



Public Database Documentation

Release 0.1

Arne de Laat

July 04, 2013

CONTENTS

1	Data access	3
1.1	Download Form	3
1.2	Downloading via Python	3
2	API Tutorial	5
2.1	First look	5
2.2	Javascript example	6
2.3	Python example	7
3	API Reference	9
3.1	API Views Reference	9
4	Status Display Reference	17
4.1	Status Display Views Reference	17
4.2	Nagios Status Reference	19
5	Indices and tables	21
	Python Module Index	23
	Index	25

The HiSPARC Public Database is the interface through which everyone can access the data and our station administration is done.

Contents:

DATA ACCESS

There are several ways in which the HiSPARC data can be accessed:

- Via the Public database Download Form
- Via Python using the SAPHIRE framework (see [SAPHIRE Tutorial](#))

On this page the second method is explained and examples are given. To access metadata, like a list of all stations or information about a specific station see the [API Tutorial](#).

1.1 Download Form

When looking at the data page for a station (i.e. [Kaj Munk College](#)), you will see a ‘Download event summary data’ link on the right. When this link is clicked you will be taken to the [Data Download Form](#). On this page you can select the station, the start and end date for which you want to download the data. There is also an option to download weather data instead of events, however, not all stations gather weather data.

When you click the Submit button without checking the checkbox to Download, the data should (Firefox always download the data) show up in your browser window. If you check the Download box the data will be downloaded to your PC as a csv file. The filename will contain the station number and the dates selected.

This csv file can be read by many programs, like Excel.

1.2 Downloading via Python

This is an example of how to download the data from the Public database in Python. In this example we will download 1 hour of data for station 202 on July 2, 2013.

First load the required libraries (requires the numpy package).

```
>>> from urllib2 import urlopen
>>> from urllib import urlencode
>>> from datetime import datetime
>>> from csv import reader
>>> from StringIO import StringIO
>>> from numpy import genfromtxt
```

Then define the url and place the start and end datetime in the query. To download weather data instead, replace 'events' by 'weather' in the url (and choose a station that has weather data, e.g. 3 or 501).

Note: Do not pass the query to the `urlopen` 'data' argument because that changes the request into a *POST* request, but you need a *GET* request.

```
>>> url = 'http://data.hisparc.nl/data/202/events'
>>> start = str(datetime(2013, 7, 2, 11, 0))
>>> end = str(datetime(2013, 7, 2, 12, 0))
>>> query = urlencode({'download': False, 'start': start, 'end': end})
>>> full_url = url + '?' + query
```

Download the data and store it in a variable

```
>>> data = urlopen(full_url).read()
```

Now use `numpy` to convert the data from csv to a `numpy` array.

```
>>> format = [('date', 'datetime64[D]'), ('time', '|S8'),
              ('timestamp', 'uint32'), ('nanoseconds', 'uint32'),
              ('pulseheights', '4int16'), ('integrals', '4int16'),
              ('n1', 'float32'), ('n2', 'float32'),
              ('n3', 'float32'), ('n4', 'float32'),
              ('t1', 'float32'), ('t2', 'float32'),
              ('t3', 'float32'), ('t4', 'float32')]
>>> a = genfromtxt(StringIO(data), delimiter="\t", dtype=format)
>>> print a[0]
(datetime.date(2013, 7, 2), '11:00:02', 1372762802L, 466307811L,
 [78, 798, -1, -1], [535, 10882, -1, -1],
 0.14720000326633453, 3.854599952697754, -1.0, -1.0,
 345.0, 12.5, -1.0, -1.0)
```


API TUTORIAL

The HiSPARC API (Application Programming Interface) simplifies metadata access from other applications. In this tutorial, we'll give some examples of how this data can be accessed and used with Javascript ([jQuery](#)) and [Python](#). We'll show you how to do some neat things. How can you get a list of all HiSPARC stations in Denmark? What is the position of station 201? Which stations had data on 20 October 2010? How does the number of measured events change during a few weeks? This tutorial will give you an overview of some of possibilities with the HiSPARC API. For details on all available classes and methods, please see the [API Reference](#).

