 11:00 Cutover Preparation	Tue 21 Apr 11:00 Cutover Preparation	Wed 22 Apr 11:00 Cutover Preparation	Thu 23 Apr 11:00 Cutover Preparation 18:00 Put bins out	Fri 24 Apr 11:00 Cutover Preparation	Sat 25 Apr
Sun 26 Apr 10:00 Timesheet	Mon 27 Apr 11:00 Cutover Preparation	Tue 28 Apr 11:00 Cutover Preparation	Wed 29 Apr 11:00 Cutover Preparation	Thu 30 Apr 11:00 Cutover Preparation 18:00 Put bins out	Fri 01 May 11:00 Cutover Preparation	Sat 02 May

When the user clicks on any events, it opens details about that event. If the event details is already HTML, it will render the HTML event body, including font, tags, anchors/links, etc. The contents of the popup is scrollable. Also included in the popup is a link to source calendar (bottom left), if one has been set in config.js. On the bottom right of the popup, the event location is displayed (if this exists). When clicked, it will take the user to the location on Google maps.

Tue 07 Apr Fix computer 11:00 Cutover Preparation 14:30 System Administration	Wed 08 Apr 11:00 Cutover Preparation 13:00 Status Review Meeting	Thu 09 Apr Testing System 11:00 Cutover Preparation 18:00 Put bins out
<div style="border: 1px solid #ccc; padding: 10px; background-color: #f9f9f9;"> <p>Fix computer ✕</p> <p>Things to consider:The peripherals on your computer are going to be pretty specific to your needs, so you'll need to think about whether you'll need them, and just how many of them you want. You may want ten USB ports but not have a care in the world about having a DVD drive. On the other hand, you may do everything online and just want as few holes in your machine as possible for soda to spill into and frazzle the circuitry.</p> <p>USB: There is, thankfully, a standard that most computer peripherals opt to use, and it can be for anything from mouses and keyboards to hard drives and monitors – heck, you can even plug a guitar in via USB if you've found the right cord. A modern version of USB, called USB 3.0, is faster than its predecessors, but there's an even newer version of USB, called USB Type-C, which offers gigabits of bandwidth and the ability to handle enough current to power a laptop. While USB Type-C will eventually become the single technology that you'll use to connect all your devices, and is a good thing to have on a new machine, you'll need to avoid off-brand cables that can fry your brand-new computer.</p> <p>If you plan to transfer a lot of data, make sure you have a fast USB port – or a fast wireless network. An alternative for some major externals (like CD drives and hard drives) is to get a computer with an eSATA port, which will let you plug in those peripherals on the fly with high data transfer rates.</p> <p>HDMI: If you're going to use your machine for entertainment, you'll probably want an HDMI output. This will allow you to connect it to most modern televisions for a high-quality visual display, and it will also run the audio out if you're planning to use the TV for sound.</p> <p>Launch full calendar 32 London Bridge St, London SE1 9SG, UK</p> </div>		
11:00 Cutover Preparation	11:00 Cutover Preparation	11:00 Cutover Preparation 18:00 Put bins out

Example of a birthday calendar

When using layout : 3 or layout : 4 no time will be shown.



```
blocks['birthdays'] = {
  type: 'calendar',
  layout: 4,
  dateFormat: 'D MMMM',
  icalurl: 'http://... ..../birthdays.ics'
  maxitems: 5,
  icon: 'fas fa-birthday-cake',
}
```

Event styling

You can customize the event styling by setting the `eventClasses` block parameter. As an example block definition:

```
blocks['f1'] = {
  maxitems: 8,
  icalurl: 'http://www.f1calendar.com/download/f1-calendar_p1_p2_p3_q_gp_alarm-20.
  ↪ics',
  layout: 1,
  icon: 'fas fa-car',
  title: 'Formula 1',
  type: 'calendar',
  startonly: true,
  lastweek: false,
  eventClasses: {
    important: 'Grand',
    normal: 'Qual'
  }
}
```

If the calendar text contains the text ‘Grand’, then the CSS class `important` will be added to the event element. If the calendar text contains the text ‘Qual’, then the CSS class `normal` will be added to the event element.

Add the following to `custom.css`:

```
.event.normal {
  color: orange !important
}

.event.important {
  background-color:yellow;
  color: green !important;
}

.description.important {
  color: black !important;
}

.eventtime.important {
```

(continues on next page)

(continued from previous page)

```
color: red !important;
}
```

By default each event uses the following CSS classes:

- event: Event container
- eventdate: Date part of event
- eventtime: Time part of event
- description: Event description

By combining these classes with the classes assigned via `eventClass` you can customize the styling, as shown in the `custom.css` example above.

Then another trick: The values in the `eventClasses` object are not just strings, but Regular Expressions. Assume we define `eventClasses` as follows:

```
eventClasses: {
  important: '^Grand',
  normal: /qual/i
}
```

The style `important` will be applied to events that **start** with 'Grand'.

The matching of 'qual' now is case insensitive.



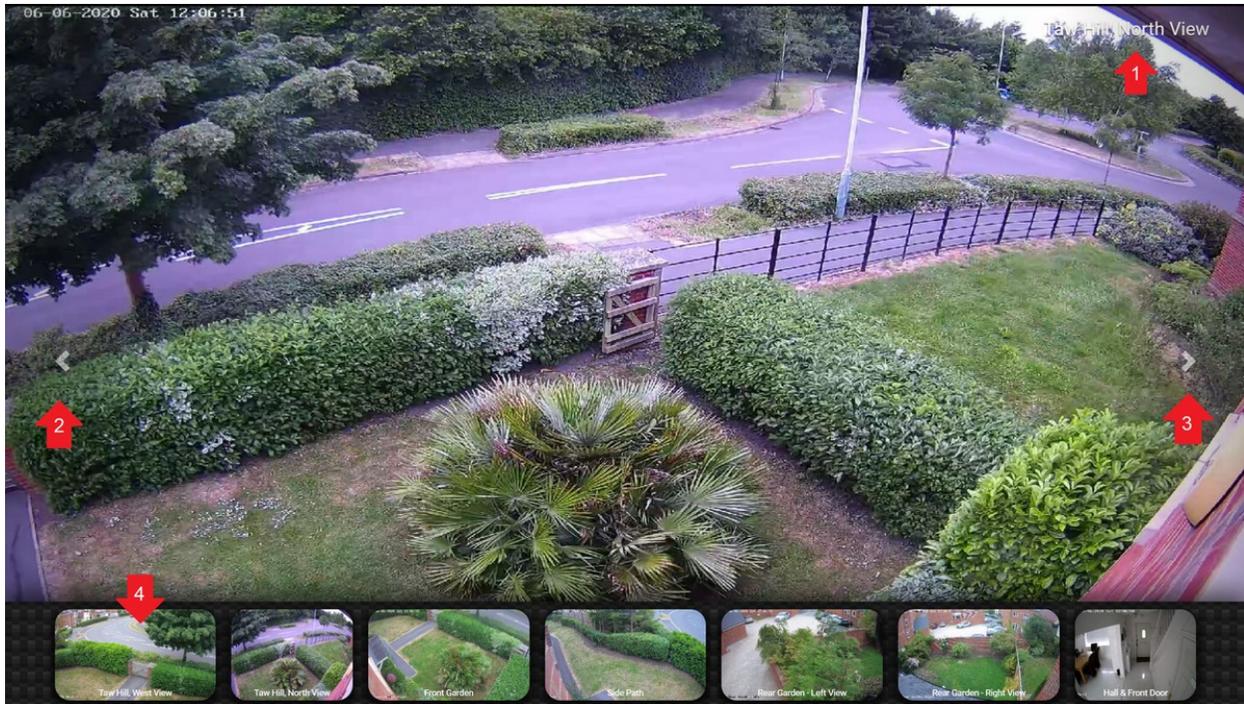
🚗 Formula 1			
vr 03.12	13:30	Free Practice 1 (Saudi Arabian Grand Prix)	
	17:00	Free Practice 2 (Saudi Arabian Grand Prix)	
za 04.12	14:00	Free Practice 3 (Saudi Arabian Grand Prix)	
	17:00	Qualifying (Saudi Arabian Grand Prix)	
zo 05.12	18:30	Grand Prix (Saudi Arabian Grand Prix)	
vr 10.12	10:30	Free Practice 1 (Abu Dhabi Grand Prix)	
	14:00	Free Practice 2 (Abu Dhabi Grand Prix)	
za 11.12	11:00	Free Practice 3 (Abu Dhabi Grand Prix)	

Block styling

In case no info is available then the CSS class `empty` will be added to block. This can be used to adjust the styling of an empty block via `custom.css`

Cameras

This has been designed mainly for those users with several cameras, which provide both an image stream **and** a video stream.



1. Each camera can display the title on the top right of the screen and in the tray
2. You can select the left icon to navigate left on the camera carousel
3. You can select the right icon to navigate right on the camera carousel
4. The images in the tray now refresh every n seconds (block refresh parameter)

```
blocks["cameras"] = {
  type: "camera",
  cameras: [
    {
      title: "Taw Hill, West View",
      imageUrl: "http://192.168.1.123:4567/videoQW.mjpg?image",
      videoUrl: "http://192.168.1.123:4567/videoQW.mjpg?video",
    },
    {
      ...
    },
    {
      title: "Hall & Front Door",
      imageUrl: "http://192.168.1.123:4567/videoHL.mjpg?image",
      videoUrl: "http://192.168.1.123:4567/videoHL.mjpg?video",
    },
  ],
  width: 6,
  height: 250,
  refresh: 0.5,
  traytimeout: 3,
  slidedelay: 3,
  forcerefresh: 1,
};
```

Camera parameters

Parameter	Description
type	camera: identifies this block as a camera (mandatory)
cameras	[...]: optional array to add multiple cameras
imageUrl	<url>: this is the url for the static image of the camera
videoUrl	<url>: this is the url for the fullscreen live video stream. The videoUrl parameter should only be set if it is a Motion JPEG (MJPEG) camera stream
title	<string>: display the name of the camera in the top right of camera stream
refresh	<number>: seconds to refresh the image
traytimeout	<number>: seconds to keep the camera tray open (default = 5)
slidedelay	<number>: seconds before sliding to the next camera (0 = no slide, default = 3)
forcerefresh	0: caching-prevention mechanism of the images (default = 1) See <i>Buttons</i>

Usage

Example of a single camera block:

```
blocks['garage_cam'] = {
  type: 'camera',
  imageUrl: 'http://192.168.1.234:5678?res=640x480&snapshot=1',
  videoUrl: 'http://192.168.1.234:5678?res=1920x1080&fps=15',
  refresh: 1,
  width: 6,
  height: 300
}
```

Camera troubleshooting

There are several challenges with camera's:

1. Find the right url
2. Solve authorization (username/password)

Finding the right url

For the imageUrl parameter Dashticz expects a still image, although a mjpeg video will also work. For the videoUrl parameter Dashticz expects a video in MJPEG format.

If the video URL is not provided, then Dashticz will use the imageUrl instead.

For most camera's the correct URL can be found via the following page:

<https://www.ispyconnect.com/>

Some cheap camera's only provide a RTSP link. RTSP is not supported in most browsers, and cannot be displayed via Dashticz. If you only have a RTSP link you have to use a third-party application for transcoding. See below.

Solve authorization

Dashticz only supports authorization if the username/password is part of the URL string. (something like <http://192.168.1.20:81/videostream.cgi?user=username&pwd=password&resolution=32&rate=0>)

Basic-auth encoded in the ip address is not supported, since this is blocked by most browsers. (<http://username:password@192.168.1.20:81/videostream.cgi?resolution=32&rate=0>)

Also Basic-auth by setting the authorization http request header is not supported.

Solve authorization via Domoticz

If your camera provides a jpeg and/or mjpeg url, but requires basic authorization, then you can request the image/video via Domoticz. Add the camera to Domoticz and use the url for the image/video as provided by Domoticz.

For more info see the Domoticz wiki:

https://www.domoticz.com/wiki/Camera_Setup

Third party video conversion

If your camera only provides a RTSP stream, then the stream needs recoding into JPEG images and a MJPEG video stream. Users reported success with the following tools:

- Motioneye <https://github.com/ccrisan/motioneye/wiki>
- Xeoma <https://felenasoft.com/xeoma/en/>

For Motioneye a Docker image exists, which works very well. Read: <https://github.com/ccrisan/motioneye/wiki/Install-In-Docker>

I'm considering to (optionally) add Motioneye to the Dashticz autoinstall script. If this would be usefull, leave a message in the Dashticz forum.

Clocks

Station Clock



This is an 'old fashioned' station clock. <http://www.3quarks.com/en/StationClock>

You can add the station clock to a column with:

```
columns[1]['blocks'] = ['stationclock'];
```

Block parameter

Parameter	Description
size	Size of the stationclock in pixels. The default size of the station clock is the column width. 200 The clock will have a width and height of 200 pixels
scale	Scale factor for the width of the clock. Should be smaller than 1 0.75: Scales the clock down to 75% (default 1 = 100%).
body	clock body (Uhrgehäuse)
dial	stroke dial (Zifferblatt)
hourhand	clock hour hand (Stundenzeiger)
minutehand	clock minute hand (Minutenzeiger)
secondhand	clock second hand (Sekundenzeiger)
boss	clock boss (Zeigerabdeckung)
minutehandbehavior	minute hand behavior
secondhandbehavior	second hand behavior

The value for all the configuration parameters can be found in the code block below:

```
// clock body (Uhrgehäuse)
StationClock.NoBody      = 0;
StationClock.SmallWhiteBody = 1;
StationClock.RoundBody   = 2;
StationClock.RoundGreenBody = 3;
StationClock.SquareBody   = 4;
StationClock.ViennaBody   = 5;

// stroke dial (Zifferblatt)
StationClock.NoDial      = 0;
StationClock.GermanHourStrokeDial = 1;
StationClock.GermanStrokeDial   = 2;
StationClock.AustriaStrokeDial  = 3;
StationClock.SwissStrokeDial    = 4;
StationClock.ViennaStrokeDial   = 5;

//clock hour hand (Stundenzeiger)
StationClock.PointedHourHand = 1;
StationClock.BarHourHand     = 2;
StationClock.SwissHourHand   = 3;
StationClock.ViennaHourHand  = 4;

//clock minute hand (Minutenzeiger)
StationClock.PointedMinuteHand = 1;
StationClock.BarMinuteHand     = 2;
StationClock.SwissMinuteHand   = 3;
StationClock.ViennaMinuteHand  = 4;
```

(continues on next page)

(continued from previous page)

```
//clock second hand (Sekundenzeiger)
StationClock.NoSecondHand      = 0;
StationClock.BarSecondHand     = 1;
StationClock.HoleShapedSecondHand = 2;
StationClock.NewHoleShapedSecondHand = 3;
StationClock.SwissSecondHand   = 4;

// clock boss (Zeigerabdeckung)
StationClock.NoBoss            = 0;
StationClock.BlackBoss        = 1;
StationClock.RedBoss          = 2;
StationClock.ViennaBoss       = 3;

// minute hand behavoir
StationClock.CreepingMinuteHand = 0;
StationClock.BouncingMinuteHand = 1;
StationClock.ElasticBouncingMinuteHand = 2;

// second hand behavoir
StationClock.CreepingSecondHand = 0;
StationClock.BouncingSecondHand = 1;
StationClock.ElasticBouncingSecondHand = 2;
StationClock.OverhastySecondHand = 3;
```

Config settings

To maintain backwards compatibility the station clock defaults can be set with the following config settings:

Settings	Description
boss_stationclock	shows hands red axis cover 'RedBoss' 'NoBoss' 'BlackBoss' 'RedBoss' 'ViennaBoss'
hide_seconds_stationclock	true Hides second hand false Default. Show second hand.

Basic clock



This is the standard clock. You can add this clock to a column with:

```
columns[1]['blocks'] = ['basicclock'];
```

Or define a custom block as follows:

```
blocks['myclock'] = {
    type: 'basicclock'
}

columns[1]['blocks'] = ['myclock'];
```

Block parameter

Parameter	Description
width	block width 1 .. 12: (default 12).
size	Size of the basic clock in pixels. The default size of the basic clock is the column width. 200 The clock will have a width of 200 pixels
scale	Scale factor for the width of the clock. Should be smaller than 1 0.75: Scales the clock down to 75% (default 1 = 100%).

Flipclock



You can add the flipclock to a column with:

```
columns[1]['blocks'] = ['flipclock'];
```

Or define a custom block as follows:

```
blocks['myclock'] = {
    type: 'flipclock'
}

columns[1]['blocks'] = ['myclock'];
```

Block parameter

Parameter	Description
width	block width 1 .. 12: (default 12).
size	Size of the flipclock in pixels. The default size of the flip clock is the column width. 200 The clock will have a width of 200 pixels
scale	Scale factor for the width of the clock. Should be smaller than 1 0 . 75: Scales the clock down to 75% (default 1 = 100%).
showSeconds	true: (=default) Show seconds false: Hide seconds
clockFace	24: 24 hour clock 12: 12 hour clock

Hayman clock



Clock by Emily Hayman. Design based off: <https://dribbble.com/shots/2271565-Day-095-Time-is-Money>

You can add the Hayman clock to a column with:

```
columns[1]['blocks'] = ['haymanclock'];
```

Or define a custom block as follows:

```
blocks['myclock'] = {
  type: 'haymanclock'
}

columns[1]['blocks'] = ['myclock'];
```

Block parameter

Parameter	Description
width	block width 1 .. 12: (default 12).
size	Size of the Hayman clock in pixels. The default size of the Hayman clock is the column width. 200 The clock will have a width of 200 pixels
scale	Scale factor for the width of the clock. Should be smaller than 1 0 . 75: Scales the clock down to 75% (default 1 = 100%).

Miniclock

You can add the miniclock to a column with:

```
columns[1]['blocks'] = ['miniclock'];
```

Usage

The clock types `dtclock`, `stationclock` and `flipclock` are responsive, meaning they will adapt the size to the block width.

Example code for the several clocks:

```
blocks['stationclock'] = {
  width: 3,
};
blocks['stationclock2'] = {
  type: 'stationclock',
  width: 3,
  boss: 'NoBoss',
  body: 4,
  secondhand: 0
};
blocks['stationclock3'] = {
  type: 'stationclock',
  width: 3,
  body: 0,
  dial: 0,
  secondhand: 1
};
blocks['stationclock4'] = {
  type: 'stationclock',
  width: 3,
  body: 3,
  dial: 1,
  boss: 'ViennaBoss',
  secondhandbehavior: 2
};
blocks['stationclock5'] = {
  type: 'stationclock',
  width: 3,
  boss: 'RedBoss' // 'RedBoss' 'NoBoss' 'BlackBoss' 'RedBoss' 'ViennaBoss'
};

blocks['clock'] = {
  width: 4
}

blocks['flipclock'] = {
  width: 8
}

blocks['miniclock'] = {
  width: 4
}

var columns = {};

columns[1] = {};
columns[1]['blocks'] = [
  'stationclock',
  'stationclock2',
  'stationclock3',
  'stationclock4',
  'clock',
```

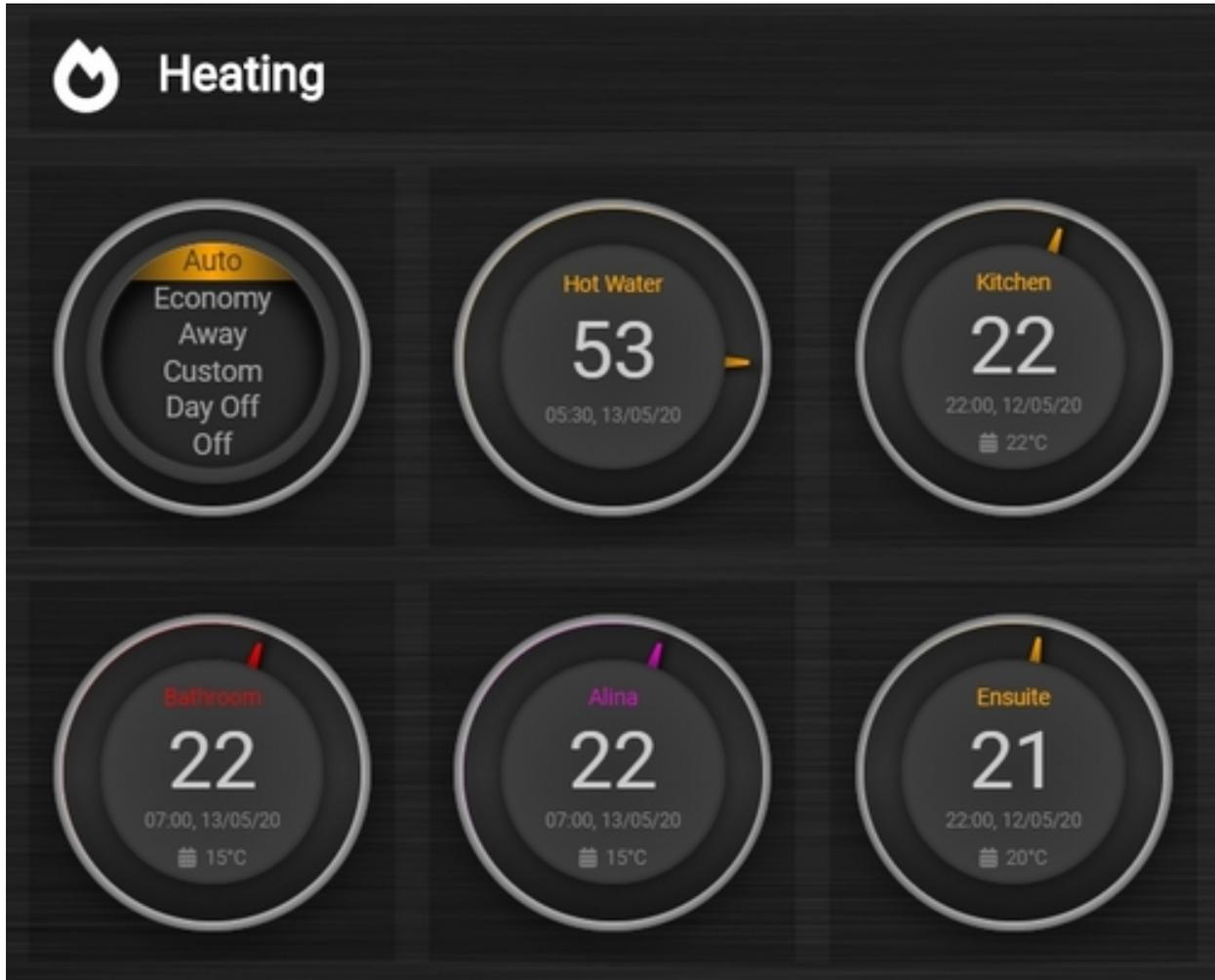
(continues on next page)

(continued from previous page)

```
'flipclock',  
'miniclock',  
];
```



Dial



Most (all?) Domoticz devices can be represented with a dial.

To represent these devices with a dial add `type: 'dial'` to the block definition:

```
blocks['my thermostat'] = {
  type: 'dial',           //Display as dial
  idx: 123,              //The Domoticz device id
  title: 'Device name', //The title of the block as shown in the dial.
  width: 6,             //The width of the block relative to the column width
}
```

For the following device types a specific dial representation has been defined:

- Type = 'Heating'
- Type = 'P1 Smart Meter'
- Type = 'Temp + Humidity + Baro'
- Type = 'Temp + Humidity'
- Type = 'Thermostat'
- Type = 'Wind'

- SubType = 'Evohome'
- SubType = 'SetPoint'
- SubType = 'Text'
- SwitchType = 'Dimmer'
- SwitchType = 'On/Off'
- SwitchType = 'Selector'

For other device types a generic dial will be used.

Block parameters

Parameter	Description
idx	<idx>: IDX of the device (mandatory if named block)
title	'custom_title': Title that will appear on the dial false: No title will be shown true: The device name will be used as title
type	'dial': Identifies this block as a dial (mandatory)
width	1..12: Dial width (optional, default 3)
height	<number>: Dial height (optional, default based on width)
color	'<string>': Color theme for the dial (default orange). Must be <i>html color</i> , <i>hex code</i> , <i>rgb</i> or <i>rgba string</i>
last_update	true: Shows last update info (default: true)
flash	true: Outer dial will flash with user or default color (default: 0)
dialimage	'img/image.png': Show an image instead of the calendar icon (default: false)
dialicon	'fas fa-icon-alt': Show a different font awesome icon (default: 'fas fa-calendar-alt')
iconSwitch	The icon to use for an on/off dial switch. 'fas fa-power-off': Default icon 'fas fa-volume-up': Show a volume icon
showring	true: Always show the outer color ring (default: false)
fixed	true: Removes the needle and numbers around the dial (default: false)
inverted	Invert the value of Up/Down dials. See <i>Up-down dials</i> false: Default for dimmers and Blinds Inverted true: Default for regular Blinds
min	<number>: Minimum value for the dial ring (if applicable) (default: 0)
max	<number>: Maximum value for the dial ring (if applicable) (default: 0)
showunit	false true: Show unit behind value (if applicable) (default: false)
value	The name of the device field to show as main value (only for default dials. Default: 'Data')
values	Used to configure the values to be shown below the main dial value. See <i>Dial values</i>)
animation	false true: Set to false to disable dial animations on change (default: true)
switchMode	The switch mode for on/off dial switches and for dials without device. 'Toggle': Toggle the dial on click (=default for most dials. See next lines for exceptions) 'On': Switch On (=default for scenes and Push On switches) 'Off': Switch Off (=default for Push Off switches)
decimals	The number of decimals to show for numbers. Default is 1. For humidity, barometer it's 0. 1: Numbers will be shown with one decimal
showvalue	true (=default). Show the main device value. false: Don't show the main device value.

3.2. Blocks

109

splitdial	Normally the dial ring color will color from the 0 value to the actual value, which can be positive or negative. Set this parameter to false to start coloring the dial ring from the minimum value, also for a negative minimum value.
-----------	---

Usage

Dimmer

You can use the dial just like a dimmer slider. Click on the dial to switch the dimmer on/off.



```
blocks["bathroom_lights"] = {  
  idx: 439,  
  title: "Bathroom",  
  type: "dial",  
  color: "#57c4d6",  
  width: 2,  
}
```

On/Off Switch

Any devices with this switchtype and type: 'dial' will automatically render as a dial button.

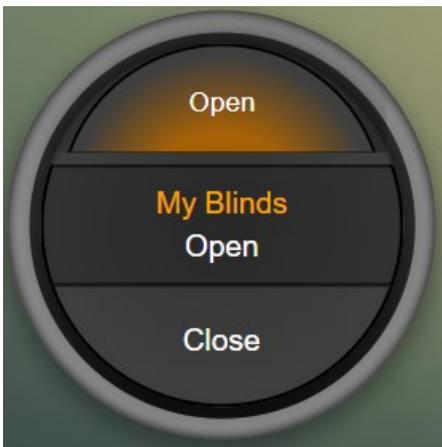


```
blocks['kitchen_lights'] = {
  idx: 451,
  title: 'Kitchen',
  type: 'dial',
  color: '#57c4d6',
  width: 2
}
```

Blinds

All four Domoticz blinds types can be rendered as dial:

- Blinds
- Blinds Percentage
- Blinds Inverted
- Blinds Inverted Percentage



The text in the up and down buttons can be configured via the block parameters `textOpen` and `textClose` respectively.

Temp + Humidity

Will display temperature as the main value and humidity as extra info below. There is enough room to display last_update with this dial.



```
blocks['temp_hum'] = {
  idx: 435,
  title: 'Weather 1',
  type: 'dial',
  setpoint: 15, // this value will be used to control the color of the outer ring,
  ↪e.g. < 15 is blue, >= 15 is orange
  min: -10, // set the minimum value for the dial range (default is 5)
  max: 40, // set the maximum value for the dial range (default is 35)
  width: 2,
  shownumbers: true, // display the numbers on the dial (default is false)
  showing: true, // display outer ring color all the time (default is false, will
  ↪only display when hover over)
  showunit: true // display unit for the dial value (default is false)
}
```

Temp + Humidity + Baro

Similar to above, but with Baro as extra info too. Last_update can be added but it is a tight fit.



```
blocks['temp_hum_baro'] = {
  idx: 72,
  title: 'Weather 2',
  type: 'dial',
  setpoint: 15,
  min: -10,
  max: 40,
  width: 2,
  /* dialicon: ['fas fa-thermometer-half', 'fas fa-arrow-down'], */ // dial icons
  ↪array when for dials have more than 1 extra info
  /* dialimage: ['volumio.png', 'air.png'], */ // dial images array when for
  ↪dials have more than 1 extra info
  showunit: true,
  shownumbers: true,
  last_update: false // disabling last update to allow for more room
}
```

Wind

This dial has a 360 degree range (like a compass). The wind direction can be set to point to where the wind is blowing from or to, by using the new “offset” parameter. Below I have set the dial to point to which direction the wind is blowing.



```
blocks['wind'] = {
  idx: 73,
  title: 'Wind',
  type: 'dial',
  setpoint: 18, // the entire outer ring will change color based on this setpoint,
  ↳ factoring in the current temperature (default 15)
  offset: 180, // 0 will point to the wind source, 180 will point to wind
  ↳ direction (default is 0)
  width: 2,
  showring: true,
  showunit: true,
  shownumbers: true,
  last_update: false
}
```

In case you want to use the wind speed as needle position instead of the wind direction, add the following block parameter:

```
subtype: 'windspeed'
```

P1 Smart Meter

Currently this is configured to use the “Today” counters; CounterDelivToday and CounterToday, i.e. production vs consumption. Unlike any other dial, zero is at “12 o’clock” (instead of the traditional dial which starts at “7 o’clock”).

Today’s energy consumption is more than production



Today's energy production is more than consumption



```
blocks['p1'] = {
  idx: 454,
  title: 'P1 Meter',
  type: 'dial',
  width: 2,
  min: -10,
  max: 10,
  showing: true,
  showunit: true,
  shownumbers: true,
  last_update: false
}
```

Show multiple values of a P1 meter



```
blocks['plcounters'] = {
  type: 'dial',
  idx: 43,
  values: [
    {
      value: 'Data0',
      unit: 'kWh',
      label: 't1',
      scale: 0.001
    },
    {
      value: 'Data1',
      unit: 'kWh',
      label: 't2',
      scale: 0.001
    },
    {
      value: 'Data2',
      unit: 'kWh',
      label: 'ret t1',
      scale: 0.001
    },
    {
      value: 'Data3',
      unit: 'kWh',
      label: 'ret t2',
      scale: 0.001
    },
  ],
  showvalue: false,
  animation: false,
  shownumbers: true,
  fixed: true,
  width: 6
};
```

Selector switch

Selector switches will be displayed as a menu. The dial menu can be shown with or without (=default) title.



```
blocks['dm'] = {
  idx: 9,
  type: 'dial',
  title: true,
  width:6,
}

blocks['dm-notitle'] = {
  idx: 9,
  type: 'dial',
  width:6,
}
```

Toon Thermostat



“SwitchType”: “Selector”

```
blocks['toon_controller'] = {
  idx: 419,
```

(continues on next page)

(continued from previous page)

```

    title: 'Toon Controller',
    type: 'dial',
    width: 3,
}

```

1 = “Type”: “Temp”, 2 = “Type”: “Thermostat”

```

blocks['toon_thermostat_temp'] = {
  idx: '421', // -> 2
  title: 'Thermostat',
  type: 'dial',
  temp: 420, // -> 1
  width: 3,
}

```

Up-down dials

You can render a Thermostat as a dial with up-down buttons by setting subtype to updown:

```

blocks['thermupdown'] = {
  type: 'dial',
  subtype: 'updown',
  idx: 15,
}

```



You can add the temperature info from another device as well:

```

blocks['thermtempupdown'] = {
  type: 'dial',
  subtype: 'updown',
  idx: 15,
  temp: 36 //Use device 36 as actual temperature sensor
}

```



Light dimmers and Blinds can be rendered as up-down dials as well.



For Light dimmers the middle button will work as on-off switch.

For Blinds the middle button will work as stop button.

With the `inverted` block parameter you can invert the values: 10% will become 90%, 70% will become 30%, etc.

I prefer that for an Up Down blinds dial the Up-button will open the blinds. The blinds percentage goes from 0% (fully closed) to 100% (fully open).

This conflicts with the defaults in Domoticz where 0 is open, and 100 is closed.

For this reason the 'inverted' block parameter by default is set to true for regular Domoticz blinds devices, and set to false for Domoticz Blinds Inverted devices.

By setting the `steps` parameter you can adjust the step size. For Thermostats the default step value is 0.5. For Dimmers and Blinds the default step value is 10 (%).

Dial values

(Not applicable to blinds dials and up-down dials)

Each dial has a main value shown in the middle of the dial.

The values to be shown below the main dial value can be selected via the `values` parameters as follows:

```
blocks[16] = {
  type:'dial',
  values:['Humidity'],
  showunit: true
}
```

Assuming that device 16 is a TempHumBar device then with the above block definition the temperature will be shown (main value) and the humidity as additional value.



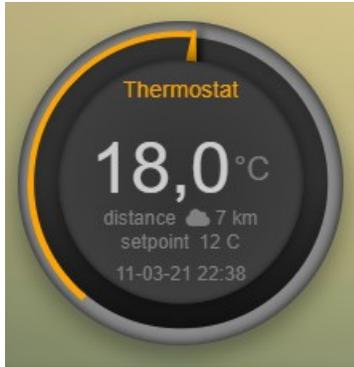
If needed you can customize the value units by adapting the `values` array as follows:

```
blocks[16] = {
  type:'dial',
  title:'HumBar',
  values: [
    {
      value:'Humidity',
      unit:'(%)',
    },
    {
      value:'Barometer',
      unit:'hPa',
    },
  ],
}
```



It's possible to combine data from several devices:

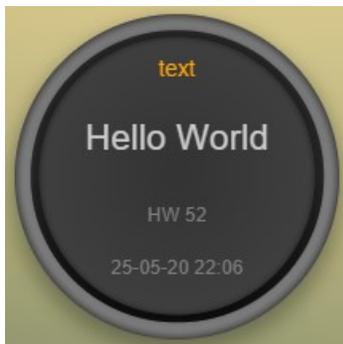
```
blocks['mytherm'] = {
  type: 'dial',
  idx: 19,
  temp: 16,
  min: 5,
  max: 30,
  values : [
    {
      idx: 10,
      label: 'distance',
      icon: 'fas fa-cloud',
      unit: 'km'
    },
    {
      label:'setpoint',
      idx: 19,
      unit: 'C'
    },
  ]
}
```



In this example the main device is device 25, which is a Thermostat device. The temperature value of device 27 is displayed, because the `temp` parameter is set to 25. Below the temperature two additional values will be displayed. As you can see you can add a label text as well.

To combine two text devices into one dial use the following:

```
blocks['combinedtext'] = {
  type: 'dial',
  idx: 15,
  values : [
    {
      idx: 16,
    },
  ]
}
```



With 15 and 16 two Domoticz Text devices.

And some more tricks:

```
blocks['combi'] = {
  type: 'dial',
  idx: 18,
  showvalue: false,
  values : [
    {
      idx: 52,
    },
    {
      idx: 16,
      value: 'Temp',
      label:'outside: ',
      unit: 'gr C',
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        addClass: 'w100'
    }
  ]
}

```



The base type of this block is a text block, because device 18 is text device. However, the value of this device is not shown, because the parameter `showvalue` is set to `false`.

Device 52 is a text device. The value is shown. Also the temperature of device 16 is displayed, with a custom label and unit. By adding 'w100' as utility class, this value is shown on a new line, instead on the same line as the other device.

By default, the 'Data' field of a device will be used as value. You can overrule this by setting the `value` parameter in the values object as shown before.

For text devices, the value will be interpreted as text instead of a number. For other devices you can add `type: 'text'` to the value object to enforce that the value will be handled as text as well.

Multiple values

You can add multiple values to most dial types. Or, add a needle representing the value of another device to for instance a dial switch:



```
blocks['sw1'] = {
  idx: 1056,
  type:'dial',
  values: [
    {
      idx: 1057,
      isNeedle: true
    },
  ],
  width: 6,
  showing: true,
  shownumbers: true,
  min: 0,
  max: 10
}
```

Value parameters

You can use the following parameters within the values definition of the dial:

Parameter	Description
label	Text to add in front of the value
icon	Name of the FontAwesome icon to place between label and value 'fas fa-car'
image	Image to place between label and value (it will replace icon if defined) 'image.jpg'
value	Name of the Domoticz device field to use as value
decimals	Number of decimals to use while formatting the value (default: 0)
scale	Multiplication factor for the value (default: 1)
type	Set to 'text' to handle value as text instead of number
unit	Text to add behind the value.
addClass	Name of the CSS class to add to this item.
isSetpoint	Handle this device/value as a setpoint device. You can adjust the device by rotating the needle.
isNeedle	The needle will follow the value of this device. It's read-only.

The following CSS classes are used:

.extra: All value items .item: One value item. .itemlabel: The label part of an item .dataunit: The combination of value and unit .data: The value part of an item .unit: The unit part of an item

The addClass parameter is applied on item level.

Custom Styling

In Domoticz you can hide the Off level of a Selector Switch. In Dashticz you can hide the Off level by adding the following code to your *custom.css*:

```
[data-id='<block_name>'] .dial-menu li:nth-child(1){
  display: none;
}
```

To change the grey dial bezel color from grey to red:

```
.dt_content .dial {
  background-color: #bb2424 !important;
}
```

To change the outer ring primary color from orange (default) to yellow:

```
.slice.primary {
  color: #d9e900;
}
```

To change the outer ring secondary color from blue (default) to lime green:

```
.slice.secondary {
  color: #26e500;
}
```

Split dials (dials which may have negative values) will receive the `negative` and `positive` class as well. In case you've redefined the primary or secondary styling in `custom.css`, then you have to update the positive/negative styling as well:

```
.slice.positive {
  color: red !important;
}

.slice.negative {
  color: blue !important;
}
```

To change the dial needle color from orange (default) to lime green:

```
.dial-needle::before {
  border-bottom-color: lime !important;
}
```

To target just one dial, you can prefix the above code snippets with block id of the dial, for example:

```
[data-id='temp_hum_baro'] .dial-needle::before {
  border-bottom-color: lime p!important;
}
```

Change the size of the dial-center:

```
.dial-center {
  height: 65%!important;
  width: 65%!important;
}
```

Hide extra data:

```
.dial[data-id='dial_name'] .extra {
  display: none;
}
```

Vertical center the dial menu:

```
.dial-menu .status {
  justify-content: center;
  display: flex;
  flex-direction: column;
}

.dial-menu .status li {
  margin: unset
}
```

Change the font of the dial menu text:

```
.dial-menu .status li {
  font-size: 75%
}
```

To change the colors of the blinds buttons:

```
.dialbtn.up {
  background-color: darkgreen;
}

.dialbtn.middle {
  background-color: darkblue;
}

.dialbtn.down {
  background-color: darkred;
}
```

And for the selected buttons:

```
/*Next block is the default styling*/
.dialbtn.selected {
  background-image: radial-gradient(rgba(255,255,255,0.5), rgba(0,0,0,0));
}

.dialbtn.up.selected {
  background-color: lightgreen;
}

.dialbtn.up.selected {
  background-color: lightred;
}
```

To change the text size in the up and down buttons of a blinds dial

```
.up .text, .down .text {
  font-size: 200%
}
```

Examples

Multicolor Selector Switch



CONFIG.js:

```
blocks['selector_switch'] = {
  idx: 123,
  type: 'dial',
  width: 5,
}

columns[1] = {}
columns[1]['blocks'] = ['selector_switch']
columns[1]['width'] = 5;
```

custom.js:

```
function deviceHook(device) {
  if (device.idx==123) {
    var level=parseInt(device.Level);
    device.deviceStatus='level'+level
  }
}
```

custom.css:

```
/*ring color*/
.level10 .dial-center {
  box-shadow: 0 0 25px 1px green !important;
}

/*selected item color*/
.level10 .status {
  --dial-color: green !important
}

/*ring color*/
.level20 .dial-center {
  box-shadow: 0 0 25px 1px red !important;
}

/*selected item color*/
.level20 .status {
  --dial-color: red !important
}
```

(continues on next page)

(continued from previous page)

```

/*ring color*/
.level30 .dial-center {
  box-shadow: 0 0 25px 1px blue !important;
}

/*selected item color*/
.level30 .status {
  --dial-color: blue !important
}

```

Windspeed



CONFIG.js:

```

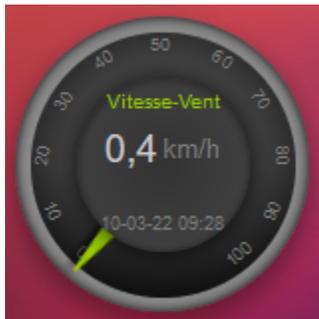
blocks['wind'] = {
  idx: 2442,
  title: 'knopen',
  type: 'dial',
  color: '#57c4d6',
  values: [
    {
      value: 'Speed',
      addClass: 'bigwind',
      decimals: 0,
    }
  ],
  setpoint: 18, // the entire outer ring will change color based on this s
  offset: 0, // 0 will point to the wind source, 180 will point to wind d
  showvalue: false,
  width: 12,
  showing: true,
  showunit: true,
  shownumbers: true,
  last_update: false
}

```

custom.css:

```
.dial-center {
    height: 65!important;width: 65!important;
}
[data-id='wind'] .dial-needle::before {
    border-bottom-color: red!important;
}
.bigwind {
    font-size: 300% !important;
    color: white !important;
    height: 40px !important;
}
```

Hide the additional data



You can set the values parameter to an empty array to hide the additional data, like this:

CONFIG.js:

```
blocks['windspeed'] = {
    idx: 39,
    title: 'Vitesse-vent',
    type: 'dial',
    subtype: 'windspeed',
    values: []
}
```

Domoticz log

You can add the Domoticz log to a column as follows:

```
columns[4] = {
    blocks: ['log']
}
```

```
2019-03-30 18:08:06.343 Status: EventSystem: Script event triggered:
/home/loki/domoticz/dzVents/runtime/dzVents.lua
2019-03-30 18:08:06.264 Status: dzVents: Info: ----- Finished
calculate_consumption.lua
2019-03-30 18:08:06.262 Status: dzVents: Info: ----- Start external script:
calculate_consumption.lua: Device: "Power (P1 Smart Meter)", Index: 43
2019-03-30 18:08:00.517 Status: dzVents: Info: Fan: ----- Finished
VentilatieBadkamer.lua
2019-03-30 18:08:00.516 Status: dzVents: Info: Fan: Target humidity is
49.998441169553
```

Garbage Collector

If you want to have a block with next pickup dates for your garbage, add the following to `CONFIG.js`, and change `zipcode` & `hounumber` to the correct data:

```
blocks['mygarbage'] = {
  company: 'cure',
  zipcode: '1234AB',
  street: 'vuilnisstraat',
  housenumber: 1,
  maxitems: 12,
  width: 12
}
```

Next, add the garbage to a column, like:

```
columns[1]['blocks'] = ['mygarbage']
```

You can change the colors of the trashcan (and/or the complete line) via the parameters in `CONFIG.js`.

Parameters

Block Parameter	Description
company	Garbage company to use. See <i>Currently supported cities/companies/services</i>
icalurl	'<url>': In case the garbage company is url the URL of the ical-file.
zipcode	The zipcode
street	Your street
houenum-ber	Your housenumber
maxitems	Number of items to show
maxdays	Number of days to show (2 is today and tomorrow). Default: 32
width	1..12: Width of the block
hideicon	true / false: To hide the garbage icon
use_names	true / false: shows name of the garbage type
use_colors	true / false: shows coloring for complete line
icon_use_colors	true / false: shows colored or only white trashcan
title	Title text
use_cors_prefix	true / false: use a CORS proxy for getting data from provider. See <i>PHP based CORS proxy</i>
mapping	Translation from description of the pickup event to a garbage type. See <i>Garbage type, color and icon</i> .
garbage	Settings for different garbage types. See <i>Garbage type, color and icon</i> .
date_separator	Text to place between the garbage type and date (default ':')
layout	Layout of the garbage rows: 0 for plain text layout, 1 for table layout, 2 for text layout with line break between garbage type and date.
ignoressl	false (default) / true: Set to true to disable https SSL checks

These block parameters can also be globally via a CONFIG.js setting:

CONFIG setting	Description
garbage_use_cors_prefix	true / false: use a CORS proxy for getting data from provider. See <i>PHP based CORS proxy</i>
garbage_mapping	Translation from description of the pickup event to a garbage type. See <i>Garbage type, color and icon</i> .
garbage	Settings for different garbage types. See <i>Garbage type, color and icon</i> .

Usage

Garbage type, color and icon

Dashticz uses two steps to show the garbage pickup type with the correct icon, text and coloring.

1. Garbage type detection. Configurable via parameter `garbage_mapping`
2. Settings (color, icon) per garbage type. Configurable via parameter `garbage`

To determine the garbage type Dashticz searches for certain text in the description of the garbage pickup event. Example for the default definition:

```
config['garbage_mapping'] = {
  rest: ['grof', 'grey', 'rest', 'grijs', 'grijze'],
```

(continues on next page)

(continued from previous page)

```

gft: ['gft', 'tuin', 'refuse bin', 'green', 'groen', 'Biodégradables', 'snoei'],
pmd: ['plastic', 'pmd', 'verpakking', 'kunststof', 'valorlux'],
papier: ['papier', 'blauw', 'blue', 'recycling bin collection'],
kca: ['chemisch', 'kca', 'kga'],
brown: ['brown', 'verre'],
black: ['black', 'zwart'],
milieu: ['milieu'],
kerstboom: ['kerst'],
};

```

As you can see 9 different garbage types have been defined. Looking at the first line of the garbage mapping: If the description of the pickup event contains the text `gft` the garbage type `rest` will be selected.

Note: The first rule that has a match with the event description will be selected.

After the mapping on a garbage type, the name, color and icon can be configured per garbage type as follows:

```

config['garbage'] = {
  gft: {kliko: 'green', code: '#375b23', name: 'GFT', icon: 'img/garbage/kliko_
↪green.png'},
  pmd: {kliko: 'orange', code: '#db5518', name: 'PMD', icon: 'img/garbage/kliko_
↪orange.png'},
  rest: {kliko: 'grey', code: '#5e5d5c', name: 'Restafval', icon: 'img/garbage/
↪kliko_grey.png'},
  papier: {kliko: 'blue', code: '#153477', name: 'Papier', icon: 'img/garbage/kliko_
↪blue.png'},
  kca: {kliko: 'red', code: '#b21807', name: 'Chemisch afval', icon: 'img/garbage/
↪kliko_red.png'},
  brown: {kliko: 'brown', code: '#7c3607', name: 'Bruin', icon: 'img/garbage/kliko_
↪brown.png'},
  black: {kliko: 'black', code: '#000000', name: 'Zwart', icon: 'img/garbage/kliko_
↪black.png'},
  milieu: {kliko: 'yellow', code: '#f9e231', name: 'Geel', icon: 'img/garbage/kliko_
↪yellow.png'},
  kerstboom: {kliko: 'green', code: '#375b23', name: 'Kerstboom', icon: 'img/
↪garbage/tree.png'},
};

```

The two examples above show the default definition of the `garbage_mapping` and `garbage` parameters. You can redefine them in your `CONFIG.js`.

In case there are multiple collection items on the same date, then they will be sorted based on the order of the keys in garbage parameter.

Currently supported cities/companies/services

Company	City or area
afvalalert	(Not working)
afvalstoffendienst	Afvalstoffendienst: 's-Hertogenbosch, Vlijmen, ... (NL)
almere	Almere (NL)
alphenaandenrijn	Alphen aan de Rijn (NL)
area	Coevorden, Emmen, Hogeveen (NL)

Table 4 – continued from previous page

Company	City or area
avalex	Avalex: Delft, ... (NL)
avri	Rivierenland (Zaltbommel, ...)(NL)
barafvalbeheer	Bar-afvalbeheer for Barendrecht, Rhoon (NL)
best	Best (NL)
blink	Blink: Asten, Deurne, Gemert-Bakel, Heeze-Leende, Helmond, Laarbeek, Nuenen, Someren (NL)
circulusberkel	Circulus Berkel: Apeldoorn, Bronckhorst, Brummen, Deventer, Doesburg, Epe, Lochem, Zutphen en Voorst (NL)
cure	Cure: Eindhoven, Geldrop-Mierlo, Valkenswaard (NL)
cyclusnv	Cyclus NV: Bodegraven-Reeuwijk, Gouda, Kaag en Braassem, Krimpen aan den IJssel, Krimpenerwaard, M...
dar	Dar: Berg en Dal, Beuningen, Druten, Heumen, Nijmegen, Wijchen (NL)
deafvalapp	Afval App (NL)
edg	EDG (DE)
gad	Grondstoffen- en Afvalstoffendienst regio Gooi en Vechtstreek (NL)
gemeenteberkelland	Berkelland: Borculo, Eibergen, Neede en Ruurlo (NL)
goes	Goes (NL)
googlecalendar	file in iCal format
groningen	Groningen (NL)
hvc	HVC Groep: 44 gemeenten in Flevoland, Noord- en Zuid-Holland (NL)
ical	File in iCal format
maashorst	Gemeente Maashorst: Uden, Volkel, Odiliapeel, Reek, Schaijk en Zeeland
meerlanden	Meerlanden: Aalsmeer, Bloemendaal, Diemen, Haarlemmermeer, Heemstede, Hillegom, Lisse, Noordwijk en ...
mijnafvalwijzer	Mijn Afval Wijzer (NL)
omrin	Leeuwarden, Opsterland, Heerenveen, Waadhoeke, ...
purmerend	Purmerend (NL)
rd4	RD4: Beekdaelen, Brunssum, Eijsden-Margraten, Gulpen-Wittem, Heerlen, Kerkrade, Landgraaf, Simpelveld, ...
recycleapp	RecycleApp (BE)
rmn	RMN: Baarn, Zeist, Nieuwegein, (NL)
rova	Rova (NL)
suez	Suez: Arnhem (NL)
sudwestfryslan	Sudwest Fryslan (NL)
twentemilieu	Twente Milieu (NL)
uden	Uden (NL)
veldhoven	Veldhoven (NL)
venlo	Venlo (NL)
venray	Venray (NL)
vianen	Vianen (NL)
waalre	Waalre (NL)
waardlanden	Waardlanden: Gorinchem, Hardinxveld-Giessendam, Molenlanden en Vijfheerenlanden (NL)

Styling

Via `custom.css` the appearance of the garbage blocks can be modified.

The generic CSS selector for a garbage block is `.garbage`. To select a specific garbage block, you can use

```
[data-id='mygarbage'].garbage
```

To give the garbage block a fixed height in combination with a vertical scroll bar if needed:

```
.garbage {
    height: 140px;
}
```

(continues on next page)

(continued from previous page)

```
overflow: auto
}
```

Instead of `.garbage` you can also use `.trash` which is maintained for backwards compatibility.

Additional CSS classes are applied to the garbage rows as follows:

- `.trashtoday`: For garbage collection scheduled for today
- `.trashtomorrow`: For garbage collection scheduled for tomorrow
- `.trashrow`: For garbage collection scheduled for the days after tomorrow

The `.trashtoday` and `.trashtomorrow` classes are also applied on block level.