Note: We'll require you to know some basic programming, i.e. to understand what an `if` statement is and `for` loop does. If you are new to coding you can try a tutorial online, for instance [Codecademy](#), we recommend learning Python or jQuery.

2.1 First look

First we will just look at what this API is. The API can be accessed via the internet by opening urls. Instead of a website you get data as a response. This data is formatted as a JSON (JavaScript Object Notation), this format can be understood by many programming languages.

To see what options the API has we will look at it in a browser. Open the following link in your browser (this will not work in Internet Explorer): <http://data.hisparc.nl/api/>.

You should now see some text, like this:

```
{ "base_url": "http://data.hisparc.nl/api/",
  "clusters": "clusters/",
  "clusters_in_country": "countries/{country_id}/",
  "configuration": "station/{station_id}/config/{year}/{month}/{day}/",
  "countries": "countries/",
  "has_data": "station/{station_id}/data/{year}/{month}/{day}/",
  ...
  "subclusters_in_cluster": "clusters/{cluster_id}/" }
```

This is the JSON, it is a dictionary (indicated by the `{` and `}` enclosing brackets): an object which has keys and values. Each key (`"clusters"`, `"has_data"`) refers to a value (`"clusters/"`, `"station/{station_id}/data/{year}/{month}/{day}/"`).

2.1.1 Cluster list

This tells us that if we want a list of all clusters we need to use the clusters option by appending "clusters/" to the base url, resulting in the following: <http://data.hisparc.nl/api/clusters/>.

With this result:

```
[{"name": "Amsterdam",
  "number": 0},
{"name": "Utrecht",
  "number": 1000},
...
{"name": "Karlsruhe",
  "number": 70000}]
```

This JSON is a list (indicated by the [and] enclosing brackets) of dictionaries, one for each cluster. Each dictionary contains the name and number of a cluster. This way information about the network of stations can be retrieved.

2.2 Javascript example

The following code example will generate a webpage which will use the API to get an up-to-date list of stations. It will then show a drop-down menu from which a station can be selected, once you have chosen a station you can click the Get info button to make Javascript get the station information. To try this you can either use this example page: [jsFiddle](#) or create your own HTML file with this code:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script src="http://data.hisparc.nl/media/static/scripts/jquery-1.8.2.min.js"></script>
<script>
    $(function() {
        $.getJSON(
            'http://data.hisparc.nl/api/stations/',
            function(data) {
                var select = $('<select>');
                var id, name;
                for (var i in data) {
                    id = data[i].number;
                    name = data[i].name;
                    select.append($('<option>').attr('value', id).text(id + ' - ' + name));
                }
                $('#station_list').append(select);
            }
        );

        $('#get_station').on('click', function() {
            var id = $('#station_list').find('select').val();
            $.getJSON('http://data.hisparc.nl/api/station/' + id + '/',
                function(data) {
                    $('#station_info').text(JSON.stringify(data, undefined, 4));
                }
            );
        });
    });
</script>
```

```
</head>
<body style="font-family: sans-serif;">
  <h2>Station info</h2>
  <p id="station_list">Choose a station: </p>
  <input type="submit" id="get_station" value="Get info">
  <pre id="station_info"></pre>
</body>
</html>
```

2.3 Python example

In this example we will use several standard Python libraries and the popular plotting library matplotlib (pylab). We start by importing the required libraries, one to get data from the urls, one to make working with dates easy and the plotting library. Then define the start values and prepare two empty lists in which the data can be placed. Then a while loop is used to loop over all days between datum and end_datum, reading each corresponding url. Finally a plot is made, setting the dates against their values.

Start Python and type (copy/paste) the following lines of code:

```
>>> from urllib2 import urlopen
>>> from datetime import date, timedelta
>>> from pylab import plot, show
>>> id = 501
>>> datum = date(2010, 10, 1)
>>> end_datum = date(2011, 2, 1)
>>> base = 'http://data.hisparc.nl/api/station/%d/num_events/%d/%d/%d'
>>> events = []
>>> dates = []
>>> while datum < end_datum:
...     url = urlopen(base % (id, datum.year, datum.month, datum.day))
...     events.append(url.read())
...     dates.append(datum)
...     datum += timedelta(days=1)
...
>>> step(dates, events)
>>> show()
```


API REFERENCE

Application Programming Interface for HiSPARC Public Database

The API simplifies data access for data contained in the [HiSPARC Public Database](#). It was born out of the need for easy access to up-to-date information about stations.

The following packages and modules are included:

urls urls that can be called and will be passed on to functions

views definitions that return specific data from the database

Contents:

3.1 API Views Reference

`class django_publicdb.api.views.Nagios`

`critical = (2, 'CRITICAL')`

`ok = (0, 'OK')`

`unknown = (3, 'UNKNOWN')`

`warning = (1, 'WARNING')`

`django_publicdb.api.views.clusters(request, country_id=None)`

Get cluster list

Retrieve a list of all clusters or only the clusters in a specific country. By cluster we here mean the main clusters, which contain subclusters.

Parameters `country_id` – a country number identifier, give this to only get clusters from a specific country.

Returns list of dictionaries containing the name and number of all clusters that matched the given parameters.

`django_publicdb.api.views.config(request, station_id, year=None, month=None, day=None)`

Get station config settings

Retrieve the entire configuration of a station. If no date is given the latest config will be sent, otherwise the latest on or before the given date.

Parameters

- **station_id** – a station number identifier.
- **year** – the year part of the date.
- **month** – the month part of the date.
- **day** – the day part of the date.

Returns dictionary containing the entire configuration from the HiSPARC DAQ.

```
django_publicdb.api.views.countries(request)
```

Get country list

Retrieve a list of all countries with active stations.

Returns list of dictionaries containing the name and number of all countries.

```
django_publicdb.api.views.get_cluster_dict(country=None)
```

```
django_publicdb.api.views.get_country_dict()
```

```
django_publicdb.api.views.get_pulseheight_drift(request, station_number,
                                                plate_number, year, month,
                                                day, number_of_days)
```

Get pulseheight drift

Parameters

- **station_number** – station number
- **plate_number** – detector number, either 1, 2, 3 or 4.
- **month, day (year)** – date for which to check
- **number_of_days** – number of days over which to determine drift

```
django_publicdb.api.views.get_pulseheight_drift_last_14_days(request,
                                                             station_id,
                                                             plate_number)
```

```
django_publicdb.api.views.get_pulseheight_drift_last_30_days(request,
                                                             station_id,
                                                             plate_number)
```