To set the block styling depending on the trash today or tomorrow schedule, you can add the following to `custom.css`:

```
/* This will give the block a red border if trash collection is collected for today*/
.trash.trashtoday {
    border-color: red
}

/* This will give the block a green border if trash collection is collected for_
↳tomorrow*/
.trash.trashtomorrow {
    border-color: green
}

/* This will reduce the opacity in case no trash is scheduled for today or tomorrow */
.trash:not(.trashtoday):not(.trashtomorrow) .dt_state {
    opacity: 0.2 !important;
}

/* This will increase the font of the trash row that is scheduled for today */
.dt_state .trashtoday {
    font-size: 20px
}
```

Besides the row styling, you can also make use of column styling. The first column contains the trash type and can be selected via CSS class `.trashtype`. The second column contains the date separator, and can be selected via class `.trashsep`. The third column contains the date, and can be selected via class `.trashdate`.

As an example, the default CSS styling for the columns is as follows:

```
.trashsep {
    width: 10px;
    text-align: center;
}

.trashdate {
    text-align: right;
}
```

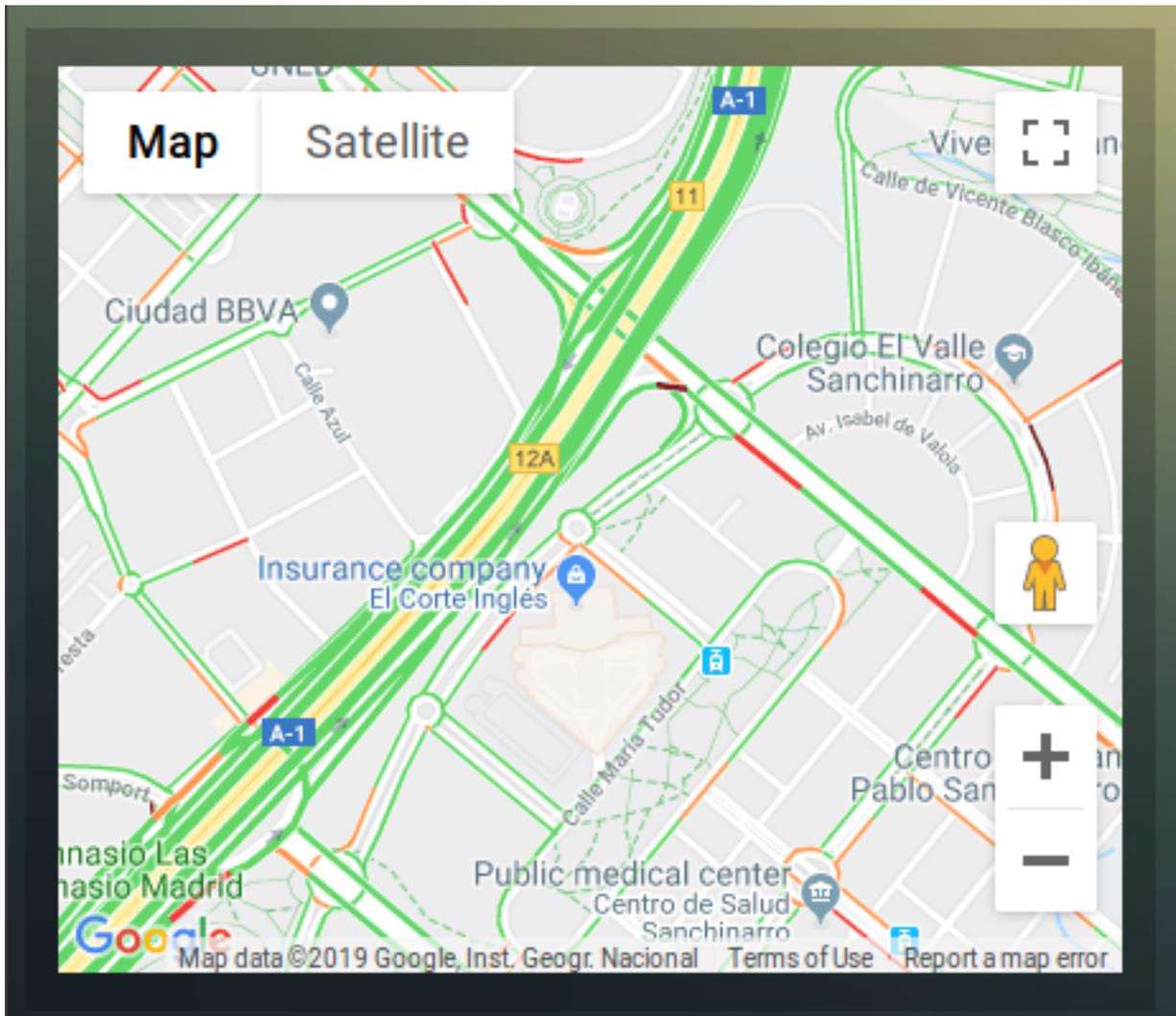
Upgrade from Dashticz 3.6.6 and earlier

In earlier versions of Dashticz the garbage block was configured via settings in `CONFIG.js` as follows:

```
var config = {}  
config['garbage_company'] = 'cure';  
config['garbage_icalurl'] = 0;  
config['garbage_zipcode'] = '1234AB';  
config['garbage_street'] = 'vuilnisstraat';  
config['garbage_housenumber'] = '1';  
config['garbage_maxitems'] = '12';  
config['garbage_width'] = '12';
```

Although this still is supported, it's recommend to switch to the new block method as described in the first section.

Google Maps



First get a Google Maps API key: *Getting a Google Maps API key*

Adding the GM to the dashboard:

```
config['gm_api'] = 'xxxxxxxxxxxxxxxx'; // The API key you received from Google
```

(continues on next page)

(continued from previous page)

```
var maps = {}
maps.location = { height: 400, width:4, latitude: 40.4989948, longitude: -3.6610076,
↳zoom:15 }
```

and used in:

```
columns[3] = {}
columns[3]['blocks'] = [maps.location]
```

Block parameters

The Google Maps module uses the following block parameters:

Parameter	Description
width	Width of the Google Maps block compared to the full screen, not the column. Full screen width is 12. Meaning width of 6 will give a width of 50% of the screen size.
height	The height of the block in pixels
latitude	Latitude
longitude	Longitude
zoom	Zoom level

Config settings

The Google Maps module uses the following config settings:

Setting	Description
gm_api	The API key you received from Google

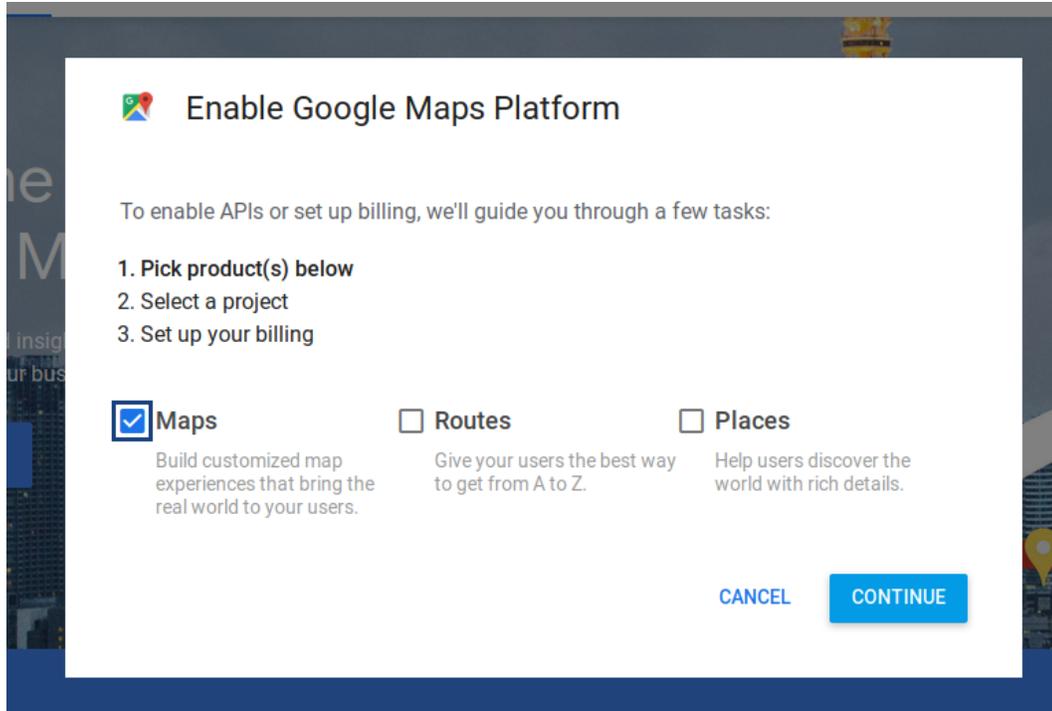
Usage

Getting a Google Maps API key

Obtain your Google Maps API key from:

<https://developers.google.com/maps/documentation/javascript/get-api-key>

- Press Get Started



- Choose Maps, and press Continue
- Follow the instructions

HTML custom block

This module can display a custom html block.

As example we will add a weather widget created on <https://weatherwidget.io>

Configure your widget on weatherwidget.io. Copy-paste the widget code into a html-file in the custom folder.

Example for Amsterdam:

```
<a class="weatherwidget-io" href="https://forecast7.com/en/52d374d90/amsterdam/" data-label_1="AMSTERDAM" data-theme="original" >AMSTERDAM</a>
<script>
!function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0];if(!d.getElementById(id))
  {js=d.createElement(s);js.id=id;js.src='https://weatherwidget.io/js/widget.min.js';
  fjs.parentNode.insertBefore(js,fjs);}(document,'script','weatherwidget-io-js');
</script>
```

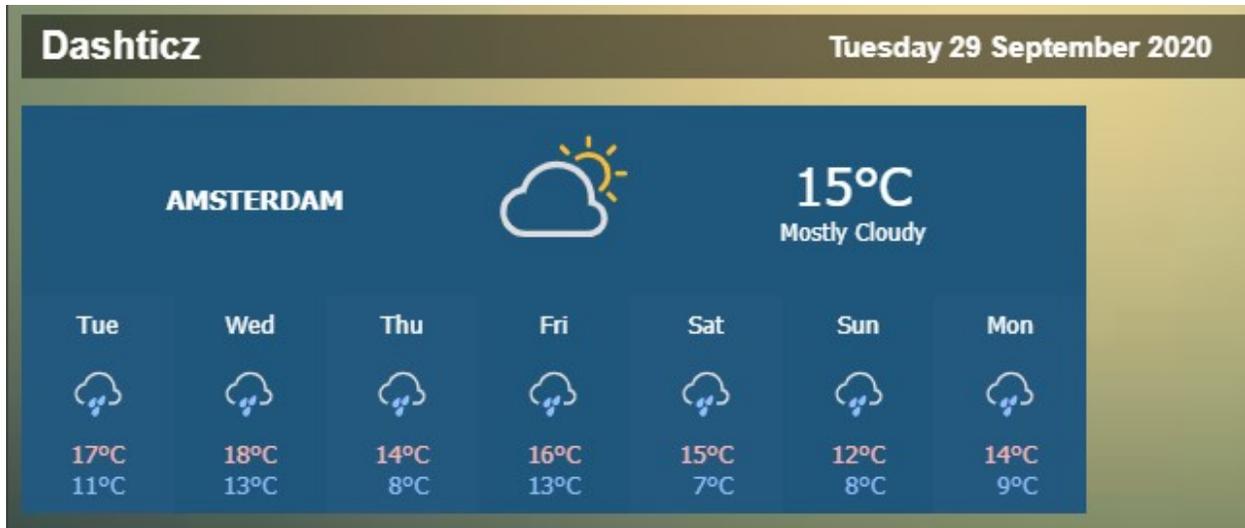
Copy this code into a file `custom/weatherwidget.html`

Create a block containing the `htmlfile` parameter:

```
blocks['weatherwidget'] = {
  htmlfile: 'weatherwidget.html'
}
```

Add “weatherwidget” to a column:

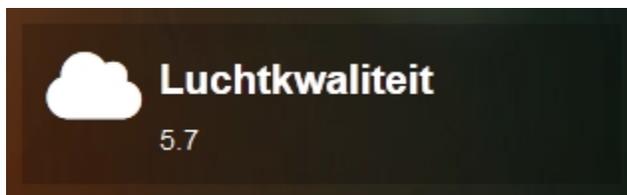
```
columns[1]['blocks'] = [
  'weatherwidget',
```



Parameters

Parameter	Description
htmlfile	Filename of the html code (relative to custom/)
margin	true / false: To display a margin around the html block (default: false)
title	'<string>': Custom title for the block
width	The block width
icon	The icon to show in the block. Default no icon.
image	The image to use instead of an icon. Location is relative to ./img

Longfonds



RIVM measures our air quality on a daily basis. The air is not equally healthy everywhere in the Netherlands. With the Longfonds check <https://www.longfonds.nl/gezondelucht/check> you can see how healthy the air is in your neighbourhood. The data for this check comes from the RIVM. - The air quality is a value from 0 (good) to 11 (very bad)

You can use the following config settings or use the block parameters to set the address:

```
config['longfonds_zipcode'] = '1234AZ';
config['longfonds_housenumber'] = '99';
```

You can customize longfonds as follows:

```
blocks['longfonds'] = {
    title: 'Luchtkwaliteit',
    switch: true,
```

(continues on next page)

(continued from previous page)

```
width: 12,
}
```

Add 'longfonds' to your column

Block parameters

Parameter	Description
zipcode	<zipcode>: Zipcode (if not set in config)
houenumber	<houenumber>: Houenumber (if not set in config)
width	1 . .12: The width of the block relative to the column width
title	'<string>': Custom title for the block
icon	Defines the icon for this block, choose from: https://fontawesome.com/icons?d=gallery&m=free 'fas fa-eye'
image	If you want to show an image instead of an icon, place image in img/ folder 'bulb_off.png'
url	'<url>': URL of the page to open in a popup frame or new window on click.
newwindow	0: open in current window 1: open in new window 2: open in new frame (default, to prevent a breaking change in default behavior) 3: no new window/frame (for intent handling, api calls). HTTP get request. 4: no new window/frame (for intent handling, api calls). HTTP post request. (forcerefresh not supported)

Moon

With the moon block you can add a picture of the current moon phase to your dashboard. Use the following code:

```
columns[2] = {}
columns[2]['blocks'] = [ 'moon' ]
```

If needed you can change the width:

```
blocks['moon'] = {
  width: 6
}
```

We have 100 moon images. A moon cycle takes approximately 28 days. That means that the moon picture will refresh approximately 4 times a day.



News

This module can display a rss feed.

A predefined modules exists with the name 'news', which can be configured with the parameter `default_news_url`. See *Config Settings*

```
config['default_news_url'] = 'http://www.nu.nl/rss/Algemeen';
```

Additional rss feeds can be defined as well. See below:

```
blocks['news_tweakers'] = {
    feed: 'http://feeds.feedburner.com/tweakers/nieuws',
    showimages: false,
    icon: 'fas fa-newspaper'
}
```

Add the news blocks to a column as usual:

```
columns[5] = {
    blocks: ['news', 'news_tweakers']
}
```

Config Settings

Settings	Description
<code>default_news_url</code>	URL of the default news feed 'http://www.nu.nl/rss/algemeen' = Example for nu.nl
<code>news_scroll_after</code>	Enter the ammount in seconds (delay) 5 = Scroll the news message every 5 seconds

News Parameters

Parameters	Description
feed	URL of the news feed 'http://www.nu.nl/rss/algemeen' = Example for nu.nl
maxheight	'max' (default) : Adjust the height of the block to the maximum height of all the news items. 'auto' : Adjust the height to fit the displayed news item. <number> : Set the height of the block to <number> pixels.
title	Title of the news block
icon	'fas fa-newspaper': icon to show in the news block
image	'image.png': image to show as icon. Image path is relative to the <dashticz>/img folder.
showimages	Show the inline images of the rss feed. false: Do not show the inline images true (=default): Show the inline images
filter	Filter the number of items or filter on publish date of items '5 items': Show the 5 most recent news items '3 days': Show the news items from the last three days.

Example

Add the following to CONFIG.js:

```
config['default_news_url'] = 'http://www.nu.nl/rss/Algemeen';

blocks['news_tweakers'] = {
  feed: 'http://feeds.feedburner.com/tweakers/nieuws'
}

columns[5] = {
  blocks: ['news', 'news_tweakers'],
  width: 4
}
```

(continues on next page)

(continued from previous page)

```
//Definition of screens
screens = {}
screens[1] = {
  columns: [1, 2]
}
screens[2] = {
  columns: [3,4]
}

screens[3] = {
  columns: [5]
}
```

This will give on the third screen the following result:



If you click on a news item, then the news article will be shown in a popup window. Not all rss-feeds support this. The news articles from Tweakers for instance are blocked.

Usage

The news module will download the information via a CORS proxy. The default settings normally work fine. For configuration see *PHP based CORS proxy*

Not all rss-feeds support inline images. In that case you can set `showimages` to `false`.

NZBGet

Note: Not tested. Please leave a note in the Dashticz Support Forum if this module is working correctly.

Show your current downloads within the Dashboard.

Add to `config.js`:

```
var _HOST_NZBGET = 'http://192.168.1.3:6789';
```

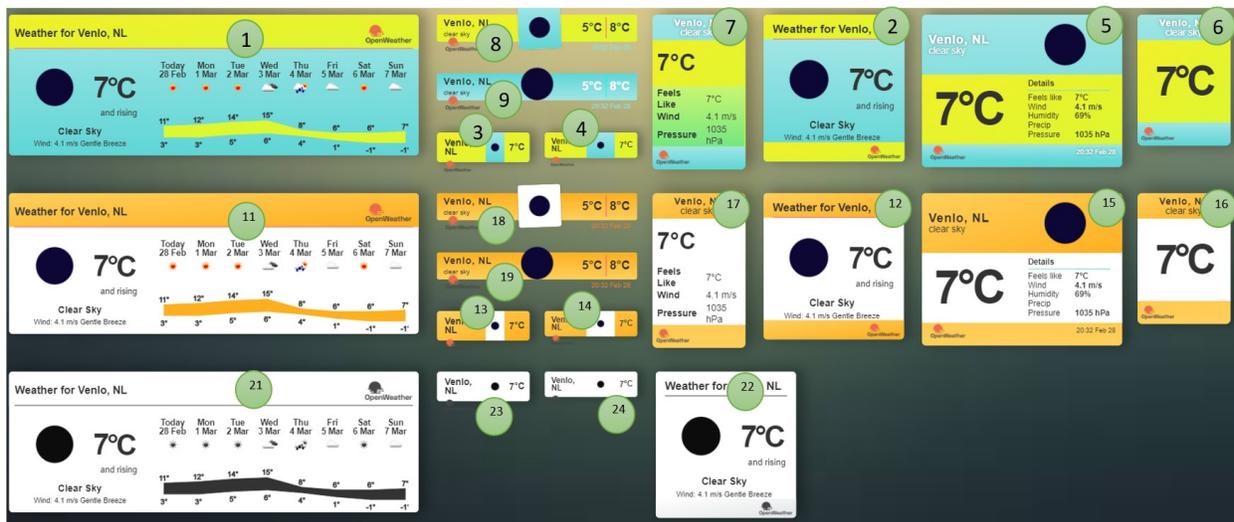
AND in the blocks, add:

```
columns[1]['blocks'] = ['nzbget']
```

If you want to use other widths, add this to config:

```
blocks['nzbget'] = {}  
blocks['nzbget']['width'] = 12;  
blocks['nzbget']['downloads_width'] = 6;
```

OWM widgets



Collection of Open Weather Map widgets.

First create an account on <https://openweathermap.org> to obtain your free api key. You can use the following config settings or use the block parameters to set the api key:

```
config['owm_api'] = '123abc...';
```

You can choose one of the 24 widgets with the layout block parameter:

```
blocks['mywidget'] = {  
  type: 'owmwidget', //to select a owm widget  
  layout: 11, //1 .. 24  
  width: 12,  
}
```

Block parameters

Parameter	Description
apikey	'123abc...': OWM api key. Default setting is the value of config['owm_api']
layout	11: Select a layout from 1 to 24
city	'Amsterdam': The city name to search for. Default is config['owm_city'] value. 2643743: or the Open Weather Map city id.
country	'nl': Country used in the city name search. Default is config['owm_country'] value.
width	1..12: The width of the block relative to the column width

Public Transport

Dashticz currently supports the following types of public transport info:

1. Public transport timetable, configurable via a public transport block:

- Train info Netherlands ('treinen')
- Bus/tram/boat info Netherlands ('ovapi')
- irail (Belgium)
- delijn (Belgium)

2. Predefined blocks, see *Predefined public transport blocks*

- 'traffic'. Traffic info from [Rijkswaterstaat Verkeersinfo](#) (Netherlands)
- 'train'. Train status from [Rijden de Treinen](#) (Netherlands)

A public transport block can be configured as follows:

```
//example station id: Utrecht
blocks['treinen']= {
  station: 'UT',
  title:'OV Info',
  show_lastupdate:true,
  provider: 'treinen',
  show_via: true,
  icon: 'fas fa-train',
  results: 5
};
```



OV Info

11:11	's-Hertogenbosch <i>Sprinter</i> <i>Vaartsche Rijn, Lunetten, Houten, Geldermalsen</i>	spoor 21
11:12	Hoofddorp <i>Sprinter</i> <i>Overvecht, Hilversum, Naarden-Bussum, Schiphol Airport</i>	spoor 1
11:12	Schiphol Airport <i>Intercity</i> <i>Bijlmer ArenA, Amsterdam Z.</i>	spoor 7
11:13	Den Haag Centraal <i>Intercity</i> <i>Gouda</i>	spoor 9
11:14	Heerlen <i>Intercity</i> <i>'s-Hertogenbosch, Eindhoven C., Weert, Roermond</i>	spoor 18

Parameters

Parameter	Description
station	The station id. See Usage sections below to find how to obtain the right station id.
title	Title of the block
show_lastupdate	false, true. To display the time of the last update.
provider	Public transport info provider to use. Choose from 'treinen' Netherlands: trains 'ovapi' Netherlands: bus, tram, boat 'irailbe' Belgium: trains 'delijnbe' Belgium: bus, tram, boat
direction	Set the line direction(s) to filter the direction. (comma seperated) (for ovapi, delijnbe) '1' : Filter on direction 1 '1,2' : Filter on direction 1 or 2
destination	Set the end destination station name to filter the direction. 'Den Haag De Uithof,Den Haag Loosduinen' (comma seperated, case sensitive)
service	Set the specific services (Dutch: lijnummers) to further filter the result '3,4' (comma seperated)
show_via	false, true. Hide the via-part.
show_direction	false, true. Show the line direction (only for ovapi and delijnbe).
lang	'nl', 'fr', 'en', 'de' : Set the language for search results (only for irailbe) Default value is derived from the Dashticz language setting.
icon	The font-awesome icon (including fas fa-) 'fas fa-bus', 'fas fa-tram', 'fas fa-train', 'fas fa-ship', 'fas fa-subway',...
interval	time in seconds for refreshing the data
results	Number of results to show
width	To customize the width. It's not recommended to change the default value (12) because of the size of the output.
url	'<url>': URL of the page to open in a popup frame or new window on click.
newwindow	0: open in current window 1: open in new window 2: open in new frame (default, to prevent a breaking change in default behavior) 3: no new window/frame (for intent handling, api calls). HTTP get request. 4: no new window/frame (for intent handling, api calls). HTTP post request. (forcerefresh not supported)
3.2. Blocks	145

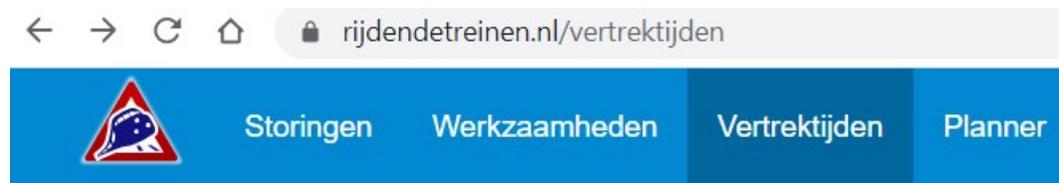
Usage

Treinen

```
blocks['treinen']= {
  station: 'UT',
  title: 'OV Info',
  show_lastupdate: true,
  provider: 'treinen',
  show_via: true,
  icon: 'fas fa-train',
  results: 5
};
```

The station code can be found in search box at: <https://www.rijdendetreinen.nl/vertrektijden>

The station code is the short code right of the station name:



toestemming geven om je locatie te delen.

Je kunt ook [actuele aankomsttijden](#) bekijken voor ieder treinstation in Neder
Bekijk dan het [treinarchief](#).

Station



Examples:

- Utrecht: UT
- Amsterdam Centraal: ASD

ovapi

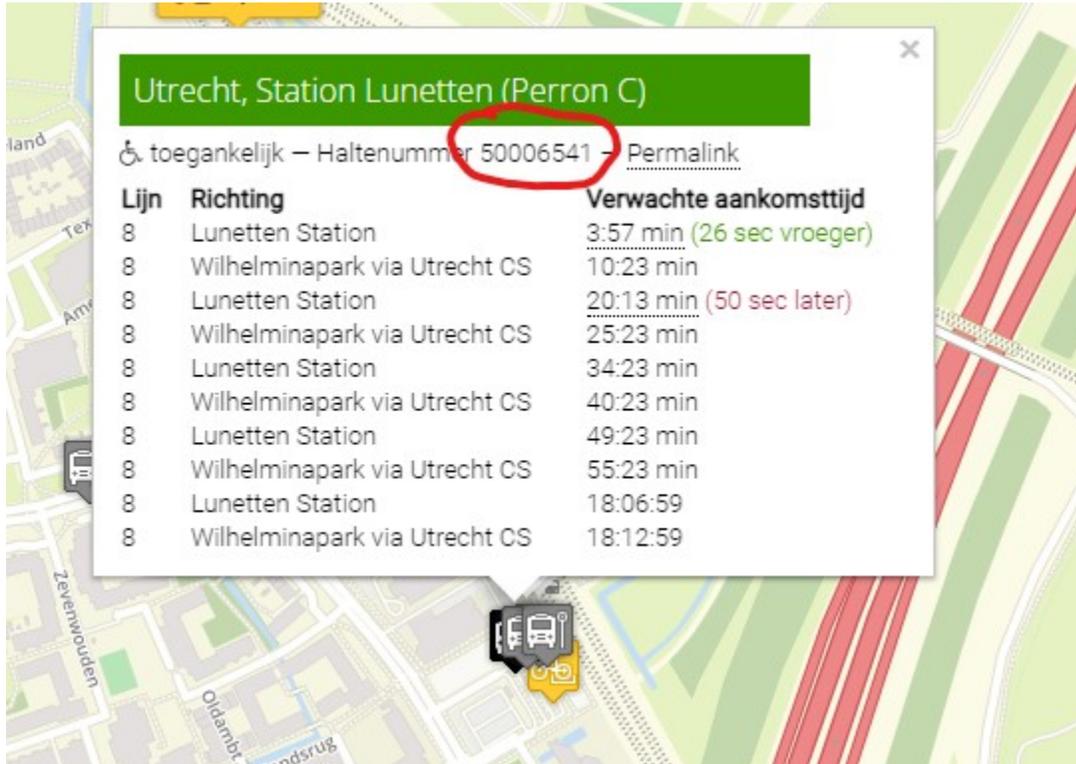
Use ovapi to obtain bus/tram/boat departure information:

```
blocks['ovapi'] = {
  station: 'utrlun', //utrlun, Amstel: 60094
  title: 'Utrecht Lunetten',
  show_lastupdate: true,
  provider: 'ovapi',
  show_via: true,
  icon: 'fas fa-bus',
  results: 5
};
```

In the previous example bus station Utrecht Lunetten is used. A bus station can be a collection of several bus stops. For instance, the busstation close to a railway station often has several platforms. Or, if there is a busstop at both sides to the road, then this also may be defined as busstation.

- A bus station has a station code.
- A bus stop has a so called tpc code.

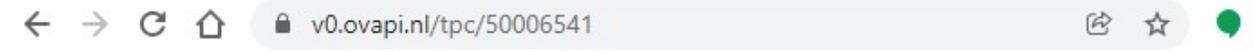
The tpc codes for individual bus stops can be found on <https://ovzoeker.nl>. On the map click on a bus stop. The popup window will show the tpc code, which is the number behind 'halte nummer':



In the previous example the tpc code for Utrecht Lunetten Perron C is 50006541.

To find the station code follow the following url: <https://v0.ovapi.nl/tpc/50006541>

In the json code that will be displayed locate the first areacode:



```
{
  "50006541": {
    "Stop": {
      "Longitude": 5.1431956,
      "Latitude": 52.06596,
      "TimingPointTown": "Utrecht",
      "TimingPointName": "Utrecht (Perron C)",
      "TimingPointCode": "50006541",
      "StopAreaCode": "utrlun",
      "TimingPointWheelChairAccessible": "NOTA",
      "IntVisualAccessible": "NOTACCESSIBLE",
      "Passes": {
        "Q00ZL_20220123_u008_7069_0":
      }
    }
  }
}
```

If you want to show all departures from all stops within a station (area) use the area code as `station` block parameter, like in the example code block above:

```
station: 'utrlun',
```

If you want to show only the departures from one specific stop or platform, use the `tpc` code as `tpc` block parameter, and remove the `station` parameter. Example:

```
blocks['mystop'] = {
  tpc: '50006541',
  title: 'Utrecht Lunetten, perron C',
  provider: 'ovapi',
  results: 5
};
```

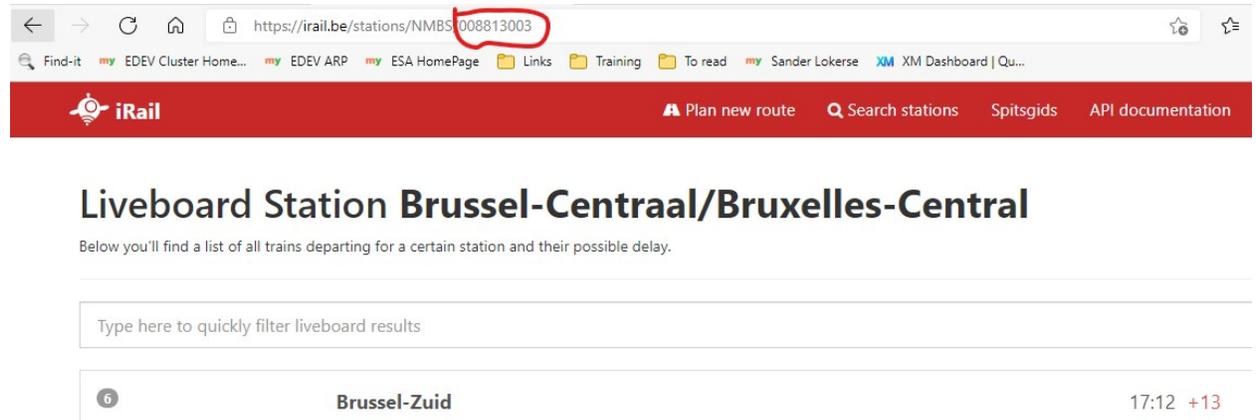
If you want to show all departures from all stops within a station (area), but there's *no station code available* you can use multiple `tpc` codes. Example:

```
blocks['mystops'] = {
  tpc: '53602050,53602060',
  title: 'Prof. Waterinklaan',
  provider: 'ovapi',
  results: 6
};
```

irail

To find the station code fill in the search box on: <https://irail.be/stations/NMBS>

After selecting your favorite station, and clicking on 'View Liveboard' the station code is the last word in the url in the address bar:



For Bruxelles Central the station code is 008813003:

```
blocks['irailbe'] = {
  station: '008813003',
  title: 'irailbe Brussel Central',
  show_lastupdate: true,
  provider: 'irailbe',
  show_via: true,
  icon: 'fas fa-train',
  results: 5
};
```



De lijn

The station code consists of 6 digits. Search for your station code in the search box on <https://delijn.be>

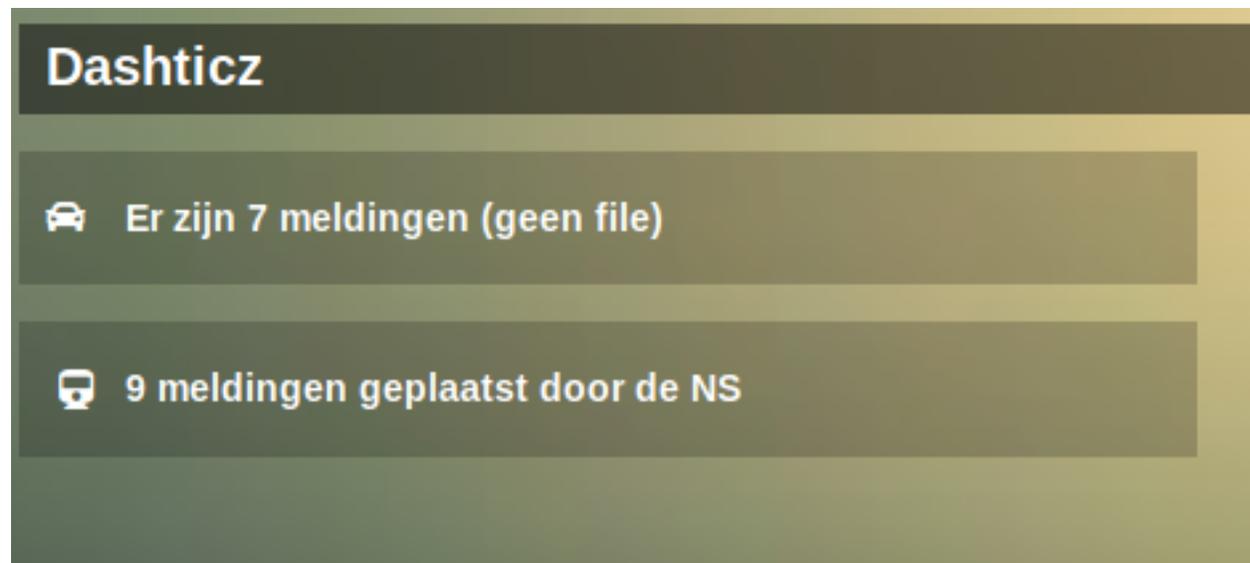
Predefined public transport blocks

There are two predefined public transport blocks:

- 'traffic': Rijkswaterstaat Verkeersinfo (The Netherlands)
- 'train': Rijden de Treinen (The Netherlands)

Example for your "CONFIG.js":

```
columns[2] = {
  blocks: ['traffic', 'train'],
  width: 5
}
```



VVS

Not supported anymore (VVS disabled it's api)

9292.nl

Not supported anymore. 9292 doesn't provide a public API key.

Styling

Font size can be changed by adding this to your `custom.css` and change to your own preference:

```
.publictransport div {
  font-size: 13px;
}
```

In case no info is available then the CSS class `empty` will be added to block. This can be used to adjust the styling of an empty block via `custom.css`

Domoticz Security Panel

There are three ways to show the Domoticz Security Panel:

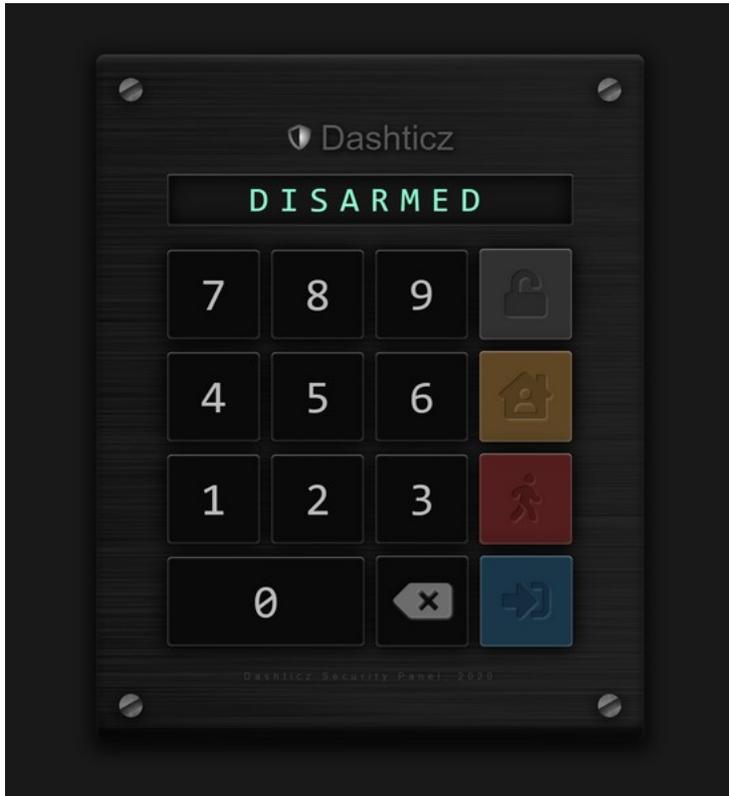
- As special block 'secpanel'
- As Domoticz device
- In a frame

Security Panel special block

Add the special block to a column:

```
columns[1]['blocks'] = ['secpanel'];
```

This will show a Dashticz security panel, which automatically scales to the column width.



Block parameter

Parameter	Description
decorate	<p><code>true</code>: Show security panel decorated with screws, title, background image. (=default)</p> <p><code>false</code> Remove the decorations. This will result in a block with reduced height.</p>
scale	<p>Scale factor for the width of the clock. Should be smaller than 1. Height scales automatically.</p> <p><code>0.75</code>: Scales the clock down to 75% (default <code>1 = 100%</code>).</p>
headerText	<p>The text to show in the header of the security panel</p> <p><code>'Dashticz'</code> (=default)</p>
footerText	<p>The text to show in the header of the security panel</p> <p><code>'Dashticz Security Panel, 2020'</code> (=default)</p>

Additionally, if you have set your Domoticz security panel to “Armed Away”, you can now configure Dashticz to secure automatically by applying the following setting in *CONFIG.js*:

```
config['security_panel_lock'] = 1;
```

If the panel has been set to “Armed Away”, it will display a fullscreen panel. The user will need to enter the code and either set “Disarm” or “Armed Home” to enable the blue enter button”. Pressing this will then show your Dashticz main screen.

Buttons:

- Green: Disarm
- Amber: Armed Home
- Red: Armed Away
- Blue: Enter Dashticz

The countdown delay is set in *Domoticz, Setup > Settings > Security Panel > Delay*. If set to zero, there will be no countdown.

The above fullscreen lock feature does not require you to also have a Security Panel block added in *CONFIG.js*. But if you want one, you can add one:

```
blocks['secpanel'] = {  
  title: '',  
  width: 6  
}
```

Or you can have a block defined and disable the fullscreen lock feature with this (which is the default if not set):

```
config['security_panel_lock'] = 0;
```

To activate the security lock while in ‘Armed home’ mode use the following:

```
config['security_panel_lock'] = 2;
```

The secpanel height is based on its width to maintain the correct aspect ratio. The height is 1.35 x the width. You can set the height and width in the block, or in custom.css:

```
[data-id='secpanel'] .dt_content {  
  height: 200px!important;  
  width: 148px!important;  
}
```

So, when the system is “Armed Away”, you want the fullscreen security panel to blink if a certain device has been changed state, e.g. closed > open?

You will need to add the “alarm” class to the fullscreen security panel’s modal (background):

```
function getStatus_123(block) {  
  if (block.device.Data == 'open') {  
    $('.sec-modal').addClass('alarm');  
  }  
}
```

This would need to be added to your custom.css:

```
.alarm {
  -webkit-animation: flash 1s infinite; /* Safari 4+ */
  -moz-animation: flash 1s infinite; /* Fx 5+ */
  -o-animation: flash 1s infinite; /* Opera 12+ */
  animation: flash 1s infinite; /* IE 10+, Fx 29+ */
}

@keyframes flash {
  0%, 49% {
    background-color: #681717;
  }
  50%, 100% {
    background-color: #e50000;
  }
}
```

Security Panel block



This is the Domoticz Security Panel as block. You can use the normal block parameters. See [Block parameters](#)

```
blocks[123] = {}; //123 is the Domoticz Security Panel device ID
```

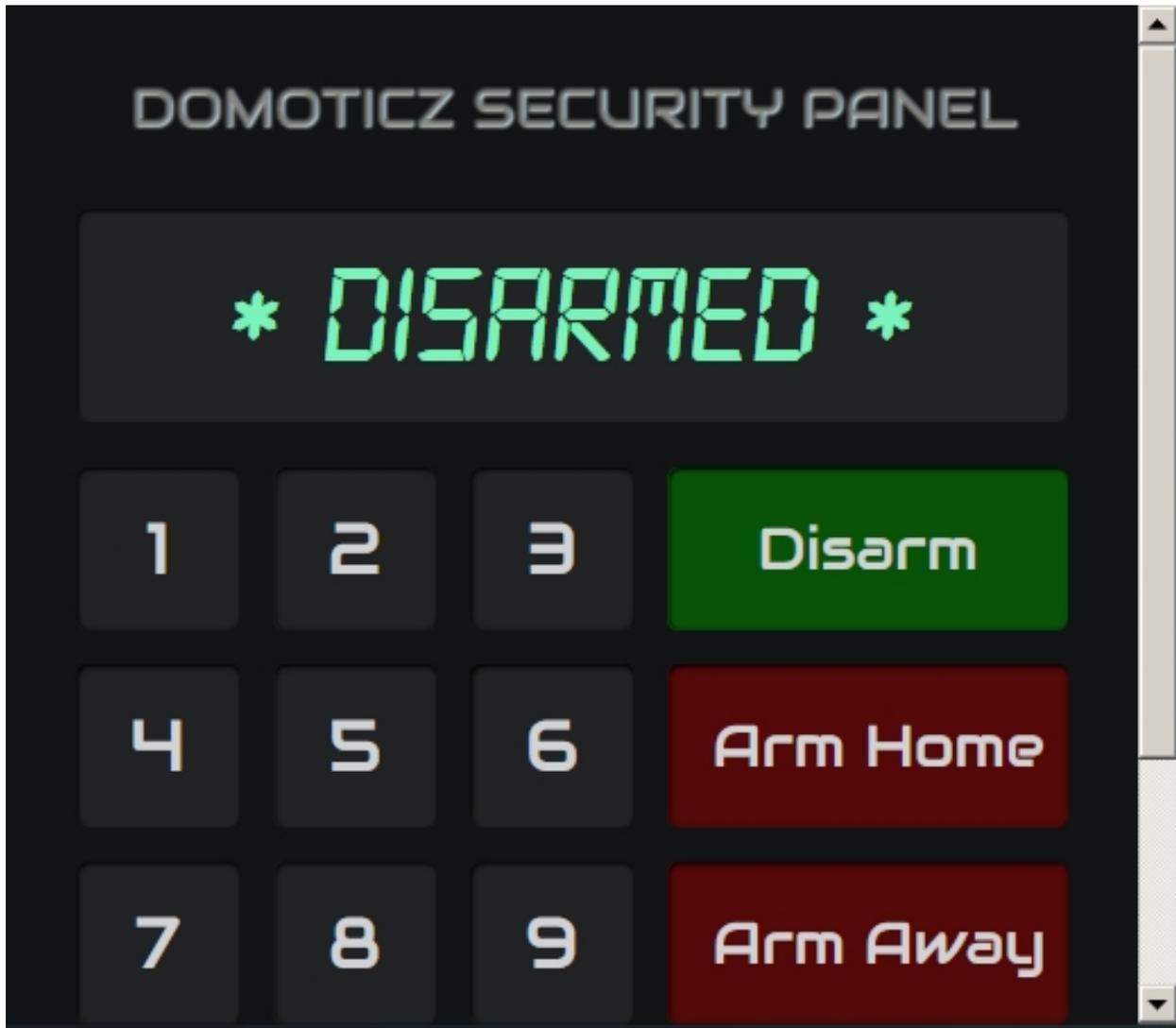
You can add this block to a column with:

```
columns[1]['blocks'] = [123];
```

CSS Tip! To the active button an additional class `btn-warning` is added:

```
.block_123 .btn.btn-warning {
  color: black !important;
}
```

Security Panel frame



This is the Domoticz Security Panel as frame. You can use the normal frame parameters. See [Frames](#)

```
frames.secpanel = {key: 'secpanel', height: 390, width: 12, frameurl: "http://<YOUR_
↳DOMOTICZ_IP>:<PORT>/secpanel/index.html"}
```

You can add this frame to a column with:

```
columns[1]['blocks'] = [frames.secpanel];
```

CSS Tip! What you can do to scale the content of the iframe. Assuming you add key: 'secpanel' to the frames.secpanel definition, you can scale the secpanel with:

```
[data-id='secpanel'].frame iframe {
  transform: scale(0.5);
  border: 0px;
  height: 600px !important;
  width: 200%;
```

(continues on next page)

(continued from previous page)

```
max-width: 200%;  
transform-origin: 0 0;  
}
```

Sonarr

Note: Not tested. Please leave a note in the Dashticz Support Forum if this module is working correctly.

Show your current series within the Dashboard.

Add to config.js:

```
config['sonarr_url'] = 'http://sonarrserver:8989';  
config['sonarr_apikey'] = '00000000000000000000000000000000';  
config['sonarr_maxitems'] = 8;
```

AND in the columns, add:

```
columns[1]['blocks'] = ['sonarr']
```

If you want to use extra options, add this to config:

```
blocks['sonarr'] = {}  
blocks['sonarr']['title'] = 'Sonarr';  
blocks['sonarr']['width'] = 8;  
blocks['sonarr']['title_position'] = 'left';  
blocks['sonarr']['view'] = 'banner';
```

Spotify

Add a Spotify block to your Dashboard.

With the Spotify block you can select a Spotify playlist, set volume, skip to next song.

The Spotify block in Dashticz is not a player, but a remote control. That means there must be a Spotify player somewhere in your local network. This can be your laptop running the Spotify player, or a Spotify connect speaker.

Note: You must have a Spotify Premium account

Before you can use it, create an app on the [Spotify website](#):

Title: Dashticz

Description: whatever ;)

On the next page, copy “Client ID” to your CONFIG.js like:

```
config['spot_clientid'] = '1112f16564cf4f4dsd4cbe8b52c58a44';
```

At the same page, enter your dashticz-url in the Redirect URIs field, for example: `http://192.168.1.3:8080/`

Warning: Don't forget to push the save button!

In `CONFIG.js` add the block like:

```
columns[2] = {}  
columns[2]['blocks'] = ['spotify']  
columns[2]['width'] = 5;
```

Complete example:

```
config['spot_clientid'] = '1112f16564cf4f4dsd4cbe8b52c58a44';  
  
columns[2] = {}  
columns[2]['blocks'] = ['spotify']  
columns[2]['width'] = 5;
```

Streamplayer

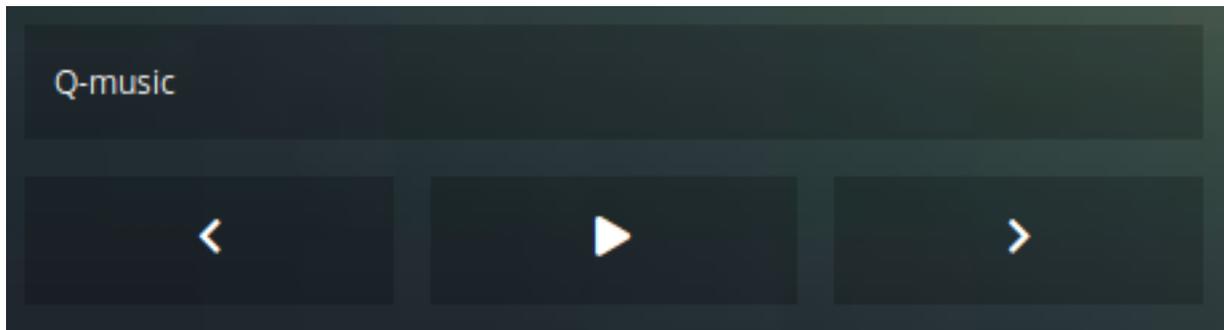
For those who like to listen to the radio on Dashticz, there is a Plugin available.

Add the following to your `CONFIG.js`:

```
var _STREAMPLAYER_TRACKS = [  
  {"track":1,"name":"Q-music","file":"http://icecast-qmusic.cdp.triple-it.nl/Qmusic_  
↪nl_live_96.mp3"},  
  {"track":2,"name":"538 Hitzzone","file":"http://vip-icecast.538.lw.triple-it.nl/  
↪WEB11_MP3"},  
  {"track":3,"name":"Slam! NonStop","file":"http://stream.radiocorp.nl/web10_mp3"},  
  {"track":4,"name":"100%NL","file":"http://stream.100p.nl/100pctnl.mp3"},  
  {"track":5,"name":"StuBru","file":"http://mp3.streampower.be/stubru-high.mp3"},  
  {"track":6,"name":"NPO Radio 1","file":"http://icecast.omroep.nl/radio1-bb-mp3"},  
  {"track":7,"name":"Omroep Brabant","file":"http://streaming.omroepbrabant.nl/mp3"},  
];
```

To enable, use the key: 'streamplayer' in the block definitions:

```
columns[2]['blocks'] = [1,4,'streamplayer']
```



To change the color of the Streamplayer buttons add the following to your `custom.css`:

```
div[data-id='streamplayer'] > div {  
background-color: blue !important;  
}
```

(continues on next page)

(continued from previous page)

```
div[data-id='streamplayer'].playing > div {  
background-color: orange !important;  
}
```

This last part is applied if the Streamplayer is in playing state.

To add an image to the Streamplayer add the following to your `CONFIG.js`:

```
blocks['streamplayer'] = {  
  image: 'radio.png'  
}
```

If you wanna use an icon instead of an image you have to change `image` to `icon`. You choose an icon from the [FontAwesome Free](#) set.

Sunrise

This block will add the actual sunrise and sunset info to Dashticz. Example:

```
columns[1]['blocks'] = ['sunrise'];
```



Timegraph

The timegraph block collects the actual measurement data of one or more Domoticz devices, and shows them in a moving time chart.

```
blocks[43] = {  
  type: 'timegraph',  
  values: ['NettUsage'],  
  duration: 60  
}
```

There is no permanent storage: After a refresh of Dashticz the previously collected data is lost.

Block parameters

Parameter	Description
width	The width of the block relative to the column width 1..12
title	Custom title for the block '<string>'
icon	Defines the icon for this block, choose from: https://fontawesome.com/icons?d=gallery&m=free 'fas fa-eye'
image	If you want to show an image instead of an icon, place image in img/ folder 'bulb_off.png'
type	type must be set to 'timegraph' for a time graph 'timegraph'
height	The graph height. '300px': The graph will have a height of 300 pixels
duration	Duration of the chartwindow in seconds. 60: Default: 300 (=5 minutes)
xTicks	Number of labels on the x-axis (= time axis) 10 (=default)
yTicks	Number of labels on the y-axis (= vertical axis) 5 (=default)
xLabels	To show labels on the x-axis true (=default): Show labels on the x-axis false : Hide labels on the x-axis
animation	Duration of the animation effect in msec. (Default: 0). 0 ... 1000. Maximum recommended value is 1000 msec
lineTension	With this parameter you can smoothen the graph line.

Usage

The `values` parameter is the most important parameter for this block. You can use it to select the data you want to show from your device. It's also possible to combine data from several devices in one graph.

Example to show the temperature of a THB device:

```
blocks[659] = {
  type: 'timegraph',
  values: ['Temp'],
  duration: 60
}
```

To show the usage and delivery graph of a P1 meter:

```
blocks[43] = {
  type: 'timegraph',
  values: ['Usage', 'UsageDeliv'],
}
```

For a P1 meter it makes more sense to combine `Usage` and `UsageDeliv` into one graph. Dashticz recognizes the custom field name `NettUsage` for this

```
blocks[43] = {
  type: 'timegraph',
  values: ['NettUsage'],
}
```

If a `values` element is a string, then it's interpreted as the field name of the domoticz device.