```
django_publicdb.api.views.get_pulseheight_fit(request, station_number,
                                                plate_number, year=None,
                                                month=None, day=None)
```

Get fit values of the pulseheight distribution for a station on a given day

Retrieve fit values of the pulseheight distribution. The fitting has to be done before and stored somewhere. This function retrieves the fit values from storage and returns to the client. Returns an error message if the values are not found on storage.

Parameters

- **station_number** – a station number identifier.
- **plate_number** – plate number in the range 1..4
- **year** – the year part of the date.
- **month** – the month part of the date.
- **day** – the day part of the date.

Returns dictionary containing fit results of the specified station, plate and date

`django_publicdb.api.views.get_station_dict (subcluster=None)`

`django_publicdb.api.views.get_subcluster_dict (cluster=None)`

`django_publicdb.api.views.has_data (request, station_id, year=None, month=None, day=None)`

Check for presence of cosmic ray data

Find out if the given station has measured shower data, either on a specific date, or at all.

Parameters

- **station_id** – a stationn number identifier.
- **year** – the year part of the date.
- **month** – the month part of the date.
- **day** – the day part of the date.

Returns boolean, True if the given station has shower data, False otherwise.

`django_publicdb.api.views.has_weather (request, station_id, year=None, month=None, day=None)`

Check for presence of weather data

Find out if the given station has measured weather data, either on a specific date, or at all.

Parameters

- **station_id** – a stationn number identifier.
- **year** – the year part of the date.
- **month** – the month part of the date.
- **day** – the day part of the date.

Returns boolean, True if the given station has weather data, False otherwise.

`django_publicdb.api.views.json_dict (dict)`

Create a json HTTPResponse

`django_publicdb.api.views.man (request)`

Give overview of the possible urls

`django_publicdb.api.views.num_events(request, station_id)`

Get total number of events for a station

Retrieve the number of events that a station has measured during its entire operation. The following functions each dig a little deeper, going for a shorter time period.

Parameters `station_id` – a station number identifier.

Returns integer containing the total number of events ever recorded by the given station.

`django_publicdb.api.views.num_events_day(request, station_id, year, month, day)`

Get total number of events for a station on a given date

Retrieve the total number of events that a station has measured on a date.

Parameters

- **station_id** – a station number identifier.
- **year** – the year part of the date.
- **month** – the month part of the date.
- **day** – the day part of the date.

Returns integer denoting the number of events recorded by the station on the given date.

`django_publicdb.api.views.num_events_hour(request, station_id, year, month, day, hour)`

Get number of events for a station in an hour on the given date

Retrieve the total number of events that a station has measured in that hour.

Parameters

- **station_id** – a station number identifier.
- **year** – the year part of the date.
- **month** – the month part of the date.
- **day** – the day part of the date.
- **hour** – the hour for which the number of events is to be retrieved.

Returns integer giving the number of events recorded by the station in hour on date.

`django_publicdb.api.views.num_events_month(request, station_id, year, month)`

Get total number of events for a station in the given month of a year

Retrieve the total number of events that a station has measured during the given month.

Parameters

- **station_id** – a station number identifier.
- **year** – the year in which to look for the month.
- **month** – the month for which the number of events is to be given.

Returns integer containing the total number of events recorded by the given station in the given month of the given year.

`django_publicdb.api.views.num_events_year(request, station_id, year)`

Get total number of events for a station in the given year

Retrieve the total number of events that a station has measured during the given year.

Parameters

- **station_id** – a station number identifier.
- **year** – the year for which the number of events is to be given.

Returns integer containing the total number of events recorded by the given station in the given year.

`django_publicdb.api.views.station(request, station_id)`

Get station info

Retrieve important information about a station.

Parameters **station_id** – a station number identifier.

Returns dictionary containing info about the station. Most importantly, this contains information about the position of the station, including the position of the individual scintillators.

`django_publicdb.api.views.stations(request, subcluster_id=None)`

Get station list