To combine data from several devices use the following notation for the `values` block parameter

```
blocks['temps'] = {
  type: 'timegraph',
  duration: 600,
  height: '100px',
  xLabels: false,
  values: [
    { idx: 28, value: 'Temp', label: 'Boiler'},
    { idx: 31, value: 'Temp', label: 'Return'},
    { idx: 27, value: 'Temp', label: 'Woonkamer'},
    { idx: 659, value: 'Temp', label: 'Badkamer'}
  ]
}
```



In the previous example the `values` elements are defined as objects. You can use the following parameters for each `values` element:

Parameter	Description
idx	123: Domoticz device number
value	'Temp': The field name of the Domoticz device.
label	'My name': The label to use for this value. The default value is the device name followed by the field name.

Topbar

This module adds a topbar to Dashticz.

Default there will be the name, clock and settings-button presented in the Topbar. You can customize the Topbar with the following settings in `CONFIG.js`:

```
var columns = {}
columns['bar'] = {}
columns['bar']['blocks'] = ['logo', 'miniclock', 'settings']
```

Item	Description
logo	Title of Topbar (as defined in <code>config['app_title'] = 'Dashticz'</code> ;))
miniclock	Clock in Topbar
settings	Settings & Fullscreen button in Topbar

Applicable config-parameters from `CONFIG.js`:

Parameter	Description
app_title	Name of the Dashboard - Title to show in the <i>Topbar</i> 'Dashticz' = Show 'Dashticz' in the top bar
hide_topbar	0 / 1 Hide or Show <i>Topbar</i>

Complete example:

```
config['hide_topbar'] = 0;
config['app_title'] = 'Dashticz';

var columns = {}
columns['bar'] = {}
columns['bar']['blocks'] = ['logo', 'miniclock', 'settings']
```

Traffic info

With a traffic info block you can show the ANWB (Dutch) traffic info.

For public transport info see *Public Transport*.

A traffic info block can be configured as follows:

```
var trafficinfo = {}
trafficinfo.anwbA1 = {
  trafficJams: true,
  roadWorks: false,
  radars: false,
  road: 'A1',
  provider: 'anwb',
  show_lastupdate: true,
  icon: 'fas fa-car',
  width: 12,
  results: 100 };
```

segStart and segEnd can also be provided to filter the results even more.



Parameters

Parameter	Description
road	Name of the road(s) to show, comma seperated (Example: “A1, A73”)
title	Title of the block
show_lastupdate	<code>false</code> , <code>true</code> . To display the time of the last update.
provider	Traffic info provider to use. Choose from 'anwb' The Netherlands
icon	The font-awesome icon (including <code>fas fa-</code>) 'fas fa-car',...
refresh	time in seconds for refreshing the data
results	Number of results to show
width	To customize the width. It's not recommended to change the default value (12) because of the size of the output.
trafficJams	<code>false</code> , <code>true</code> . To show traffic jam info
roadWorks	<code>false</code> , <code>true</code> . To show road work info
radars	<code>false</code> , <code>true</code> . To show radar info
showempty	Control text to show in case of no traffic announcements <code>false</code> : Don't show a message in case of no traffic announcements <code>true</code> : Display default message in case of no traffic announcements '<text>': Display <text> in case of no traffic announcements
showemptyroads	Control text to show in case of no traffic announcements for a certain road (only applicable in combination with block parameter <code>road</code>) <code>false</code> : Don't show a message in case of no traffic announcements for a certain road. <code>true</code> : Display default message in case of no traffic announcements for a certain road. '<text>': Display <text> in case of no traffic announcements for a certain road.
url	'<url>': URL of the page to open in a popup frame or new window on click.
newwindow	0: open in current window 1: open in new window 2: open in new frame (default, to prevent a breaking change in default behavior) 3: no new window/frame (for intent handling, api calls). HTTP get request. 4: no new window/frame (for intent handling, api calls). HTTP post request. (forcerefresh not supported)

Styling

In case no info is available then the CSS class `empty` will be added to block. This can be used to adjust the styling of an empty block via `custom.css`

TV Guide



In `CONFIG.js` add:

```
var tvguide = {}  
tvguide.dutch = { key:'dutch', icon: 'fas fa-tv', width:12, channels: [1,2,3,4,31,46,  
↪92], maxitems: 10 }
```

And add the tvguide to a column with:

```
columns[4] = {  
  blocks: [tvguide.dutch]  
}
```

When you click on the TV Guide block a `www.tvgids.nl` popup will open (default). You can set your own url via the `url` parameter (optional).

TV Guide Parameters

Parameters	Description
title	Title of the TV Guide block.
width	1 . . 12 Width of the block.
layout	Shows or hides the channel name 0: Shows the channel name (=default) 1: Hides the channel name
key	'key': unique identifier.
icon	'fas fa-icon': icon to show in the TV Guide block. You choose an icon from the FontAwesome Free set.
image	'image.png': image to show as icon. Image path is relative to the <dashticz>/img folder.
maxitems	Maximum number of items to show.
url	'<url>': The web address of the page to open in the popup window when clicking the block.
channels	Selected channels.

The parameter `channels` contains an array of the selected channel IDs. Find the channel ID in the list below:

Table 5: Channel IDs

ID	Channel
1	NPO 1
2	NPO 2
3	NPO 3
4	RTL 4
5	Eén
6	Canvas
7	BBC 1
8	BBC 2
9	ARD
10	ZDF

Continued on next page

Table 5 – continued from previous page

ID	Channel
11	RTL
12	WDR Fernsehen
13	NDR Fernsehen
15	RTBF La 1
16	RTBF La 2
17	TV 5
18	National Geographic
19	Eurosport 1
21	Cartoon Network
24	Film 1 Premiere
25	MTV
26	CNN
28	Sat 1
29	Discovery Channel
31	RTL 5
32	TRT int.
34	Veronica
36	SBS 6
37	NET 5
38	ARTE
39	Film1 Family
40	AT 5
46	RTL 7
49	VTM
50	3Sat
58	PRO 7
59	2BE
60	VT4
64	NPO Zapp Xtra NPO Best
65	Animal Planet OUD
70	NPO Cultura
83	3voor12
86	BBC World
89	Nickelodeon
90	BVN
91	Comedy Central
92	RTL 8
99	Ziggo Sport Select
100	RTV Utrecht
101	RTV West
102	RTV Rijnmond
103	NH
104	BBC Entertainment
105	Private Spice
107	Film 1 Sundance
108	RTV Noord
109	Omrop Fryslân
110	RTV Drenthe
111	RTV Oost

Continued on next page

Table 5 – continued from previous page

ID	Channel
112	Omroep Gelderland
113	Omroep Flevoland
114	Omroep Brabant
115	L1 TV
116	Omroep Zeeland
148	Fox Sports Eredivisie
301	BBC 4
304	AMC
305	Discovery World
306	Discovery Science
308	3voor12 Central
309	3voor12 On Stage
310	3voor12 Portal
311	Disney XD
312	Nick Jr.
313	Boomerang
315	CBS Reality
317	Comedy Family
401	Playboy TV
403	Goed TV
404	FOXlife
406	Ons
407	OUTTV
408	RTL Lounge
409	Rtl crime
410	101 TV
411	Film1 Action
412	Film1 Premiere +1
413	HISTORY
414	Investigation discovery
415	Travel Channel
416	Nat Geo Wild
417	Extreme Sports Channel
419	Ziggo Sport Golf
420	Ziggo Sport Racing
422	Euronews
423	Al Jazeera Engels
424	Disney Channel
427	MTV Brand new
428	Brava NL
429	Oranje TV
430	Film1 Drama
434	Dusk
435	24 Kitchen
436	Eurosport 2
437	Comedy Central Extra
438	TLC
439	Animal Planet
440	Fox

Continued on next page

Table 5 – continued from previous page

ID	Channel
460	SBS 9
461	Pebble TV
462	Shorts TV
464	BBC First
465	RTL Z
466	Ziggo Sport
467	Spike
468	Fox Sport 2
469	Fox Sport 3
470	Fox Sport 4
471	KPN presenteert
472	Crime + Investigation
473	Viceland

Weather forecast

Note: In version 3.8.2 Dashticz switched to the new weather block as described here.

Dashticz supports the following weather forecast provider:

- Open Weather Map: <https://openweathermap.org/>

Before you can use the weather module, you must request an API key at <https://openweathermap.org/>

Open Weather Map

A basic weather block can be defined as follows:

```
blocks['weather'] = {
  type: 'weather',
  apikey: 'abc123...xyz', /Your OpenWeatherMap API key
  city: 'Amsterdam',
}
```



Besides the daily forecast, you can also show the current weather or an hourly forecast.

Parameters

Block parameter	Description
type	'weather'. To select a weather block
width	1..12: Width of the block
refresh	3600 Update once per hour. (default=3600, minimum=900, 15 minutes)
scale	Number between 0 and 1 to make the weather block smaller 1 Normal block width 0.5 50% width
apikey	'abc123...xyz'. OWM api key
city	'Amsterdam'. City name. You can also use the OWM city id code.
country	'nl'. Country code.
name	'My place'. Name to use instead of city name on the dashboard.
lang	'nl': Language to use for OWM data
layout	Choose a layout for the weather block 0: Daily forecast (=default) 1: Hourly forecast 2: Current weather 3: Current weather detailed 4: Combination of 2,3,0,1
count	5: Number of forecast items to show (default=3). Only for daily and hourly forecast.
interval	Use every n-th forecast item. 1. Use every forecast item (=default) 3. Set to 3 to get 3-hourly forecast
decimals	Number of temperature decimals to show 1 One decimal (=default)
showMin	Show/hide minimum temperature (only for daily forecast) false: Hide minimum temperature true: Show minimum temperature (=default)
showRain	Show/hide rain rate (only for daily and hourly forecast) false: Hide rain rate true: Show rain rate (=default)
showDescription	Show/hide weather description (only for daily and hourly forecast) false: Hide weather description
3.2. Blocks	true: Show weather description 169
showWind	Show/hide wind info and wind dial (daily and hourly forecast only)

The weather module makes use of the following CONFIG parameters:

Parameter	Description
<code>owm_api</code>	'<api-key>' API-key provided by https://openweathermap.org/
<code>owm_city</code>	Your city or nearby city to use in OWM. You can also fill in the city id here. 'Utrecht' '2748075'
<code>owm_name</code>	Name to use instead of city name 'Tuinwijk'
<code>owm_country</code>	Your country to use in OWM 'nl'
<code>owm_lang</code>	Set language for the description of the forecast (rain, cloudy, etc.). For available languages, see https://openweathermap.org/forecast5/#multi ' ' (empty string, default) Use Dashticz language setting
<code>owm_cnt</code>	Number of forecast elements (3-hour intervals or days) to show 1..5
<code>owm_min</code>	Show minimum temperature on 2nd row (only for daily forecast) false/true
<code>static_weathericons</code>	true Static weather settings false (default) Animated weather icons
<code>use_beaufort</code>	This config setting is used as default value for block parameter <code>useBeaufort</code> true Use Beaufort for wind speed false Use m/s for wind speed

Usage

In the next examples the config parameter `owm_api` and `owm_city` have been set globally, so they are not part of the weather block definitions.

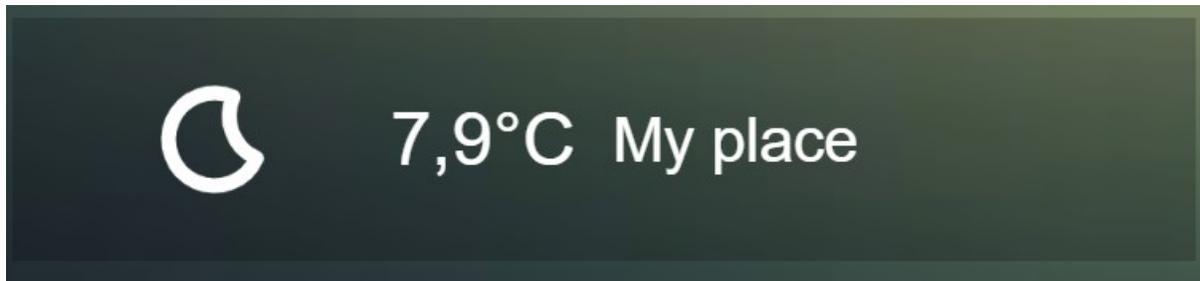
To show the hourly forecast with an 3 hour interval:

```
blocks['weather1'] = {
  type: 'weather',
  layout: 1,
  count: 7,
  interval: 3,
}
```



To show the current weather, with a custom name:

```
blocks['weather2'] = {
  type: 'weather',
  layout: 2,
  name: 'My place',
}
```



To show detailed info on the current weather:

```
blocks['weather3'] = {
  type: 'weather',
  layout: 3,
  name: 'Home is home',
}
```



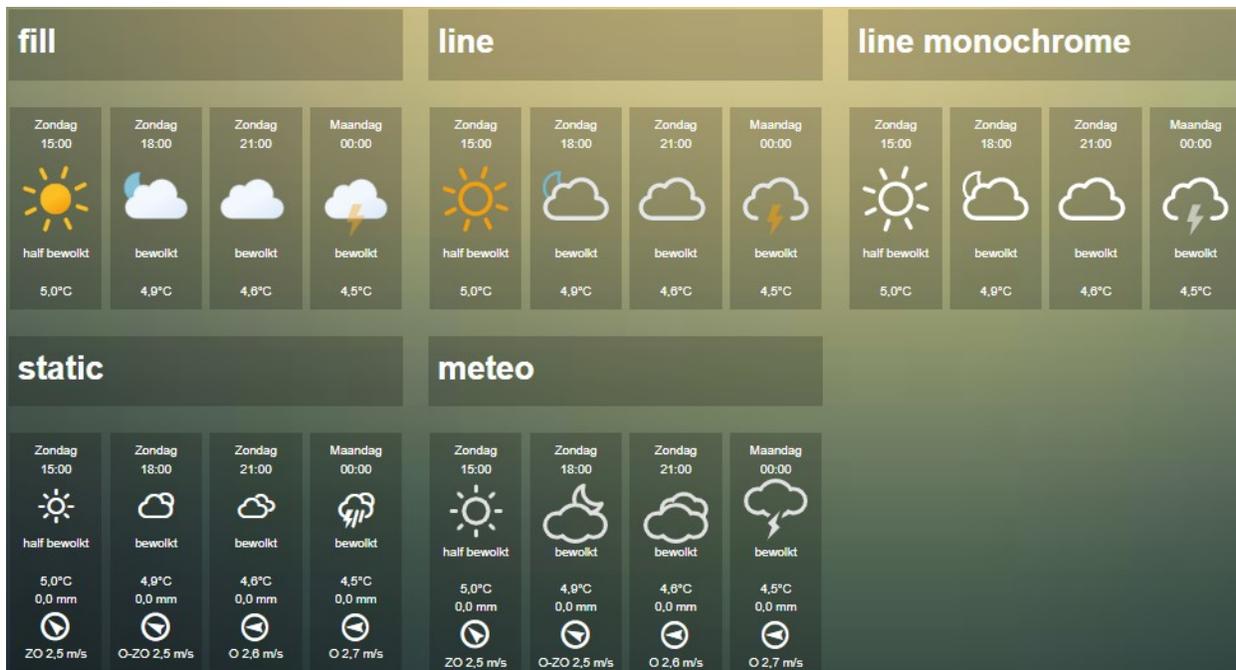
Icons

Via the block parameter `icons` you can choose one of the predefined icon sets:

- 'line' (=default)
- 'linestatic'
- 'fill'
- 'static'
- 'meteo'

By setting the block parameter `monochrome` to true the icons will be displayed as monochrome.

This will give the following icons sets to choose from:



styling

All blocks have the css class `weather` assigned in combination with `weather_0`, `weather_1`, ..., where the number indicates the layout number.

Further, all info items have css classes assigned. The names are self explanatory.

- `icon`: Weather icon
- `day`: Day item ('Saturday')
- `time`: Forecast time (hourly forecast only)
- `city`: City name
- `description`: Weather description
- `temp`: temperature
- `max`: Max temperature
- `min`: Min temperature
- `temp`: Current temperature
- `feels`: Feel-like temperature
- `rain`: Rain rate
- `humidity`
- `pressure`: Barometric pressure
- `windspeed`
- `windgust`
- `winddirection`

To capitalize the day of the week have to add the following code to `custom.css`:

```
.weather .day {
  text-transform: capitalize;
}
```

Ziggo Horizon Box

Note: Not tested

If you have a Ziggo Horizon Box you can add it to a column with:

```
columns[1]['blocks'] = ['horizon'];
```

Config Settings

Settings	Description
<code>switch_horizon</code>	'<url>' url of the Horizon box

3.3 Columns

The various examples in the documentation show how to add blocks to columns, and how to add columns to screens.

Besides the column-width there are no additional column parameters or settings.

One thing to mention is that's possible to reuse a column on more than one screen. This can be useful if for instance you want to have the same first column for all screens.

3.4 Screens

There is the ability to use multiple screens within Dashticz. Each screen can use it's own background. The background can also automatically change for the part of the day.

```
//if you want to use multiple screens, use the code below:

var screens = {}
screens[1] = {}
screens[1]['background'] = 'bg1.jpg';
screens[1]['background_morning'] = 'bg_morning.jpg';
screens[1]['background_noon'] = 'bg_noon.jpg';
screens[1]['background_afternoon'] = 'bg_afternoon.jpg';
screens[1]['background_night'] = 'bg_night.jpg';
screens[1]['columns'] = [1,2,3]

screens[2] = {}
screens[2]['background'] = 'bg3.jpg';
screens[2]['background_morning'] = 'bg_morning.jpg';
screens[2]['background_noon'] = 'bg_noon.jpg';
screens[2]['background_afternoon'] = 'bg_afternoon.jpg';
screens[2]['background_night'] = 'bg_night.jpg';
screens[2]['columns'] = [4,5,6]
```

3.4.1 Screen parameters

Parameter	Description
background	Defines the screen background - the image file must be in the <dashticz>/img folder 'bg1.jpg'
background_morning	Defines the screen background for morning (06:00-10:59) - the image file must be in the <dashticz>/img folder 'bg_morning.jpg'
background_noon	Defines the screen background for noon (11:00-15:59) - the image file must be in the <dashticz>/img folder 'bg_noon.jpg'
background_afternoon	Defines the screen background for afternoon (16:00-19:59) - the image file must be in the <dashticz>/img folder 'bg_afternoon.jpg'
background_night	Defines the screen background for night (20:00-05:59) - the image file must be in the <dashticz>/img folder 'bg_night.jpg'
columns	Defines which columns are shown on this screen [1, 2, 3]
auto_slide_page	Redefines the auto slide time as set in config['auto_slide_pages'] for this screen. 3: The time before auto slide to the next page is 3 seconds.

3.4.2 Usage

Layout per device

It is now possible to use another column/block setup per resolution.

To setup, use this code in config.js, change according your own needs:

```
var screens = {}
screens['default'] = {}
screens['default']['maxwidth'] = 1920;
```

(continues on next page)

(continued from previous page)

```
screens['default']['maxheight'] = 1080;

screens['default'][1] = {}
screens['default'][1]['background'] = 'bg9.jpg';
screens['default'][1]['columns'] = [1,2,4]

screens['default'][2] = {}
screens['default'][2]['background'] = 'bg9.jpg';
screens['default'][2]['columns'] = [5,6,7]

screens['tablet'] = {}
screens['tablet']['maxwidth'] = 1024;
screens['tablet']['maxheight'] = 768;
screens['tablet'][1] = {}
screens['tablet'][1]['background'] = 'bg9.jpg';
screens['tablet'][1]['columns'] = [3,1]

screens['tablet'][2] = {}
screens['tablet'][2]['background'] = 'bg9.jpg';
screens['tablet'][2]['columns'] = [2,4]
```

Note: If you are testing this on your laptop with resizing your browser window, refresh to rebuild the columns/blocks.

Standby Screen

There is the ability to let Dashticz go into standby mode. This defined with the `config['standby_after']` parameter in the `CONFIG.js` file. The screen get sort of grayed out and you can show items on the standby theme. These items **MUST** have been declared and used in the Dashboard:

```
config['standby_after'] = 5; //Enter standby mode after 5 minutes

var columns_standby = {}

columns_standby[1] = {}
columns_standby[1]['blocks'] = ['clock','currentweather_big','weather'] //specify_
↳blocks for the standby mode
columns_standby[1]['width'] = 12;
```

The following config settings are applicable to the standby screen:

Setting	Description
standby_after	Enter the amount of minutes 0 = No standby mode(default) 1 . . 1000 = Switch to standby after <value> minutes
standby_call_url'	[URL] Enter the url for adjusting the brightness when entering stand-by mode
standby_call_url_on_end	[URL] Enter the url for adjusting the brightness when exiting stand-by mode

Auto swipe, auto slide

Two auto swipe modes exist

1. Auto swipe back to a specific screen (default)
2. Auto slide to the next screen

The 'swipe back' mode is selected by setting `config['auto_swipe_back_after']` to non zero. The 'next screen' mode is selected by setting `config['auto_slide_pages']` to non zero.

The initial delay before starting 'next screen' mode, can be set via `config['auto_swipe_back_after']`.

The default timeout which is used for each screen in 'next screen' mode can be defined by `config['auto_slide_pages']`. However, you can overrule this for each screen by adding the `auto_slide_page` parameter to the screen block. In case the screen parameter `auto_slide_page` is 0, then this screen will be skipped during auto slide.

All timeouts (`auto_swipe_back_after`, `auto_slide_pages`, `auto_slide_page`) are defined in seconds.

The auto swipe countdown timer will reset after mouse moves and screen touches.

3.4.3 Styling

If you want to be able to scroll the screen vertically add the following to `custom.css`:

```
.swiper-slide {
  overflow: auto!important
}
```


4.1 Styling via custom.css

Note: In Dashticz v3.2.0 the css-classes for most of the special blocks have been updated. See *Special blocks*

Contents

- *Styling via custom.css*
 - *Introduction into CSS*
 - *Domoticz blocks*
 - * *Used classes for Domoticz blocks*
 - * *Background color*
 - * *Styling of lastupdate text*
 - * *Icon colors of a Domoticz switch*
 - *Special blocks*
 - * *CSS class definition for special blocks*
 - * *Block titles*
 - * *Font Size*
 - * *Smaller Title blocks (Height)*
 - * *Button: Render the title below the icon (all buttons)*
 - * *Render the title below the icon (specific button)*
 - *Generic block related*

- * *Hover background color*
- * *Reduced space around blocks*
- * *Rounded corners*
- *Icons*
 - * *Larger (Bulb) icons*
 - * *All Icons on the Dashboard Larger*
 - * *Larger Logitech Media Server buttons*
- *Fonts & Text Size*
 - * *Change font size of 1 specific (text) device*
 - * *Change font size of public transport module*
 - * *Fontsize Trashcan Module*
- *Color & Transparency*
 - * *Transparent Buttons Thermostat*
 - * *Colored Lightbulbs*
- *Lightbulbs color & Opacity*
- *Miscellaneous*
 - * *Hide block*
 - * *Remove Swiper Pagination Bullet*
 - * *Remove break line*
 - * *Customized drop down block*
 - * *Change size and color of Standby Screen items*
 - * *Prevent click handling of a block*
- *Popup windows*

Almost all visual elements on the Dashticz dashboard are styled via so called style sheets. Normally the abbreviation CSS (Cascading Style Sheets) is used for this. The source files that define the style information also have the `.css` extension.

There are a lot of creative users on the Domoticz Forum, that modify the CSS. A lot of examples can be found on the forum. Some examples will be summarized here, which you can use as a starting point to customize your own dashboard.

The default styling is defined in the file `<dashticz>/css/creative.css`. While you can use this file for inspiration, you should not modify it, since the default styling can be modified in the file `custom.css`, located in the folder `<dashticz v3>/custom`

TIP: Comments can be used to explain the code, and may help when you edit the source code at a later date. A CSS comment starts with `/*` and ends with `*/`. Comments can also span multiple lines:

```
/* This is a single-line comment */
```

This sections has some examples of CSS that can be placed in `custom.css`, so you can create your own look and feel.

4.1.1 Introduction into CSS

A lot of CSS tutorials are available online. The first google result: <https://www.w3schools.com/css/>

A CSS style definition consists of two parts.

1. Selector
2. Style modifier

With the selector part you select certain elements on the Dashboard. With the style modifier part the visual appearance can be defined.

Almost all Dashticz elements on the dashboard have one or more class definitions associated with it. For instance most blocks have the class `transbg` associated with it. As an example, You can use this class name to change the background colors for all blocks at once. Add the following to `custom.css`:

```
.transbg {
  background-color: red !important;
}
```

In this example the selector element is `.transbg`. With this selector all elements with the class `transbg` are selected. On the selection we apply a new `background-color` style.

In some cases, also in the example above, the statement `!important` needs to be added, to enforce that this style setting will overrule other style settings that have been defined by other style modifiers.

If you want to change only the background of one specific block you have to narrow the selector. For this you can use the `data-id` parameter. All blocks that you have defined with `blocks[idx]` will have the `data-id` parameter with value `'idx'` attached to it:

```
[data-id='120'] {
  background-color: red !important;
}

[data-id='120'] {
  background-color: red !important;
}
```

All blocks on the dashboard have a unique id, which are sequentially numbered. How to find the block id will be explained later. Assuming the block you want to change has block id 3 then add the following to `custom.css`:

```
#block_3 .transbg {
  background-color: red !important;
}
```

This means: Change the background color to red for the elements with the class `transbg` associated with it within the block with the id `block_3`.

So remember, blocks can have classes, parameters and id's associated with them. Blocks are selected by choosing the right class, parameter, and/or id.

If you right-click on a block, and select `Inspect` you can see the assigned classes in DevTools.

4.1.2 Domoticz blocks

Used classes for Domoticz blocks

Example block definition:

```
blocks[120] = {  
  width: 6  
}
```

Placed in a column it can give the following result:



The whole block has class `block_120` First line: `title` Second line: `state` Third line: `lastupdate`

Besides the specific block label `.block_120` the whole block will also contain the generic css class label `.mh`.

In case the Domoticz device contains subdevices, like a TempHumBar device, three devices will be created. In this case instead of `.block_120` the labels `.block_120_1`, `.block_120_2`` and ```.block_120_3` will be used.

If you have used a specific blocks key in combination with the `idx` parameter, the key label will be used as CSS class label as well, like this:

```
blocks['mydevice'] = {  
  idx: 120,  
  width: 6  
}
```

This block will have the CSS class label `.block_mydevice`. Again, if device 120 has subdevices, the following CSS classes will be assigned: `.block_mydevice_1`, `.block_mydevice_2` and `.block_mydevice_3`.

Last variation: A specific blocks key in combination with a specific subdevice:

```
blocks['mydevice'] = {  
  idx: '120_2',  
  width: 6  
}
```

This block will have the CSS class label `.block_mydevice`

Background color

To change the background color of all Domoticz blocks:

```
div[class*='block_'] {  
  background-color: red !important;  
}
```

To change the height of only this block:

```
.block_120 {  
  height: 150px !important;  
}
```

Styling of lastupdate text

To change the font-size and color of the lastupdate text of this block:

```
.block_120 .lastupdate {
  font-size: 20px;
  color: blue;
}
```

Icon colors of a Domoticz switch

To change the icon colors for only this block:

```
.block_120 .on {
  color:#F1C300;
}

.block_120 .off {
  color:#fff;
}
```

In the previous example you can see the `on` class or `off` class can be used to select a block depending on the state of the Domoticz device.

4.1.3 Special blocks

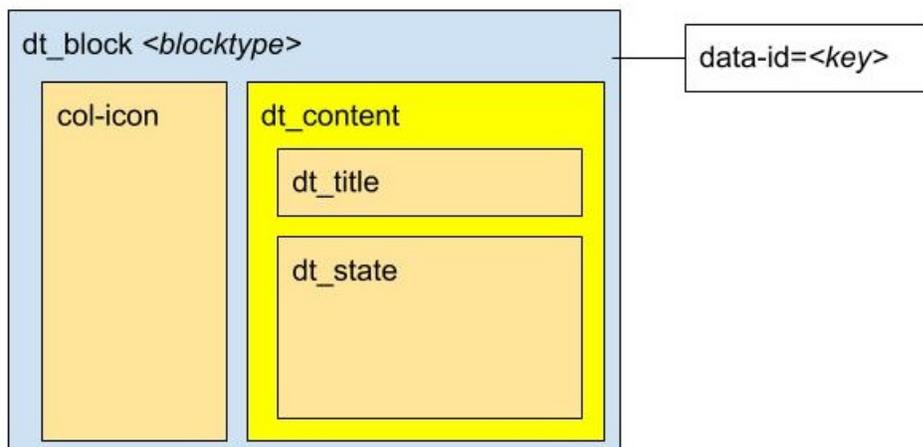
CSS class definition for special blocks

The CSS class approach for special blocks are slightly different as the Domoticz blocks. Also not all special blocks have been transformed to this template yet. It's applicable to the following blocks:

- alarmmeldingen, blocktitle, button, calendar, camera, coronavirus, dial, frame, graph, longfonds, news, nzbget, publictransport, sepanel, stationclock, streamplayer, traffic, trafficinfo

Currently it's not applicable to:

- coins, garbage, sonarr, spotify, weather_owm, weather



Each top level block has the class `dt_block` and the name of block type as class assigned. If you have defined this block via `blocks['mykey']=...` then the value of the `data-id` parameter will be set to `'mykey'`. If you have

defined the block by using an object, like `buttons.buienradar=` then you can define the key by making use of the key-parameter in your block definition.

So if you want to select all blocktitles, add the following to custom.css:

```
.blocktitle {
  background: blue !important;
}
```

If you want to change the title part of all blocktitles:

```
.blocktitle .dt_title {
  font-size: 50px;
  color: red;
}
```

If you want to change only a specific blocktitle:

```
[data-id='title1'].blocktitle {
  background: yellow !important;
}
```

Block titles

Example block definition:

```
blocks['myblocktitle'] = {
  type: 'blocktitle',
  title: 'My Devices Block'
}
```

To select all the blocktitles and change the background color:

```
.blocktitle {background-color: gray !important;}
```

To change the background color for only this block title:

```
.dt_block[data-id='myblocktitle'] {background-color: gray !important;}
```

As you can see in the previous example we use the generic block selector `dt_block` having the value `myblocktitle` for the parameter `data-id`. This is the generic way to select a specific special block.

Font Size

To change the font size of this block title:

```
.dt_block[data-id='myblocktitle'] .dt_title {
  font-size: 30px;
}
```

Smaller Title blocks (Height)

```
.blocktitle {
  height: 60px !important;           /* default height=75px */
  padding-top: 3px !important;       /* center text for new height */
}
```

Button: Render the title below the icon (all buttons)

```
.button {
  flex-direction: column !important;
  min-height: 85px;
}
```

A Domoticz device block normally has a height of 85 pixels (small devices: 75 pixels).

Render the title below the icon (specific button)

You have to add the key parameter to your button definition in CONFIG.js:

```
buttons.mybutton = {
  key: "mykey",
  icon: "fas fa-newspaper",
  title: "newspaper"
}
```

And then add the following to custom.css:

```
.button[data-id='mykey'] {
  flex-direction: column !important;
}
```

4.1.4 Generic block related

Hover background color

```
.transbg.hover.mh:hover { background-color: red;}
```

Reduced space around blocks

To make the space between all blocks smaller:

```
.transbg[class*="col-xs"] {
  border: 3px solid rgba(255,255,255,0);           /* border: 7px -> 3px - Smaller_
↳space between blocks */
}
```

Rounded corners

Rounded corners for all blocks:

```
.transbg[class*="col-xs"] {  
  border-radius: 20px;                               /* Rounded corners */  
}
```

4.1.5 Icons

Larger (Bulb) icons

```
.far.fa-lightbulb:before{  
  font-size: 24px;  
}  
  
.fas.fa-lightbulb:before{  
  font-size: 24px;  
}
```

All Icons on the Dashboard Larger

To make all icons on the Dashboard larger in one move, just simple add (choose font-size wisely!!):

```
.far, .fas, .wi {  
  font-size:24px !important;  
}
```

Larger Logitech Media Server buttons

```
.fas.fa-arrow-circle-left {  
  font-size: 50px !important;  
}  
.fas.fa-stop-circle {  
  font-size: 50px !important;  
}  
.fas.fa-play-circle {  
  font-size: 50px !important;  
}  
.fas.fa-arrow-circle-right {  
  font-size: 50px !important;  
}  
.fas.fa-pause-circle {  
  font-size: 50px !important;  
}
```

4.1.6 Fonts & Text Size

Change font size of 1 specific (text) device

Every block has an unique identifier-classname, which look something like “.block_xxx” (where xxx is the idx of your choice) that can be used in css. Example:

```
.block_233 {
  font-size:120px !important;
  color:red !important;
}
```

Change font size of public transport module

```
.publictransport div {
  font-size: 13px;
}
```

Fontsize Trashcan Module

```
.trash .state div.trashrow {
  font-size: 12px;
}

.trash .state div.trashtoday {
  font-size: 16px;
}

.trash .state div.trashtomorrow {
  font-size: 14px;
}
```

4.1.7 Color & Transparency

Transparent Buttons Thermostat

```
.input-groupBtn .btn-number {
  opacity: 0.5;
  color: white;
  background-color: rgb(34, 34, 34);
  border-radius: 0px;
  padding: 6px 10px 6px 10px;
  line-height: 20px;
  background-color: transparent;
}
```

Colored Lightbulbs

It is possible to use colors for the bulb-icons. In `custom.css` add something like:

```
.fas.fa-lightbulb {
  color:#F1C300;
}
.far.fa-lightbulb {
  color:#fff;
}
```

Result:



4.1.8 Lightbulbs color & Opacity



- Color: green
- Opacity: 0.4

```
.fas.fa-lightbulb {  
  color: rgba(0,255,0,0.4)  
}
```

4.1.9 Miscellaneous

Hide block

```
div[data-id='myblock'] {  
  display: none  
}
```

Change 'myblock' to your own block name

Remove Swiper Pagination Bullet

```
.swiper-pagination-bullet {  
  display: none !important  
}
```

Remove break line

```
.block_107 br:nth-child(2) {  
  display: none  
}
```

Change 107 to your own block number

Customized drop down block



```
.block_438 .icon {
  width: 85%;
  height: auto;
}
.block_438 .title,
.block_438 br:nth-child(2) {
  display: none;
}
.block_438 .col-data select {
  font-size: 150%;
  width: 100%;
  background-color: rgb(242,242,242);
}
```

Change 438 to your own block number

Change size and color of Standby Screen items

```
.standby .clock{
  font-size:250px !important;
  color: #718084!important;
}
.standby .weekday,
.standby .date {
  font-size:80px !important;
  color: #4E585B !important;
}
```

Prevent click handling of a block

For some blocks you may want to disable click handling, for instance for the news block, or for the frame content. This section describes how this may be achieved.

Assume you've defined your block in CONFIG.js via:

```
blocks['news'] = {
  ...
}
```

then you can disable handling of click events by adding the following to custom.css:

```
[data-id="news"].dt_block {
  pointer-events: none
}
```

or, add the addClass block parameter:

```
blocks['news'] = {
  addClass: 'noclick',
  ....
}
```

and add the following to custom.css:

```
.noclick {
  pointer-events: none
}
```

Both methods also work for frames.

4.1.10 Popup windows

Popup windows have the following class attached to it: `modal-dialog-custom`.

The popup window contains a div with the class `modal-content`. Depending on the popup type, the following classes will be applied as well:

- `modal-url` For an url opened in a popup window
- `modal-graph` For a graph opened in a popup window
- `modal-popup` For a popup created from the `popup` block parameter, except when the `popup` parameter refers to a graph block. In the latter case, the `modal-graph` parameter will be applied.

A `modal-url` popup window, will have a white background, white border, and black 'close' button.

The other popup windows will have a black background, and a white 'close' button.

The default styling is a black background, with white 'close' button.

As an example, to give url-popups a green background with a red close button:

```
.modal-content.modal-url {
  background-color: green
}

.modal-content.modal-url .close {
  color: red;
  opacity: 1;
}
```

4.2 Functionality via custom.js

There is the possibility to use your own functions in Dashticz. For this you can edit the file `<dashticz folder>/custom/custom.js`.

4.2.1 function afterGetDevices()

This predefined function will be called after every update of a Domoticz device.

You can enter code inside this function which you want to be called.

Of course, you can also use stuff like `$(document).ready()` etc...

The following example shows how you can change the styling of a Domoticz device based on the status:

```
function afterGetDevices(){
  if (Domoticz.getAllDevices()[120].Data == 'Off') {
    $('.block_120 .title').addClass('warningblue');
    $('.block_120 .state').addClass('warningblue');
  }
  else {
    $('.block_120 .title').removeClass('warningblue');
    $('.block_120 .state').removeClass('warningblue');
  }
}
```

In this example the CSS style 'warningblue' is applied to the title and state part of Domoticz block 120 if the state is 'Off' (as used by switches). For this example to work you must also add the definition for warningblue to custom.css. For instance as follows:

```
.warningblue {
  color: blue !important;
}
```

4.2.2 function getExtendedBlockTypes(blocktypes)

Some blocktypes are filtered out by their distinct name and therefore will not produce the nice icons. Referring to blocktypes.Name = {} section in blocks.js. Add the following function to show the icons and data-layout for blocks with your own names.

```
function getExtendedBlockTypes(blocktypes){
  blocktypes.Name['Maan op'] = { icon: 'fa-moon-o', title: '<Name>', value: '<Data>' }
  blocktypes.Name['Maan onder'] = { icon: 'fa-moon-o', title: '<Name>', value: '<Data>' }
  return blocktypes;
}
```

4.2.3 function getBlock_IDX(block)

Want your block to show up differently then Dashticz generates and do you have a little bit of coding skills? Add to custom.js one of the examples:

```
function getBlock_233(block){ //change 233 to the idx of your device!
  var idx = block.idx; //the Dashticz id
  var device = block.device; //The Domoticz device info
  var $mountPoint = block.$mountPoint // The DOM entry point for this block

  $mountPoint.find('.mh') //Find the correct block
    .off('click') //remove
  any previous click handler
    .click(function() { //install new click
  handler
    switchDevice(block, 'toggle', false); //switch the device on click
  })
    .addClass('hover'); //and add the
  predefined hover css class
```

(continues on next page)

(continued from previous page)

```

var html='';
html+='<div class="col-xs-4 col-icon">';
  if(device['Status']=='Off') html+=iconORimage(block,'fas fa-toggle-off','','off_
↪icon');
  else html+=iconORimage(block,'fas fa-toggle-on','','on icon');
html+='</div>';
html+='<div class="col-xs-8 col-data">';
html+='<strong class="title">'+device['Name']+</strong><br />';
if(device['Status']=='Off') html+='<span class="state">AFWEZIG</span>';
else html+='<span class="state">AANWEZIG</span>';

if(showUpdateInformation(block)) html+='<br /><span class="lastupdate">
↪'+moment(device['LastUpdate']).format(settings['timeformat'])+'</span>';
html+='</div>';
return html;
}

```

4.2.4 function getStatus_IDX(block)

This function is called when a Domoticz device is updated. You can use this function for instance to set specific block parameters depending on the value of the device.

Example, change the icon based on the device value (in this case device 413, the first subdevice):

```

function getStatus_413_1(block) {
  var usage = block.device.Usage;
  if (parseFloat(usage) > 0) {
    block.icon = 'fas fa-sun slow-spin'
  } else {
    block.icon = 'fas fa-sun';
  }
}

```

Example, add a red background to a switch when energy usage reaches a limit:

```

function getStatus_145(block) {
var idx = block.idx;
var device = block.device;
  if(parseFloat(device['Data'])>23){
    block.addClass='warning';
  }
  else {
    block.addClass='';
  }
}

```

And in custom.css add your css, according to this example:

```

.warning {
  background: rgba(199,44,44,0.3) !important;
  background-clip: padding-box;
}

```

Or if you like a blinking version:

```
.warning {
  background: rgba(199,44,44,0.3) !important;
  background-clip: padding-box;
  border: 7px solid rgba(255,255,255,0);
  -webkit-animation: BLINK-ANIMATION 1s infinite;
  -moz-animation: BLINK-ANIMATION 1s infinite;
  -o-animation: BLINK-ANIMATION 1s infinite;
  animation: BLINK-ANIMATION 1s infinite;
}

@-webkit-keyframes BLINK-ANIMATION {
  0%, 49% {
    background-color: rgba(199,44,44,0.3);
    background-clip: padding-box;
    border: 7px solid rgba(255,255,255,0);
  }
  50%, 100% {
    background-color: rgba(199,44,44,0.7);
    background-clip: padding-box;
    border: 7px solid rgba(255,255,255,0);
  }
}
```

Example, change the block background color to the setted light RGB color:

```
function getStatus_20(block){
  var color = JSON.parse(block.device.Color);
  if (block.device.Data!='Off') {
    var colorStr = `rgb(${color.r},${color.g},${color.b})`;
    $(block.mountPoint + ' > div').css('background-color', colorStr)
  }
  else
    $(block.mountPoint + ' > div').css('background-color', '')
}
```

In case you have defined a block with a custom key name in combination with the `idx` parameter, then the key name will be used in the function call. Example: You use the following block definition:

```
blocks['myblock'] = {
  idx: 145
}
```

In the previous example the following function in `custom.js` will be called: `function getStatus_myblock(block)` (and not `function getStatus_145(block)`)

Example, Trigger on different text values:

```
function getStatus_14146(block){ // where 14146 is your block IDX
  var idx = block.idx;
  var device = block.device;
  var tekst = device['Data'];

  if(device['Data']=="No Data"){
    block.addClass='none'
    block.icon = 'far fa-thumbs-up'
  }
  // geen actie
```

(continues on next page)

(continued from previous page)

```

else if (tekst.includes('geen waarschuwingen')){
    block.addClass='none'
    block.icon = 'far fa-thumbs-up'
    block.title = 'Geen Melding'
}

// zware windstoten
else if (tekst.includes('Kans op zware windstoten')){
    block.addClass='warningyellow'
    block.icon = 'fas fa-wind'
    block.title = 'Wind'
}
}

```



function getChange_IDX(block)

This function gets called when the value of a Domoticz device changes. This function will only get called after updating the block. If you want to change the block definition as a result of the status you should use the getStatus function as described above.

Change value of another block

By calling `Dashticz.setBlock` from the `getStatus` function you can change another block as well. Example:

```

function getStatus_2(block) {
    var idx = block.idx;
    var device = block.device;
    console.log(device.Level)
    if (parseFloat(device.Level) === 0) {
        block.title='level 0';
        block.icon='fas fa-train';

        Dashticz.setBlock('mytitle', {
            title: 'also 0',
            icon: 'fas fa-train'
        });
    }
    else {
        block.title='level is not 0 but ' + device.Level;
        block.icon="fas fa-bus";

        Dashticz.setBlock('mytitle', {

```

(continues on next page)

(continued from previous page)

```
        title: 'not 0, but ' + device.Level,
        icon: 'fas fa-bus
    });
}
}
```

The `getChange_2` function gets called when the data of device with index 2 changes.

This previous example will also change a block that is defined by `blocks['mytitle']` (for instance a blocktitle):

```
blocks['mytitle'] = {
  type: 'blocktitle',
  title: 'Default',
  icon: 'fas fa-car'
}
```

4.2.5 Device hook

On every device update the device hook is called first. You can use the device hook to manipulate device data before the update is send to the Dashticz blocks.

Example:

```
function deviceHook(device) {
  if (device.idx==43) {
    device.NettCalc = parseFloat(device.Usage) - parseFloat(device.UsageDeliv)
    device.deviceStatus = device.NettCalc >= 0 ? 'positive':'negative';
  }
}
```

The device parameter contains the device data as received from Domoticz. In the previous example the device field 'NettCalc' is calculated, which can be used in for instance a dial.

You can also call `Domoticz.setBlock` functions from the device hook.

You could do something similar in the `getStatus` functions, but then it's required that a block is defined that uses that specific device. For the device hook function that's not necessary: It will be called on any device update received from Domoticz.

One special functionality is the device field `deviceStatus`. You can set this field to a certain CSS class name. This CSS class name will be applied to the Dashticz blocks that use this device. You could set the value of `deviceStatus` based on the current device value or based on the value of another device.

Happy hacking :)

Various tips and tricks will be collected here.

5.1 Dashticz security

Dashticz is not secure with its default installation. If you want to have access to Dashticz from outside your local network, you should use a VPN connection, or enable user authentication for the webserver you use.

If you are using Apache you can enable user authentication as follows:

5.1.1 Step 1

First create a credentials file preferably in a location not accessible via your webbrowser, for instance in `/home/pi/dashticzpasswd`

In the terminal window type the following:

```
htpasswd -c /home/pi/dashticzpasswd admin
```

and choose a password. In this example you add the user 'admin'. You can choose a different user name of course.

5.1.2 Step 2

In your dashticz folder create a `.htaccess` file:

```
cd /home/pi/dashticz  
nano .htaccess
```

with the following content:

```
AuthUserFile /home/pi/dashticzpasswd
AuthName "Please Enter Password"
AuthType Basic
Require valid-user
```

In the first line use the location of your credentials file you created in the first step.

If you want to be able to login from your local network without user/password authentication, then use the following for `.htaccess`:

```
AuthUserFile /home/pi/dashticzpasswd
AuthName "Please Enter Password"
AuthType Basic
<RequireAny>
    Require valid-user
    Require ip 192.168.1
</RequireAny>
```

Replace 192.168.1 with your own subnet.

Note: If you are using a reverse proxy in your local network to access Dashticz, then the Apache server thinks the traffic comes from your local network, and will give access without asking for a password. See method 2 below for an alternative setup.

5.1.3 Step 3

Enable the usage of local `.htaccess` files in Apache.

This step depends a bit on your Apache configuration. For a default Linux installation:

```
cd /etc/apache2
sudo nano apache2.conf
```

Now look for your web root folder. In the default setup this is `/var/www/`. In the `apache2.conf` file look for:

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

Replace `AllowOverride None` with `AllowOverride All`, so you should have:

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

Save the file and restart Apache:

```
sudo service apache2 restart
```

Now if you browse to Dashticz you get a prompt to enter your login credentials.

5.2 Dashticz security (method 2)

You can choose a specific port for serving Dashticz. Then you can choose to only expose this Dashticz port to the outside world, for instance via a reverse proxy. In my situation I have a reverse proxy on my NAS, that forwards a certain incoming url to the dedicated Dashticz port on my Dashticz server.

5.2.1 Step 1

First create a credentials file preferably in a location not accessible via your webbrowser, for instance in `/home/pi/dashticzpasswd`

In the terminal window type the following:

```
htpasswd -c /home/pi/dashticzpasswd admin
```

and choose a password. In this example you add the user 'admin'. You can choose a different user name of course.

5.2.2 Step 2

Create a new Apache2 configuration:

```
cd /etc/apache2/conf-available
sudo nano dashticz.conf
```

Add the following content to the `dashticz.conf` file:

```
Listen 8081
<VirtualHost *:8081>
    DocumentRoot "/home/pi/dashticz"
</VirtualHost>

<Directory "/home/pi/dashticz">
    AuthUserFile /home/pi/dashticzpasswd
    AuthName "Dashticz Password"
    AuthType Basic
    <RequireAny>
        Require valid-user
        <RequireAll>
            Require ip 192.168.1
            Require not ip 192.168.1.16
        </RequireAll>
    </RequireAny>
</Directory>
```

With the previous example the folder `/home/pi/dashticz` will be served on port 8081. Choose/check your own folder and (available) port. Apache will ask for username/password, except from your local network (192.168.1.xxx). On my system the reverse proxy runs on 192.168.1.16. If the traffic is coming from that IP address, then the user must be authenticated.

5.2.3 Step 3

```
#Enable configuration
sudo a2enconf dashticz.conf
#Reload apache2
sudo service apache2 reload
```

5.3 Use of Web Fonts

Add the following to custom.js:

```
$('<link rel="stylesheet" type="text/css" href="//fonts.googleapis.com/css?
↪family=Orbitron" />').appendTo('head');
```

Add the following to custom.css:

```
.webfont {
  font-family : orbitron;
}
```

5.4 Changing alert icon colors dynamically

Assumptions:

- Today alert IDX in Domoticz=115 (find your own IDX and replace in the code below)
- Level grades (as defined in Domoticz): Level 1 - normal (no alert, GREEN), Level 2 - Light warning (YELLOW), Level 3 - Warning (ORANGE), Level 4 - Critical (RED).

Add the following to custom.js:

```
function getStatus_115(block) {
  var device=block.device;
  if(device['Level']==1) {
    block.addClass='alertnormal';
  }
  else if (device['Level']==2) {
    block.addClass='alertlight';
  }
  else if (device['Level']==3) {
    block.addClass='alertmedium';
  }
  else {
    block.addClass='alerthigh';
  }
}
```

Add the following to custom.css:

```
.alertnormal .col-icon {
  color: green !important;
}
.alertlight .col-icon {
  color: yellow !important;
}
```

(continues on next page)

(continued from previous page)

```
.alertmedium .col-icon {  
    color: orange !important;  
}  
.alerthigh .col-icon {  
    color: red !important;  
}
```


6.1 Upgrade instructions

6.1.1 v3.8.9: Dial styling

In previous versions you had to change the dial ring coloring via:

```
.dial .bar.primary,  
.dial .fill.primary {  
    border-color: #d9e900 !important;  
}  
  
.dial .bar.secondary,  
.dial .fill.secondary {  
    border-color: #26e500 !important;  
}
```

From v3.8.9 you have to rewrite this as:

```
.slice.primary {  
    color: #d9e900  
}  
  
.slice.secondary {  
    color: #26e500  
}
```

The same counts for dial rings labeled as positive or negative

Also see Blinds *Custom Styling*.

6.1.2 v3.7.6: Dial values

In v3.7.6 there are some changes in the behavior of the `values` parameter for a dial block.

The values defined in this parameter will be added just below the main value of a dial: By default the main value will be displayed as well.

To hide the main value, add `showvalue: false` to the dial block definition.

6.1.3 v3.7.3: Popup blocks and more

Note: Make a backup of the files in the custom folder when updating!

Main changes in v3.7.3 are the following:

- Improved consistency in popup windows
- Possibility to add one or more Dashticz blocks into a popup window

Both points required some code redesign. Also some cleanup actions were implemented. The main changes are summarized below.

New blocks for 'moon' and 'log'

Both blocks now can be added with there name, or by setting the type parameter. Example:

```
blocks['moon'] = {
  width:6
}

blocks['logexample'] = {
  type: 'log'
}

columns[1] = {
  blocks: ['moon', 'logexample']
}
```

The previous method to define 'moon' via the `btnimage` parameter value `moon` is not supported anymore.

The 'log' parameter of a button is not supported anymore.

Removal of 'old' calendar block

Until now Dashticz supported the 'old' and the 'new' calendar block. The 'old' calendar block has been removed.

See `newcalendar`

To be able to recognize a calendar block you have to add `icalurl` parameter to the block definition.

Styling of popup windows

The CSS classes and default styling for popup windows changed. So if you have some custom popup styling in your `custom.css` then probably you have to update this.

See *Popup windows* for additional information.