Retrieve a list of all stations or all stations in a subcluster.

Parameters **subcluster_id** – a subcluster number identifier. If given, only stations belonging to that subcluster will be included in the list.

Returns list containing dictionaries which consist of the name and number of each station (matching the subcluster).

`django_publicdb.api.views.stations_with_data(request)`

Get stations with data

Retrieve a list of all stations which have recorded data.

Returns list of dictionaries containing the name and number of each station that has measured events.

`django_publicdb.api.views.stations_with_data_day(request, year, month, day)`

Get stations with data

Retrieve a list of all stations which have data on the given date.

Parameters

- **year** – the year part of the date.
- **month** – the month part of the date.
- **day** – the day part of the date.

Returns list of dictionaries containing the name and number of each station that has measured events on the given date.

`django_publicdb.api.views.stations_with_data_month(request, year, month)`

Get stations with data

Retrieve a list of all stations which have data in the given month.

Parameters

- **year** – the year part of the date.
- **month** – the month part of the date.

Returns list of dictionaries containing the name and number of each station that has measured events in the given month.

`django_publicdb.api.views.stations_with_data_year(request, year)`

Get stations with data

Retrieve a list of all stations which have data in the given year.

Parameters **year** – the year part of the date.

Returns list of dictionaries containing the name and number of each station that has measured events in the given year.

`django_publicdb.api.views.stations_with_weather(request)`

Get stations with weather data

Retrieve a list of all stations which have weather data.

Returns list of dictionaries containing the name and number of each station that has measured weather data.

`django_publicdb.api.views.stations_with_weather_day(request, year, month, day)`

Get stations with weather data

Retrieve a list of all stations which have weather data on the given date.

Parameters

- **year** – the year part of the date.
- **month** – the month part of the date.
- **day** – the day part of the date.

Returns list of dictionaries containing the name and number of each station that has measured weather data on the given date.

`django_publicdb.api.views.stations_with_weather_month(request, year, month)`

Get stations with weather data

Retrieve a list of all stations which have weather data in the given month.

Parameters

- **year** – the year part of the date.
- **month** – the month part of the date.

Returns list of dictionaries containing the name and number of each station that has measured weather data in the given month.

`django_publicdb.api.views.stations_with_weather_year(request, year)`

Get stations with weather data

Retrieve a list of all stations which have weather data in the given year.

Parameters **year** – the year part of the date.

Returns list of dictionaries containing the name and number of each station that has measured weather data in the given year.

`django_publicdb.api.views.subclusters(request, cluster_id=None)`

Get subcluster list

Retrieve a list of all subclusters or all subclusters in a specific cluster.

Parameters **cluster_id** – a cluster number identifier, give this to only get subclusters from this cluster.

Returns list of dictionaries containing the name and number of all subclusters that matched the given parameters.

`django_publicdb.api.views.validate_date(date)`

Check if date is outside HiSPARC project range

If not valid, a 404 (Not Found) should be returned to the user.

Returns boolean, True if the date is in the range, False otherwise.

Contents:

STATUS DISPLAY REFERENCE

Station Status Display

The Status Display provides webpages that display summaries of data measured by stations.

The following packages and modules are included:

nagios definitions that access nagios (vpn.hisparc.nl) to retrieve station status

urls url resolvers

views definitions which return pages with data for the HiSPARC stations

Contents:

4.1 Status Display Views Reference

`django_publicdb.status_display.views.create_histogram(type, station, date)`
Create a histogram object

`django_publicdb.status_display.views.create_plot_object(x_values,
y_series,
x_label,
y_label)`

`django_publicdb.status_display.views.get_barometer_dataset_source(request,
sta-
tion_id,
year,
month,
day)`

`django_publicdb.status_display.views.get_config_source(station_id, type)`

`django_publicdb.status_display.views.get_current_config_source(request,
sta-
tion_id)`

`django_publicdb.status_display.views.get_dataset_source(station_id, year,
month, day,
type)`

`django_publicdb.status_display.views.get_eventtime_histogram_source` (*request*,
station_id,
year,
month,
day)

`django_publicdb.status_display.views.get_gps_config_source` (*request*, *station_id*)

`django_publicdb.status_display.views.get_gpspositions` (*configs*)
Get all unique gps positions from the configs

`django_publicdb.status_display.views.get_histogram_source` (*station_id*,
year, *month*,
day, *type*)

`django_publicdb.status_display.views.get_pulseheight_histogram_source` (*request*,
station_id,
year,
month,
day)

`django_publicdb.status_display.views.get_pulseintegral_histogram_source` (*request*,
station_id,
year,
month,
day)

`django_publicdb.status_display.views.get_temperature_dataset_source` (*request*,
station_id,
year,
month,
day)

`django_publicdb.status_display.views.get_voltage_config_source` (*request*,
station_id)

`django_publicdb.status_display.views.help` (*request*)
Show the static help page

`django_publicdb.status_display.views.nav_calendar` (*station*, *theyear*, *the-month*)
Create a month calendar with links

`django_publicdb.status_display.views.nav_months` (*station*, *theyear*)
Create list of months with links

`django_publicdb.status_display.views.nav_years` (*station*)
Create list of previous years

`django_publicdb.status_display.views.plot_config` (*type*, *configs*)

Create a plot object from station configs

`django_publicdb.status_display.views.plot_dataset` (*type*, *station*, *date*,
log=False)

Create a dataset plot object

`django_publicdb.status_display.views.station` (*request*, *station_id*)

Show most recent histograms for a particular station

`django_publicdb.status_display.views.station_config` (*request*, *station_id*)

Show configuration history for a particular station

`django_publicdb.status_display.views.station_data` (*request*, *station_id*, *year*,
month, *day*)

Show daily histograms for a particular station

`django_publicdb.status_display.views.station_has_data` (*station*)

Check if there is valid event or weather data for the given station

Parameters *station* – Station object for which to check.

Returns boolean indicating if the station has recorded data, either weather or shower, between 2002 and now.

`django_publicdb.status_display.views.station_status` (*request*, *station_id*)

Show daily histograms for a particular station

`django_publicdb.status_display.views.stations` (*request*)

Show the default station list

`django_publicdb.status_display.views.stations_by_country` (*request*)

Show a list of stations, ordered by country, cluster and subcluster

`django_publicdb.status_display.views.stations_by_name` (*request*)

Show a list of stations, ordered by station name

`django_publicdb.status_display.views.stations_by_number` (*request*)

Show a list of stations, ordered by number

`django_publicdb.status_display.views.stations_on_map` (*request*, *coun-*
try=None, *clus-*
ter=None, *subclus-*
ter=None)

Show all stations from a subcluster on a map

Contents:

4.2 Nagios Status Reference

`django_publicdb.status_display.nagios.down_list` ()

Get Nagios page which lists DOWN hosts

Returns list of station short names of stations that are DOWN.

`django_publicdb.status_display.nagios.get_station_status(station, down, problem, up)`

Check if station is in down, problem or up list.

Parameters

- **station** – station_id for which you want the status.
- **down** – list of stations that are DOWN.
- **problem** – list of stations that have a service CRITICAL (but are UP).
- **up** – list of stations that are UP.

Returns string denoting the current status of requested station, if the station occurs in multiple lists, the worst case is returned.

`django_publicdb.status_display.nagios.problem_list()`

Get Nagios page which lists hosts with a problem

Returns list containing station short names of stations for which the host has status OK, but some services are CRITICAL.

`django_publicdb.status_display.nagios.retrieve_station_status(query)`

Get station list from Nagios page which lists hosts of certain level

Parameters *query* – query to filter stations on Nagios.

Returns list of station short names on the given page.

`django_publicdb.status_display.nagios.status_lists()`

Get various station status lists from Nagios

Returns down, problem, up. lists containing station short names that have the status the variable name implies.

`django_publicdb.status_display.nagios.up_list()`

Get Nagios page which lists UP hosts

Returns list of station short names of stations that are OK.

Contents:

INDICES AND TABLES

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

d

- `django_publicdb.api`, [9](#)
- `django_publicdb.api.views`, [9](#)
- `django_publicdb.status_display`, [17](#)
- `django_publicdb.status_display.nagios`,
[19](#)
- `django_publicdb.status_display.views`,
[17](#)

C

clusters() (in module `django_publicdb.api.views`), 9
 config() (in module `django_publicdb.api.views`), 9
 countries() (in module `django_publicdb.api.views`), 10
 create_histogram() (in module `django_publicdb.status_display.views`), 17
 create_plot_object() (in module `django_publicdb.status_display.views`), 17
 critical (in `django_publicdb.api.views.Nagios` attribute), 9

D

`django_publicdb.api` (module), 9
`django_publicdb.api.views` (module), 9
`django_publicdb.status_display` (module), 17