Popup windows new functionality

In the following situation a block click will trigger a new window:

- If the block is a Domoticz block (but not a switch), and the 'graph' parameter is not `false`, then Dashticz will open a popup graph.
- The block contains an url parameter. Dashticz will open the url. The `newwindow` block parameter determines how the url will be opened.
- The block contains a `popup` parameter. The `popup` parameter refers to a block parameter. Dashticz will open the block indicated by the `popup` parameter in a popup window.

With the `newwindow` block parameter you indicate how the popup will be opened. New option '5' has been added to open the url in a new browser tab.

The 'auto_close' parameter will work on popups with `newwindow` value 1 (=new window), 2 (=modal popup) and 5 (=new tab)

New block parameter blocks

With the 'blocks' parameter a block acts in fact like a column definition. Example:

```
blocks['mypopup'] = {
  blocks: [1,2,'mydial','graph_3', 'log']
}
```

In the previous example a block with the name 'mypopup' is created, consisting of two Domoticz devices, a dial, a graph, and a log block.

You can add 'mypopup' to a column, or use it as value for a `popup` parameter:

```
blocks[123] = {
  popup: 'mypopup'
}
```

In this example a block for Domoticz device 123 will have a popup block consisting of 5 other blocks.

6.1.4 v3.4.9: Custom keys in block definition

Note: Breaking changes. See notes below.

In case you have defined a block with a custom key as follows:

```
blocks['myblock'] = {
  idx: 123,
  width:4
}
```

then this will have the following consequences:

- The block will have the CSS class `.block_myblock` and not `.block_123`. You may have to change something in `custom.css`

- If device 123 has subdevices, they will receive the CSS class `.block_myblock_1` etc. and not `.block_123_1`
- On device change the function `getStatus_myblock(block)` in `custom.js` will be called, and not `getStatus_123(block)`

6.1.5 v3.4.1: Redesign block definition

Note: Breaking changes. See notes below.

Although not so visible from the outside v3.4.1 contains a significant design update on how Dashticz blocks are managed internally.

Some advantages:

- You can use the same Domoticz ID with different block settings:

```
blocks['first_one'] = {
  idx: 123,
  width:4
}

blocks['second'] = {
  idx: 123,
  width:12
}

columns[1]['blocks'] = [
  'first_one', 'second'
]
```

This will display Domoticz device <123> twice: the first one having width 4, the second one having width 12.

- Inline block definition:

```
columns[1]['blocks'] = [
  { idx: 123, width: 4},
  { idx: 123, width: 12}
]
```

This will give the same results as the previous example.

- The `on` and `off` classes will be added automatically to a switch-block. This enables auto-hide possibility in combination with the `addClass` parameter

```
blocks[123] = {
  addClass: 'myClass'
}
```

And in `custom.css`:

```
.myClass.off {
  display: none
}
```

Block parameters

Note: Breaking change: refresh interval parameter

Setting the refresh interval of a block now has been standardized to the `refresh` parameter. You may have to replace `interval` by `refresh` for some blocks. The value represents the refresh interval in seconds.

A new block parameter:

Parameter	Description
<code>addClass</code>	The CSS class name, that will be added to the block.

custom.js

Note: Breaking change: `getBlock`, `getStatus` and `getChange` interface in `custom.js`

There are changes in the function parameters for the following functions:

- `getBlock_IDX`
- `getStatus_IDX`
- `getChange_IDX`

These functions now receive `block` as parameter, containing all relevant parameters for the block.

The examples in *Functionality via `custom.js`* have been updated.

6.1.6 v3.2.0: Change in HTML template for special blocks

In the last years several special blocks have been added. However, the html templates for each block all were a bit different. The usage of css class names was not consistent. This made maintainability difficult as well. For that reason I decided to refactor the code towards the usage of a standard html template. In this version the following blocks have been migrated to the new format:

- Block title
- Buttons
- Frames
- Longfonds
- News
- Public transport
- Stationclock
- Streamplayer
- Traffic
- Trafficinfo
- Train
- TV Guide

Block parameters

For these blocks you can set the following block parameters:

- icon: Icon name
- image: Image name. Image path is relative to the <dashticz>/img folder.
- title
- key: Unique identifier

These parameters all work in the same way. The default styling however might be different, depending on the block type.

The following sections provide an overview of additional changes per block type.

Button

The current button parameter `image` has been renamed to `btnimage`.

A button block can have a `image` parameter, which defines the image to be used as icon, and a `btnimage` parameter. This parameter defines the image of the button itself.

Public transport

The icon parameter of the Public transport block now must be set to the full icon name, as defined by FontAwesome.

Previous format

```
icon: 'train', //change: full icon name
```

New format:

```
icon: 'fas fa-train',
```

CSS class definitions

The CSS class definitions for each element of these blocks (block, icon, title, state) have been made consistent. This means you might have to update the styling in your `custom.css`, or the classes you use in `custom.js`. See *Special blocks*

6.2 Release Notes

For Dashticz's **beta** version Release Notes go to: <https://dashticz.readthedocs.io/en/beta/releasenotes/index.html>

For Dashticz's **master** version Release Notes go to: <https://dashticz.readthedocs.io/en/master/releasenotes/index.html>

6.2.1 Recent changes

Fixes

- Swiper vertical scroll bar

Code

- [Prelim] Calendar: New ical module to parse calendar data. Should solve most calendar issues, especially related to recurring events. Select via `method:2`

6.2.2 V3.9.3 Beta (9-3-2022)

Enhancements

- Dial: Needle step size configurable via `steps` block parameter.
- Dial: For wind device, add block parameter `subtype: 'windspeed'` to use wind speed for needle position instead of wind direction.
- Dial: For wind device, add block parameter `subtype: 'windgust'` to use wind gust for needle position instead of wind direction.
- Dial: Up/down dials for Thermostats, Blinds and Dimmers. See [Up-down dials](#)
- Public Transport: New block parameter `direction` to filter on line direction number. See [Public Transport](#)

Fixes

- Public transport: Translations
- Dial: Fix min, max setpoint setting in CONFIG.js
- Garbage: Fix for Purmerend, Suez, Blink

6.2.3 V3.9.2 Beta (27-2-2022)

Enhancements

- Garbage: Added Maashorst (Uden, Volkel, Odiliapeel, Reek, Schaijk en Zeeland)
- Public transport: New block parameter `show_direction` to show bus line direction.
- Public transport: New block parameter `lang` to set language for search results (for `irailbe` only).

Fixes

- Garbage: Uden (new URL, same as Maashorst)
- Garbage: Rova (for some zipcodes)

Code

- Switched to worker-timers, to improve background refresh
- Prevent caching `index.html`
- Update caching behavior

6.2.4 V3.9.1 Beta (13-2-2022)

Code

- Update development dependencies
- Update FontAwesome, Popper, IRO and Swiper to latest versions

6.2.5 V3.9.0 Beta (10-2-2022)

Beta version derived from v3.9 Master

6.2.6 v3.9 Master (10-2-2022)

Enhancements

- Trafficinfo: Add block parameters `showempty` and `showemptyroads` to control what to show in case of no announcements. See *Traffic info*

Fixes

- Trafficinfo: Bug fixes (wrong road name if no announcements)
- P1 Smart Meter: Display `NettUsage` as default value (=Usage-Delivery)

6.2.7 v3.8.11 Beta (28-1-2022)

Note: Some changes in dial styling, especially dial font sizes.

Enhancements

- New block type 'Door Lock Inverted'
- Dial: Selector menu can show title. See *Selector switch*

6.2.8 v3.8.10 Beta (23-1-2022)

Note: Public Transport changed. See *Public Transport*.

Note: Dial ring styling changed. See *v3.8.9: Dial styling*.

Enhancements

- Special blocks: Add class `empty` in case the special block is empty. Applicable to `alarmmeldingen`, `calendar`, `traffic`, `trafficinfo` and `train`.
- Graph: Improvement in customized axes styling. See *Styling of X- and Y- axes*
- Publictransport: Added ‘ovapi’ and ‘treinen’ as providers. Removed 9292, mobiliteit and VVS (non working APIs). Changed rendering. For all changes see *Public Transport*.

Fixes

- Changed dial styling for ring and blinds text. See *Custom Styling*.
- Dial: P1 decimals configurable via decimals block parameter.
- Calendar: Fixed issues with some recurring events in ical modules (PHP5 as well as PHP7 version)

6.2.9 v3.8.9 Beta (23-12-2021)

Note: Your images in buttons now might scale to the full block width. This is a side effect of the fix of the moon scaling. Reduce the block width in case your image is too wide.

Enhancements

- Dial: Support for blinds. See *Blinds*
- Frame: Add block parameters `scaletofit` and `aspectratio` to automatically scale the frame content to the block width. See *Frames*

Fixes

- Moon image scaling

6.2.10 v3.8.8 Beta (17-12-2021)

Fixes

- Garbage: Recycleapp (BE)
- Dials: Fix for so called spltdial with 0 not at top. For instance: `min=-10` and `max=50`

6.2.11 v3.8.7 Beta (5-12-2021)

Note: Weather icons changed. See *Icons*

Note: CSS styling for calendar events changed. See *Event styling*

Enhancements

- Calendar: eventClasses block parameter to customize styling based on event description. See *Event styling*
- Weather: New block parameter `icons` to set weather icons to 'line', 'linestatic', 'fill', 'static' or 'meteo'. See *Icons*
- New upgrade scripts in Makefile (Documentation to be updated)

Fixes

- Garbage: Recycleapp (BE), Avalex, Suez

6.2.12 v3.8.6 Beta (22-10-2021)

Enhancements

- Graphs: Now you can also display switch information in your graphs

Fixes

- Graphs: Fixes in y-axes labeling

6.2.13 v3.8.5 Beta (15-10-2021)

Fixes

- Make door lock switchable.
- Garbage: Venlo (new website)
- Custom function `getStatus` will be called twice. Second time after block creation (fixed)
- Weather block: fixed rain rate in hourly forecast
- Graph: Fix for displaying energy values, for instance for P1 devices

6.2.14 v3.8.4 Beta (13-8-2021)

Fixes

- Calendar fixes (recurring events, multiple events on same moment)
- ANWB traffic info: Change API v1 to v2
- Garbage: Fix for Rova

6.2.15 v3.8.3 Beta (29-5-2021)

Enhancements

- Weather: Added layout 4 option. See *Weather forecast*
- Weather: Colored icons (animated weather icons only). See *Weather forecast*
- Weather: show/hide wind dial and wind info, Wind as Beaufort, show/hide first forecast card

Fixes

- Weather: Changed styling of current weather block (center the three parts)
- Weather: Fix styling of forecast block for white Dashticz template
- Merged changes from master v3.8.0.1 and v3.8.0.2

Code

- Bump Swiper.js from 5.4.5 to 6.4.2

6.2.16 v3.8.2 Beta (24-4-2021)

Note: Breaking changes: New weather block.

Enhancements

- Rewrite of the weather block. See *Weather forecast*.

6.2.17 v3.8.1 Beta (14-4-2021)

Enhancements

- Change in auto swipe behavior. See *Auto swipe, auto slide*.

6.2.18 v3.8.0 Beta (10-4-2021)

Enhancements

- Auto slide timer configurable per screen via screen parameter `auto_slide_page`
- Fix for columns without block parameter
- Fix for icon size for special blocks on screen width < 975 pixels

6.2.19 v3.8.0.2 Master (14-5-2021)

Fixes

- Fix potential error in startup behavior

6.2.20 v3.8.0.1 Master (26-4-2021)

Fixes

- Standby: Prevent click to activate a Dashticz block while in standby

6.2.21 v3.8 Master (9-4-2021)

Master version derived from v3.7.7 Beta.

If your current Dashticz version is lower than v3.7.2 then before upgrading make a copy of custom/custom.css and custom/custom.js first!

See the upgrade instructions at v3.7.2 below.

6.2.22 v3.7.7 Beta (8-4-2021)

Fixes

- Garbage: Repaired Area, EDG, Groningen, Meerlanden

Enhancements

- P1 Smart Meter: Computed fields 'NettUsage', 'NettCounterToday' and 'NettCounter' which can be used as value in dials.
- Garbage: Set block parameter 'ignoressl' to true to disable https SSL checks.

Code

- Update of the external npm modules

6.2.23 v3.7.6 Beta (12-3-2021)

Enhancements

Note: Breaking changes. See *v3.7.6: Dial values* for update instructions

- Several dial enhancements. See *Dial values*
- Device hook: Function in custom.js which is called on every device update. See *Device hook*

Fixes

- Blinds: Support textOn and textOff block parameters

6.2.24 v3.7.5 Beta (28-2-2021)

Enhancements

- OWM widgets. See *OWM widgets*

Fixes

- Dials: Fix dimmer decimals
- Dials: Improved formatting
- Dials: Improved error handling
- Dials: Support setpoint for default dial

6.2.25 v3.7.4 Beta (20-2-2021)

Fixes

- Fix for Spotify block (removed the additional dummy block)
- Spotify: Improved playlist popup layout
- Improved error handling in PHP modules for calendar and garbage
- Dials: Resize disabled (to prevent size changes after first rendering)
- Garbage: block with company: 'ical' will now be detected correctly as Garbage block instead of Calendar
- Garbage: recycleapp.be
- Colorpicker: Add support for Hue RGBWW device by adding mode:1 block parameter

Enhancements

- Dial: block parameter `iconSwitch` to set the fontawesome icon to use for an on/off switch
- Dials: Support added for text devices and for dials without device.
- Dials: Text devices will be recognized correctly in default dial as well, meaning you can combine several text devices into one dial.
- Dials: Set number of decimals with `decimals` parameter
- Garbage will be sorted in the same order as `garbage` block parameter (or `config['garbage']`)

6.2.26 v3.7.3 beta (24-1-2021)

Note: Make a backup of CONFIG.js, custom.css and custom.js

Code

- Redesign internal block framework
- Removed old calendar block ‘icalendar’ and calendarurl config setting

Enhancements

- Calendar: (New calendar block, layout 0 and 1 only) The class ‘agenda-empty’ is applied to the calendar block in case there are no appointments.
- Battery Level indicator for Domoticz devices. Battery icon will be displayed when the battery level is below `batteryThreshold`. See *Battery level*.
- TV Guide: Block parameter `layout` has been added, to display the TV guide with/without channel name. See *TV Guide*
- Graph: Block parameter `labels` has been added, to rename the device names that are used in `groupByDevice` graphs.

Fixes

- Bugfix security panel lock screen default setting
- Show last update time when `last_update` is set as block parameter
- Graph: Fix for block parameter aggregate as array
- Calendar: Update icalparser for PHP8 compatibility

6.2.27 3.7.2 Beta (27-12-2020)

Note: Update instructions.

I’ve removed `custom/custom.css` and `custom/custom.js` from the Dashticz repository, because these are user configuration files, and should not be part of the Dashticz repository.

However, that means this update cannot be installed with `git pull` directly, because then git will report an error if you have modified one or both files.

To solve this, first make a backup of these two files:

```
mv custom/custom.js custom/custom.js.bak
mv custom/custom.css custom/custom.css.bak
```

In case you use the `custom_2` folder, repeat these steps for that folder:

```
mv custom_2/custom.js custom_2/custom.js.bak
mv custom_2/custom.css custom_2/custom.css.bak
```

Then update to the latest version as usual:

```
git pull
```

And restore your backups:

```
mv custom/custom.js.bak custom/custom.js
mv custom/custom.css.bak custom/custom.css
```

And for the custom_2 folder:

```
mv custom_2/custom.js.bak custom_2/custom.js
mv custom_2/custom.css.bak custom_2/custom.css
```

You only have to do this once: Next updates can be installed with a normal ‘git pull’

Enhancements

- Calendar: New block parameter `emptytext` to define the text to show where there are no calendar appointments. Only works for the new calendar block. See [newcalendar](#)
- Custom graph: aggregate parameter can be an array to specify different aggregation methods per data element. See [GroupBy](#)
- Graph: New parameters `axisRight` to show the first Y axis on the right (default is `false`), and `axisAlternating` to show Y axes alternating left/right (default: `true`).
- Support for device (sub)type Managed Counter
- Flipclock: New block parameters `showSeconds` (true or false) and `clockFace` (12 or 24)
- Security panel: New block parameters (`decorate`, `headerText`, `footerText`, `scale`). See [Domoticz Security Panel](#)

Fixes

- Graph: Fix for data acquisition day graph gas device.
- Colorpicker: Some fixes in warm white/cold white color setting.
- Improved styling of modal popup windows.

6.2.28 3.7.1 Beta (19-12-2020)

Enhancements

- Graph: Enable graphs for Lux device type
- Popup window: Add `newwindow: 5` to open an url as image instead of iframe (doc to be updated)
- Clock: New Hayman clock. Add block ‘haymanclock’ to a column, or use `type: 'haymanclock'` in your block definition.
- Clock: New basicclock, which is the same as the normal clock, but then responsive. (scales with the width)
- Clock parameters: haymanclock, flipclock, stationclock and basicclock all support the block parameters `size` to set the width of the clock and the parameter `scale` to scale down the width with a relative factor (`scale: 0.6`)

6.2.29 3.7.0 Beta (13-12-2020)

Code

- NPM update, code formatting

6.2.30 3.7 Master (13-12-2020)

Master version derived from 3.6.9 Beta

6.2.31 3.6.9 Beta (10-12-2020)

Enhancements

- Garbage: New garbage block parameter `maxdays` to set the number of days to show the garbage collection info (2 means today and tomorrow)
- Stationclock: New block parameter `size` to set the size of the clock. See *Station Clock*
- Stationclock: New configuration parameters. See *Station Clock*

Fixes

- Garbage: Fix DeAfvalApp (https instead of http)
- Garbage: Add avri as garbage company
- Garbage: add layout as block parameter. Use `layout : 0` to format the garbage rows as one string and `layout : 1` to use table layout.
- Garbage: Fix Afvalwijzer 2021 data
- Prevent `:hover` effect for touch devices

6.2.32 3.6.8 Beta (27-11-2020)

Enhancements

- Garbage: New providers Suez (Arnhem), Blink (Asten, Deurne, Gemert-Bakel, Heeze-Leende, Helmond, Laarbeek, Nuenen, Someren), Purmerend
- Garbage: New provider afvalstoffendienst
- Garbage: New provider GAD
- Colorpicker: Add support for WW dimmers (Philips Hue)
- Chart: For custom graphs you can define the icon to use for each graph button. See *Custom graphs*
- Timegraph: New special block to define a moving time graph. See *Timegraph*
- Garbage: Additional styling. See *Styling*
- Garbage: New block parameter `date_separator` to configure the text between garbage type and date
- Garbage: Format as table. See *Styling*

Fixes

- Calendar: Add 'method:0' to your calendar block definition in case you experience issues with recurring events. Only works for the new calendar block. See `newcalendar`
- Fix for X10 security motion device.

6.2.33 3.6.7 Beta (4-11-2020)

Update of the Garbage module. See *Upgrade from Dashticz 3.6.6 and earlier* for upgrade information.

6.2.34 3.6.6 Beta (30-10-2020)

Enhancements

- Dashticz URL parameters. See *Dashticz URL parameters*
- Dials: Set the block parameter `animation` to `true` or `false` to enable/disable dial animations.
- Add `timeout` CSS class to Domoticz devices in the timeout state. See *Styling*

Fixes

- Garbage: Fix for Mijnafwijzer on iOS
- Disable Dashticz refresh if `config['dashticz_refresh']` is 0
- Bugfix initialization code

6.2.35 3.6.5 Beta (22-10-2020)

Fixes

- Button: `newwindow: 3` handling is fixed.
- Scenes: Switch always on
- Switched to an alternative server to provide the covid-19 data

Enhancements

- Button, special blocks: Initiate the `url` parameter as POST request by setting `newwindow: 4`
- Add support for Domoticz x10 security sensor
- Dial: Combine data from several devices. See `genericdial`

6.2.36 3.6.4 Beta (6-10-2020)

Fixes

- PV Output Temp device.

Update notes

- The icon for PV Output blocks are not automatically set to ‘fas fa-sun’ anymore. You still can do this manually in a block definition. In a future version I’ll improve the default settings for Domoticz device types.

6.2.37 3.6.3 Beta

Enhancements

- Set config setting `security_panel_lock` to 2 to activate security panel lock in ‘Armed Home’ mode as well.
- Dial type now enabled for most devices. See `genericdial`

Fixes

- Remove scroll bar of the modal security panel (security panel lock)
- New config setting `use_cors` to enable CORS proxy for OWM. Set to `true` on Android 4.4.2.
- Garbage: `recycleapp`

6.2.38 3.6.2 Beta

Fixes

- Fix for graph issues in 3.6

6.2.39 3.6.1 Beta

Enhancements

- Custom HTML block. See *HTML custom block*

6.2.40 3.6.0 Beta

Beta version, same as 3.6 master.

Code

- Update of the external js modules

6.2.41 3.6 Master

Enhancements

- New Dashticz config parameter ‘`swiper_touch_move`’ to disable/enable swiping the screen on touch
- Graph: The ‘today’ button now shows the full day data. The range ‘day’ still exists as well, which still can be used in custom graphs.

- Add support for device with subtype 'Current'
- Popup graphs enabled by default for most block types. To disable a popup graph, add `graph: false` to the block definition.

Code

- Update FontAwesome to 5.14.0

Fixes

- Camera block
- Garbage: Ophaalkalender (BE) doesn't work anymore. It has been replaced by recycleapp.
- Security panel home symbol.
- Garbage: Meerlanden switched to ximmio as garbage data provider
- Garbage: Fixed method to retrieve data from mijnafvalwijzer
- Fixed use_favorites config setting. Changed default to false, meaning all devices will be available for Dashticz.
- Remove CORS for OWM data

6.2.42 3.5.2 Beta

Enhancements

- New colorpicker for RGB devices, including support for whites. The `no_rgb` setting is obsolete. See [RGB Color picker](#)

Fixes

- Fix for Omrin garbage provider
- Fix for Venlo garbage provider

Code

- Update to jquery 3.5.1

6.2.43 3.5.1 Beta

Enhancements

- Domoticz textblocks, traffic, trafficinfo, longfonds and public transport now support the block parameters `url`, `newwindow`, `forcerefresh` and `password` giving it the same behavior as a button if you want to open an url on click.

Fixes

- Change traffic info provider for traffic block

6.2.44 3.5.0 Beta

Same as 3.5 Master

6.2.45 3.5 Master

New master release derived from 3.4.10 beta.

See the release notes for the beta releases below for all changes.

6.2.46 3.4.10 (Beta) (7-6-2020)

Enhancements

- Japanese language support (preliminary)
- Improved Camera block . See *Cameras*

Fixes

- Stop called twice for Blinds stop button
- Improve Dial representation on Android devices
- Improved graph groupBy function

6.2.47 3.4.9.1 (Beta) (26-5-2020)

Fixes

- Several bug fixes

6.2.48 3.4.9 (Beta) (25-5-2020)

Fixes

- Improved number formatting for graph header and tooltip. See *Number format*
- Block definition with custom keys: consistency in block selection for subdevices, CSS class application and function names in custom.js. This may result in a breaking change. See *v3.4.9: Custom keys in block definition*

6.2.49 3.4.8 (Beta) (20-5-2020)

Enhancements

- Improved trafficinfo layout

Fixes

- IE11 support
- iOS9 support

Code

- Standardized formatting of source code

- Removed eslint warnings (first batch)

6.2.50 3.4.7 (Beta) (18-5-2020)

Enhancements

- Support for Dials. See *Dial*

Fixes

- Refresh of graph while in standby

6.2.51 3.4.6 (Beta) (13-5-2020)

Enhancements

- Enable graphs for Voltage and Distance devices
- Parameter `timeformat` to configure time format for ‘alarmmeldingen’. See *Alarmmeldingen*
- TV guide (Dutch: *tvgids*) made clickable
- More options to customize the graph header. See *customHeader*

Fixes

- Fix for ANWB Traffic Info (new API)
- Fix for recurring calendar events (older than 3 year, without end date)

6.2.52 3.4.5 (Beta) (23-4-2020)

Fixes

- Garbage: Cure moved to ‘mijnafvalwijzer’
- Synchronization Domoticz security panel state
- Bug fix popup chart refresh

6.2.53 3.4.4 (Beta) (18-4-2020)

Enhancements

- Add ‘Current’ Domoticz device type.
- Improved security panel. See <todo>

Fixes

- Fix for refresh of Scenes/Groups and some temperature sensors

6.2.54 3.4.3 (Beta) (9-4-2020)

Enhancements

- New calendar layout. See *newcalendar*

Fixes

- Group/scene status refresh
- Unit parameter, which can be used for formatting the value of some Domoticz devices. See *Formatting*

6.2.55 3.4.2 (Beta) (3-4-2020)

Enhancements

- Add dewpoint block for TempHumBar devices
- Corona block type
- Custom header for graph blocks. See *customHeader*
- Camera block. See *Cameras*

Fixes

- Calendar recurring events (experimental)

Internal

- Refactoring blocktypes

6.2.56 3.4.1 (Beta)

Note: Breaking changes. See *v3.4.1: Redesign block definition* for update instructions

Redesign

- Domoticz blocks: inline blocks. Use `idx` as parameter in your block definition to indicate the block is a domoticz device. See *v3.4.1: Redesign block definition*

Enhancements

- Support for showing a graph more than once on the dashboard.
- Support for RGBWZ devices
- Omrin garbage company
- Calendar: Optionally display start time only by setting `startonly` block parameter
- New block parameter `password` to password protect switches, buttons, thermostats, sliders.
- Filter parameter for the news block. Define as block parameter. Example:

```
blocks['my_news'] {
  feed: 'http://www.nu.nl/rss/Algemeen',
  filter : '5 items', // to only show the 5 latest news items, or:
  filter: '2 days',  // to only show news items of the last 2 days, or:
  filter: '1 month', // to only show news items from last month
}
```

- New special block: `alarmmeldingen` (Dutch). See *Alarmmeldingen*
- Update other blocks from `custom.js` functions by calling `Dashticz.setBlock`. See *Change value of another block*

Fixes

- Requests to Domoticz will not be send via a websocket connection (not reliable)
- Fix for Evo devices
- Improved the height adjustment of a news block with inline images
- Fix for updating devices via `getStatus_idx` in `custom.js`
- Fix for initial update of block defined by `getBlock_<idx>()` in `custom.js`

6.2.57 3.4.0 Beta (8-2-2020)

Enhancements

- Websocket interface for Domoticz version > 4.11000 to receive instant device updates. See [Websocket connection](#)
- The News block will show the inline images. By setting the news block parameter 'showimages: false' the inline images will be hidden. See [Config Settings](#)
- graph and multigraph have been combined into the same graph block. See [Graphs](#).