`django_publicdb.status_display.nagios` (module), 19
`django_publicdb.status_display.views` (module), 17
 down_list() (in module `django_publicdb.status_display.nagios`), 19

G

get_barometer_dataset_source() (in module `django_publicdb.status_display.views`), 17
 get_cluster_dict() (in module `django_publicdb.api.views`), 10
 get_config_source() (in module `django_publicdb.status_display.views`), 17
 get_country_dict() (in module `django_publicdb.api.views`), 10
 get_current_config_source() (in module `django_publicdb.status_display.views`), 17

get_dataset_source() (in module `django_publicdb.status_display.views`), 17
 get_eventtime_histogram_source() (in module `django_publicdb.status_display.views`), 17
 get_gps_config_source() (in module `django_publicdb.status_display.views`), 18
 get_gpspositions() (in module `django_publicdb.status_display.views`), 18
 get_histogram_source() (in module `django_publicdb.status_display.views`), 18
 get_pulseheight_drift() (in module `django_publicdb.api.views`), 10
 get_pulseheight_drift_last_14_days() (in module `django_publicdb.api.views`), 10
 get_pulseheight_drift_last_30_days() (in module `django_publicdb.api.views`), 10
 get_pulseheight_fit() (in module `django_publicdb.api.views`), 10
 get_pulseheight_histogram_source() (in module `django_publicdb.status_display.views`), 18
 get_pulseintegral_histogram_source() (in module `django_publicdb.status_display.views`), 18
 get_station_dict() (in module `django_publicdb.api.views`), 11
 get_station_status() (in module `django_publicdb.status_display.nagios`), 19
 get_subcluster_dict() (in module `django_publicdb.api.views`), 11
 get_temperature_dataset_source() (in module `django_publicdb.status_display.views`), 18
 get_voltage_config_source() (in module `django_publicdb.status_display.views`), 18

H

`has_data()` (in module `django_publicdb.api.views`),
[11](#)

`has_weather()` (in module
`django_publicdb.api.views`), [11](#)

`help()` (in module
`django_publicdb.status_display.views`),
[18](#)

J

`json_dict()` (in module `django_publicdb.api.views`),
[11](#)

M

`man()` (in module `django_publicdb.api.views`), [11](#)

N

`Nagios` (class in `django_publicdb.api.views`), [9](#)

`nav_calendar()` (in module
`django_publicdb.status_display.views`),
[18](#)

`nav_months()` (in module
`django_publicdb.status_display.views`),
[18](#)

`nav_years()` (in module
`django_publicdb.status_display.views`),
[18](#)

`num_events()` (in module
`django_publicdb.api.views`), [11](#)

`num_events_day()` (in module
`django_publicdb.api.views`), [12](#)

`num_events_hour()` (in module
`django_publicdb.api.views`), [12](#)

`num_events_month()` (in module
`django_publicdb.api.views`), [12](#)

`num_events_year()` (in module
`django_publicdb.api.views`), [13](#)

O

`ok` (`django_publicdb.api.views.Nagios` attribute), [9](#)

P

`plot_config()` (in module
`django_publicdb.status_display.views`),
[18](#)

`plot_dataset()` (in module
`django_publicdb.status_display.views`),
[19](#)

`problem_list()` (in module
`django_publicdb.status_display.nagios`), [20](#)

R

`retrieve_station_status()` (in module
`django_publicdb.status_display.nagios`), [20](#)

S

`station()` (in module `django_publicdb.api.views`), [13](#)

`station()` (in module
`django_publicdb.status_display.views`),
[19](#)

`station_config()` (in module
`django_publicdb.status_display.views`),
[19](#)

`station_data()` (in module
`django_publicdb.status_display.views`),
[19](#)

`station_has_data()` (in module
`django_publicdb.status_display.views`),
[19](#)

`station_status()` (in module
`django_publicdb.status_display.views`),
[19](#)

`stations()` (in module `django_publicdb.api.views`), [13](#)
`stations()` (in module
`django_publicdb.status_display.views`),
[19](#)

`stations_by_country()` (in module
`django_publicdb.status_display.views`),
[19](#)

`stations_by_name()` (in module
`django_publicdb.status_display.views`),
[19](#)

`stations_by_number()` (in module
`django_publicdb.status_display.views`),
[19](#)

`stations_on_map()` (in module
`django_publicdb.status_display.views`),
[19](#)

`stations_with_data()` (in module
`django_publicdb.api.views`), [13](#)

`stations_with_data_day()` (in module
`django_publicdb.api.views`), [13](#)

`stations_with_data_month()` (in module
`django_publicdb.api.views`), [14](#)

`stations_with_data_year()` (in module
`django_publicdb.api.views`), [14](#)

stations_with_weather() (in module
django_publicdb.api.views), [14](#)
stations_with_weather_day() (in module
django_publicdb.api.views), [14](#)
stations_with_weather_month() (in module
django_publicdb.api.views), [14](#)
stations_with_weather_year() (in module
django_publicdb.api.views), [15](#)
status_lists() (in module
django_publicdb.status_display.nagios), [20](#)
subclusters() (in module
django_publicdb.api.views), [15](#)

U

unknown (django_publicdb.api.views.Nagios at-
tribute), [9](#)
up_list() (in module
django_publicdb.status_display.nagios), [20](#)

V

validate_date() (in module
django_publicdb.api.views), [15](#)

W

warning (django_publicdb.api.views.Nagios at-
tribute), [9](#)