In case you update from 3.3.5 beta: The parameter `multigraphTypes` has been replaced by `graphTypes`

Optimizations

- Dashticz will only receive the updates for devices that changed since the previous update. This will increase responsiveness. In the previous version Dashticz received all device info at every update (default 5 second cycle).

6.2.58 3.3.5 Beta (28-1-2020)

Fixes

- Garbage Uden
- Restored PHP5 compatible ical library next to the PHP7 library. The PHP5 library is selected automatically on systems with PHP version lower than 7.1. The PHP5 library doesn't show yearly recurring events correctly.

6.2.59 3.3.4 Beta (22-1-2020)

Enhancements:

- Multigraph functionality. See [Graphs](#).

6.2.60 3.3.3.1 Master (4-2-2020)

Fixes

- Garbage Uden
- Reenabled PHP5 calendar module

6.2.61 3.3.3 Master (22-1-2020)

Fixes

- New PHP ical library to solve issue with recurring events. Note: PHP 7.1 or higher is required.

6.2.62 3.3.2 Master (18-1-2020)

Master version derived from 3.3.1 beta.

If you are upgrading from a previous master version please read *v3.2.0: Change in HTML template for special blocks*.

Additional fixes

- Fix standby screen in case of single screen.

6.2.63 3.3.1 Beta (13-1-2020)

Enhancements

- Complete dimmer block is clickable (not just the icon)

Fixes

- Multiple stationclocks
- Background fill complete screen in case of single screen
- Add dimmer for RGBWWZ devices
- TwenteMilieu garbage collection
- Bar-afvalbeheer garbage collection (for Barendrecht, Rhoon). Use ‘barafvalbeheer’ as garbage_company.

6.2.64 3.3.0 Beta (5-1-2020)

Enhancements

- Evohome support. See *Evohome*

Fixes

- Improved error handling
- Improved handling of chart data
- Almere garbage
- Login screen background image

6.2.65 3.2.1 (10-12-2019)

Enhancements

- Addition of special block ‘secpanel’ which adds a Domoticz like security panel. See *Domoticz Security Panel*

Fixes

- Swiper transition effect

- Update to latest jQuery version to solve security alert

6.2.66 3.2.0

Warning: Breaking changes

Main change:

- Standardization of the html template for special blocks. See *v3.2.0: Change in HTML template for special blocks*

Other changes:

- Enable swiper for mobile devices
- Update to swiper 5.2.0. Added the config parameters `vertical_scroll` and `enable_swiper` to control swiping and scrolling behavior. See *Config parameters*
- Bundle most external dependencies (webpack, babel, package.json)

Fixes

- Calendar: Improved handling for recurring events
- Blinds: Fix for custom icons

6.2.67 3.1.2 (26-10-2019)

Enhancements

- Improved calendar layout for full day events. Added timezone adjust parameters.

Fixes

- Fix for loading Dashticz without external network
- Load Sonarr images via CORS proxy

6.2.68 3.1.1 (15-10-2019)

Enhancements

- Show calendar with table formatting by setting blockparameter `calFormat:1`. See *calTable*
- Session Time Out option

Fixes

- Graph for barometer device
- Almere garbage provider
- Wind speed unit interpretation in case of non default Domoticz setting
- Protect parameter for dimmers.
- Removal of ES6 dependency (introduced by the graph update)

Upgrading from earlier versions:

buttons: Use the `btnimage` parameter instead of the `image` parameter. The parameter `isimage` is not used anymore.

6.2.69 3.1.0 (18-9-2019)

Enhancements

- New config setting '`start_page`' to set Dashticz start page number
- New parameter '`scrollbars`' to set scrollbars in frame. See *Frames*
- New graph module. It's not completely backwards compatible. Especially styling will be different. See *Graphs*

Fixes

- Faster initial display of the Dashticz dashboard.

6.2.70 3.0.6 (28-8-2019)

Enhancements

- OpenWeatherMap module: support for using the city id as city name
- Icon/image options for blocktitles

Fixes

- Docker PHP timezone
- News update in standby
- Robustness install script and makefile
- Auto restart docker container after reboot
- Documentation updates (Thanks to HansieNL)

6.2.71 3.0.5 (4-8-2019)

- Update of documentation.
- Improvements in the automatic installation script.

6.2.72 3.0.4 (1-8-2019)

Main changes:

- New Domoticz Github location: <https://github.com/Dashticz/dashticz>
- New graph options to set the graph appearance. See *Graphs*.

Fixes:

- OWM Weather layout

6.2.73 3.0.3 (20-7-2019)

Main changes:

- Fixed the broken Spotify module
- Improved layout (icon size for certain screen widths)

6.2.74 3.0.2 (19-7-2019)

Main changes:

- New block parameters (textOn, textOff, imageOn, imageOff, iconOn, iconOff) to control the display of block text, icons and images depending on the device state.

6.2.75 3.0.1 (25-6-2019)

Main changes (thanks to Steven):

- New special block: Traffic information based on providers, ANWB is the first one. See *Traffic info*.
- Additional filter options for the public transport module. See *Public Transport*.

Fixes:

- Update of the installation script. See *Automatic install*

6.2.76 3.0.0 (13-6-2019)

This is the first Dashticz v3 release.

Main change: New Domoticz Github location: https://github.com/dashticzv3/dashticz_v3

New functionality:

- Change in `forcerefresh` parameter of a button to support cheap Chinese webcams.
- Support for TempBaro device
- Sizing the y-axis of the graph to relevant data
- Adding possibility to draw graph data for Qubino ZMNHTDx Smart meter
- Add bar graph type option.
- Streamplayer: Add class when in playing state to enable styling via custom.css
- Radio streaming image (radio-streaming.png)

Fixes:

- Make index2.html consistent with index.html
- Streamplayer error handling

6.2.77 2.5.9 (11-3-2019)

New functionality:

- Caching prevention mechanism also applied to button popup frame (`forcerefresh` parameter)
- Added Air Quality as graph type (and CO2 as graph property)
- Support of RGB dimmers (RGBW and RGBWW dimmers were supported already)
- Added confirmation option for switches (See `confirmation` parameter in Domoticz blocks)

Small fix:

- TwenteMilieu garbage pickup dates

6.2.78 2.5.8 (8-3-2019)

Small fixes:

- Prevent caching of the version info.

6.2.79 2.5.6 and 2.5.7

- Graph improvements. See *Graphs* for usage description.
 - Selection of values you want to show in a graph via the `graphTypes` parameter. See *Block parameters*.
 - Support for the `title` and `width` parameter in a graph block.
- Additional mechanism to prevent caching of images in a button via the `forcerefresh` parameter. See *forcerefresh*.
- Change background color for active ‘slide’ button. See *Slide button*.
- Flash on change. See *Flash on change*.

If you have defined the flash parameter for a device-block, then the block will flash on change. The formatting of the flash can be modified via the class `.blockchange` in your `custom.css`.

The parameter `config['blink_color']` is (temporarily?) not used anymore. (reason: the apply background mechanism didn’t work for non-touch devices)

- Improved layout of blinds
- Update of Romanian language
- Update to FontAwesome 5.7.2
- Fix for some RFX meters (incl. water meter)

7.1 Connection

This section addresses several connection issues

7.1.1 Network time-out errors

Description

Dashticz in most cases won't load, or loads partially. DevTools network tab shows 408 errors.

Applicability

Dashticz server runs in a Docker container. You are using a Pi Raspbian 10 (Buster)

Check this with:

```
cat /etc/os-release
```

Solution

There is an incompatibility issue between one of the libraries. You can fix this by executing the following commands:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 04EE7237B7D453EC_
↳ 648ACFD622F3D138
echo "deb http://deb.debian.org/debian buster-backports main" | sudo tee -a /etc/apt/
↳ sources.list.d/buster-backports.list
sudo apt update
sudo apt install -t buster-backports libseccomp2
```

After this stop and start the Docker container by executing the following commands in the Dashticz folder on your PI:

```
make stop
make start
```

7.2 Docker

In case you tried to install Dashticz several times via the automatic install script, you might end in having multiple Dashticz instances in parallel. The sections below can help in cleaning up.

To see all docker containers:

```
sudo docker ps
```

This will give output similar as:

CONTAINER ID	IMAGE	COMMAND	CREATED	
→STATUS	PORTS	NAMES		
500c42ae47eb	dtv3-8083	"docker-php-entrypoi..."	3 weeks ago	↳
→ Up 2 weeks	0.0.0.0:8083->80/tcp	dtv3-8083		
af0cd4278a60	dtv3-8082	"docker-php-entrypoi..."	3 weeks ago	↳
→ Up 2 weeks	0.0.0.0:8082->80/tcp	dtv3-8082		

In this example you see there are two Dashticz containers running, on port 8082 and port 8083.

To stop the container with name dtv3-8083, which is running on port 8083:

```
sudo docker stop -t 5 dtv3-8083
```

The -t 5 parameter will give the container 5 seconds to gracefully shut down.

After this you have an unused, stopped image dtv3-8083.

To get an overview of all images:

```
sudo docker images
```

This will show something like:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	a5d13a69533b	3 weeks ago	414MB
dtv3-8082	latest	5c0cc9bf78e7	3 weeks ago	414MB
dtv3-8083	latest	5c0cc9bf78e7	3 weeks ago	414MB
dtv3-8084	latest	5c0cc9bf78e7	3 weeks ago	414MB
php	apache	5e1d7ed3b92a	5 weeks ago	414MB
hello-world	latest	fce289e99eb9	16 months ago	1.84kB

To remove all unused containers and images:

```
sudo docker system prune
```

You will see a prompt, to which you can answer Y:

```
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache
```

(continues on next page)

(continued from previous page)

```
Are you sure you want to continue? [y/N] Y
Deleted Containers:
8140945ab5e770297a9b89751d3d8c303ca1ad2702dbaf1b6b68e6b1c6266aca
394f7f28dc9a26336a17534f539061a49caf624b090f0174620412472146e932
c4f8f460f86ee27a82090c15c070a66bf8a2763f430c8d534301c54ba585e62c
4401d69b78159b14e0f618758d2a547da34e2d9709ab661be51cedfc66774537

Deleted Images:
deleted: sha256:a5d13a69533bc8695fbd6feb0d094462d2b5a6d3d89256cd77c6f4b71c82271d
deleted: sha256:4db803ea9cd4b5e91e9ea3fa0746707aa3bbe6138f02083aa49aec2e6e59b6bb
deleted: sha256:36335a64948142f427dd6b94808682d7e49e4e252fdcd73dfa4f347f78513412
```

You can check the results with `sudo docker ps` and `sudo docker images`

If you want to change the port number of a running Dashticz container:

Switch to the Dashticz root folder:

```
cd <dashticz>
```

If you followed the automatic install script, there will be Makefile.ini file, with the following content:

```
APP=dtv3-8084
PORT=8084
```

To stop the current container:

```
make stop
```

Edit Makefile.ini, and change the PORT number, and APP name (the last one is not really needed, but makes life easier):

```
nano Makefile.ini
```

Then restart the container:

```
make start
```

And voila, you now have a Dashticz container running on a new port.

More useful Docker commands on:

[https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/
#remove-one-or-more-containers](https://linuxize.com/post/how-to-remove-docker-images-containers-volumes-and-networks/#remove-one-or-more-containers)

7.3 Domoticz blocks

This sections addresses several issues related to displaying Domoticz blocks

7.3.1 Domoticz block not visible in Dashticz

There are a few possible causes

use_favorites

In `CONFIG.js` you have `CONFIG['use_favorites']=1`, while you did not mark your device as favorite in Domoticz

Solution:

Set `CONFIG['use_favorites']=0` in `CONFIG.js` or mark your device as favorite in Domoticz

device id incorrect

Check that your device id actually exists in Domoticz

Room plan

If you make use of a room plan in Dashticz, then check that you've added the device to the room plan in Domoticz.

User rights

If you've configured a specific Domoticz user for Dashticz, then check that this Domoticz user has access to the device.

Support

Describe your issue in the Dashticz forum <https://www.domoticz.com/forum/viewforum.php?f=67>

Include the Domoticz json device description, which you can obtain via the following link:

```
http://domoticz ip:port/json.htm?type=devices&rid=123
```

Fill in your domoticz ip and port. Replace 123 with your device id.

8.1 Code

All on Github.

8.1.1 Development environment

I prefer to use Visuals Studio Code.

8.1.2 Source code

In general the ‘rules’ are:

- ES5 compliant
- Styling if possible via creative.css
- Server code PHP 5 compliant
- use ‘prettier’ for code formatting
- eslint should give no warnings

Some users still have a ES5 browser, like old Android tablets. In the past I used the ‘const’ keyword a few times. This resulted in complaints.

For ES5 vs ES6 see for instance: <http://es6-features.org>

We could switch to ES6, but then probably we have to start using Babel as well, to transpile back to ES5. This is something I would like to prevent, because it makes the develop/build environment a bit more demanding.

Guidelines:

- No ES6 syntax, like ‘const’, ‘let’, arrow functions, class keyword, object spread, Promise.
- No synchronous AJAX calls.

- jQuery promises may be used.

8.1.3 Dependencies

Most external libraries are managed via npm and bundled with Webpack. If you need to update the bundle, follow the following steps.

1. Install node
2. Install the dependencies

```
npm install
```

3. Build the bundle

```
npm run build
```

The sources for the bundle can be found in `src/`. These files will be transpiled with Babel to ES5.

The Swiper library is in ES6 format, and will be transpiled to ES5 as well. See `babel.config.js` for the configuration.

The regular Dashticz files will NOT be transpiled: They should be in ES5 format.

jQuery 3.4.1

In December 2019 Dashticz was upgraded from jQuery 2.2.4 to jQuery 3.4.1, because of a reported security vulnerability in jQuery 2.2.4. Migration is checked via `jquery-migrate` plugin, which can be enabled in `src/index.js`.

The `spectrum-colorpicker` is not fully compatible with jQuery 3. The `jquery-migrate` plugin generates warnings in the console if the `jquery-migrate` plugin is enabled.

8.1.4 Design

Since v3.2.0 a standard component will be used for most special blocks. All blocks are loaded by `js/dashticz.js`. The source code for each block can be found in the `js/components` folder.

If you want to add a new block with the name 'myblock', then follow the following steps:

Create the file `js/components/myblock.js`. A minimal implementation should contain the following:

```
var DT_myblock = {
  name: "myblock",
  run: function (me) {
    //this function will be called after the component has been initialized and
    ↪has been mounted into the DOM.
    //me.mountPoint: Mountpoint of the container (dt_block)
    //For basic usage you will add additional code to $(me.mountPoint + ' .dt_
    ↪state')
    //me.block: Reference to the block definition in CONFIG.js
  }
}

Dashticz.register(DT_myblock); //Don't forget to register the block
```

Add 'myblock' to the components variable at the start of `js/dashticz.js`

An example of a more extensive implementation:

```

var DT_myblock = {
  name: "myblock",
  canHandle: function (block, key) {
    //returns a boolean to indicate whether the block can be handled by this_
↪component
    //key is the identifier (string) that is used in the column definition to_
↪select a block.
    //In case an object is provided in the column definitions (like buttons,_
↪frames) then key is undefined
  },
  init: function () {
    //Will be called for initialization
    //returns a jquery deferred (similar to a Promise)
  },
  defaultCfg: { //All optional. defaultCfg can also be a function and then will_
↪receive block as parameter.
    icon: 'fas fa-newspaper', // string to define the default icon
    containerClass: function (block) { //function returning a string containing_
↪class names that will be added to the block
      return 'hover'
    }, //Or:
    containerClass: 'hover',
    containerExtra: function (block) { //function or string returning additional_
↪settings for the container HTML element (dt_block)
      return (block && block.maxheight) ? ' style="max-height:' + block.
↪maxheight + 'px;overflow:hidden;" : ''
    }
  },
  defaultContent: function (me) { //Optional. function (or string) returning the_
↪static content of dt_state
    return '<ul id="newsTicker"></div>'
  },
  run: function (me) {
    //this function will be called after the component has been initialized and_
↪has been mounted into the DOM.
    //me.mountPoint: Mountpoint of the container (dt_block)
    //For basic usage you will add additional code to $(me.mountPoint + ' .dt_
↪state')
    //me.block: Reference to the block definition in CONFIG.js
  },
  refresh: function (me) {
    // if me.block.refresh is defined, and this function exists, then this_
↪function will be called every <me.block.refresh> seconds.
  }
}

Dashticz.register(DT_myblock); //Don't forget to register the block

```

Add 'myblock' to the components variable at the start of js/dashticz.js

The following key words are reserved and should not be modified in me or block:

- key
- mountPoint
- type
- name

And specifically for me:

- block

8.1.5 Github workflow

We use a PR (Pull Request) based workflow, with preferably one new/changing feature per branch. All work is derived from the beta branch. If the beta branch is stable, a master branch will be derived from the beta branch.

For big changes a temporary branch will be created to test the new functionality by a bigger audience.

Basic workflow

1. Create an account on Github.com
2. Fork the Dashticz repository on github.com
3. Clone your own repository locally:

```
cd <working directory of choice>
git clone https://github.com/<username>/dashticz
cd dashticz
```

4. Add the dashticz upstream remote:

```
git remote add upstream https://github.com/Dashticz/dashticz
```

5. Get the latest changes:

```
git checkout beta
git fetch upstream
git merge upstream/beta
```

6. Create a new branch for your changes:

```
git checkout -b mynewfeature
```

7. Make the changes

8. Add the new files (if any):

```
git add .
```

9. Commit the changes:

```
git commit -am "My new feature"
```

10. Push the changes to your own Dashticz repository:

```
git push origin mynewfeature
```

11. On github.com create a Pull Request with the request to merge your own branch into beta

12. Have some patience, and lokonli will merge your PR

After your PR has been merged, you should cleanup your repository.

13. Delete your mynewfeature branch from your Dashticz repository on github.

14. Switch back to the beta branch:

```
git checkout beta
```

15. get the new beta:

```
git fetch upstream
git merge upstream/beta
```

16. Delete your local mynewfeature branch. It's not needed anymore, because it has been merged:

```
git branch -d mynewfeature
```

If you want to make additional changes, go back to step 6

Test branch

If additional testing is required then lokonli will not merge directly into beta (step 12), but will create a test branch. To continue working on this testbranch:

```
git fetch upstream
git checkout testbranch
git merge upstream/testbranch
git checkout -b mynewfeature
```

Then you have a new branch 'mynewfeature' derived from testbranch. Continue with step 7-10 to make your changes.

On github create a PR with the request to merge your new branch into testbranch.

Updating documentation

If possible update the documentation together with your code changes in the same PR. For updating the documentation see [Documentation](#)

8.2 Design

Starting to collect some design info here. Some structure will be added later.

8.2.1 File structure

main.js is the starting point. This file is responsible for:

- loading all necessary js files, the config files, the css files.
- Creation of screen and column layout, including the multi screen swiper, top bar, settings screen.

When this is ready, blocks.js gets the lead.

Blocks

blocks.js is responsible for creating all blocks on the Dashboard. Depending on the block type this responsibility is delegated to helpers.

Switches are handled by switches.js Thermostats and EvoHome devices are handled by thermostat.js

Several specific Domoticz devices are handled by block.js itself, like P1 smart meter and TempHumBar devices.

Most of the non-Domoticz block will be handled by special components, which can be found in the `js/components` folder. The `Dashticz.js` file functions as generic component factory. Depending on the block type, the specific components factories are called to create a component instance.

Services

The most important service modules are:

- `dashticz.js`: Responsible for creation of components, as described above
- `domoticz-api.js`: Responsible for handling all interaction with Domoticz
- `settings.js`: Responsible for configuring all Dashticz settings

8.2.2 Domoticz-api

This module handles most of the communication with Domoticz. Two types of communication channels are being used:

- websocket connection, supported from Domoticz 2020.1 onwards.
- http connection

At startup `domoticz-api` tries to initiate a websocket connection. The advantage of a websocket connection is:

- less overhead, since the websocket connection stays alive. So no overhead for recreating a http connection
- Domoticz actively pushes device changes to the websocket interface, meaning instant device updates in Dashticz

At startup `Domoticz-api` requests all devices from Domoticz, and caches the information for later use.

Subscribe

`domoticz-api` provides a subscribe interface:

```
Domoticz.subscribe(idx, getCurrent, callback)
```

`idx`: Domoticz device index you want to get a subscription on.

To subscribe to a variable add 'v' in front of the variable index, like 'v1'. To subscribe to a group or scene add 's' in front of the group index, like 's1'

There are some special purpose indices, which can be subscribed to as well:

idx	Description
<code>_secstatus</code>	Domoticz security status
<code>_secondelay</code>	Delay used by Domoticz before switching the alarm to 'Arm Away' or 'Arm Home'
<code>_Sunrise</code>	Sunrise time
<code>_Sunset</code>	Sunset time

If `getCurrent` is set to `true` then the callback function will be called with the current value directly.

The callback function receives the device info as parameter.

getAllDevices

You can request the actual status of devices via `Domoticz.getAllDevices`:

```
var alldevices = Domoticz.getAllDevices();
var device = alldevices[123]; //To get device 123
```

Domoticz request

Use the request interface to send a request to Domoticz:

```
request(query, forcehttp)
```

Parameter	Description
query	The Domoticz json query: everything after <code>/json.htm?</code>
forcehttp	Set to <code>true</code> (=default) to enforce http connection.

The default value of `forcehttp` is `true`, because Domoticz doesn't handle all websocket commands correctly. Examples:

```
* EvoHome devices (and thermostats?)
```

The function `request` returns a JQuery promise, containing the Domoticz reply as resolve parameter.

Message queue

Sometimes it might be needed to prevent handling of device updates by Dashticz. You can put the message queue of a specific id on hold via:

```
Domoticz.hold(idx)
```

Don't forget to release the message queue afterwards:

```
Domoticz.release(idx)
```

8.3 Translations

Dashticz supports a long list of languages. The language files are managed via <https://lokalise.co>.

We appreciate your contributions to the language files! Please leave a remark in the Dashticz forum:

<https://www.domoticz.com/forum/viewtopic.php?f=67&t=18697>

8.3.1 Guidelines for merging

Changes to 'en-uk.json' and 'nl-NL.json' in the beta branch are pulled automatically by lokalise.co

In lokalise.co:

- edit incorrect and missing translation keys.
- click 'download' -> build only.

This will create a new pull request in the Dashticz repository.

On github:

- Merge the pull request.
- Delete the lokalise branch.

Done.

8.3.2 Merging of changes in other language files

Save the changed language file locally.

In Lokalise click on 'upload' and upload the changed language file.

Then follow the same steps as above.

8.4 Documentation

All documentation is maintained in docs-folder of the Github source tree.

For generating the documentation Sphinx and readthedocs.org is being used.

For more information see:

- sphinx
- readthedocs
- rst: reStructuredText, the markup language we use

Basic

For basic modification of the documentation just edit the files in the docs folder, push your changes to github and create a pull request. After merging of the change the documentation will be pulled by readthedocs.org and the new version of the documentaion becomes available on <https://dashticz.readthedocs.io>

Advanced

1. install Sphinx:

```
sudo apt update
sudo apt install python-pip
sudo apt install python-sphinx
```

2. Make changes:

```
cd [dashticz_folder]/docs
nano mydocumentation.rst
```

3. Create a local html version:

```
cd [dashticz_folder]/docs
make html
```

4. Check whether you are happy with the result by opening the generated html file in the build folder in your browser.
5. Clean up the build results:

```
make clean
```

6. Push your changes to Github and create the PR.

8.4.1 Coding styles

Headers

Example:

```
Header level 1
=====

Header level 2
-----

Header level 3
~~~~~
```

These header levels will appear in the table of contents on the left.

Tables

Preferred method for defining a parameter table:

```
.. list-table::
   :header-rows: 1
   :widths: 5, 30
   :class: tight-table

   * - Parameter
     - Description
   * - width
     - ``1..12``: The width of the block relative to the column width
   * - title
     - ``'<string>'``: Custom title for the block
```

Alternative 1:

```
.. csv-table::
   :header: Parameter, Description
   :widths: 5, 30
   :class: tight-table

   forcerefresh,"| Control the caching-prevention mechanism of the images for a_
↪button.
   | ``0`` : Normal caching behavior (=default)
```

(continues on next page)

(continued from previous page)

```
| ``1``, ``true`` : Prevent caching by adding t=<timestamp> parameter to the url.↵
↵Not all webservers will handle this correctly
| ``2`` : The image is loaded via php, preventing caching. (php must be enabled on↵
↵your Dashticz server)"
```

The previous example will not show a horizontal scroll bar. If you want to have a horizontal scroll bar then remove
:class: tight-table

Alternative 2:

```
=====
Parameter      Description
=====
forcerefresh   | Control the caching-prevention mechanism of the images for a button.
                | ``0`` : Normal caching behavior (=default)
                | ``1``, ``true`` : Prevent caching by adding t=<timestamp> parameter↵
↵to the url. Not all webservers will handle this correctly
                | ``2`` : The image is loaded via php, preventing caching. (php must↵
↵be enabled on your Dashticz server)
=====
```

CHAPTER 9

Indices and tables

- `genindex`
- `modindex`
- `search`