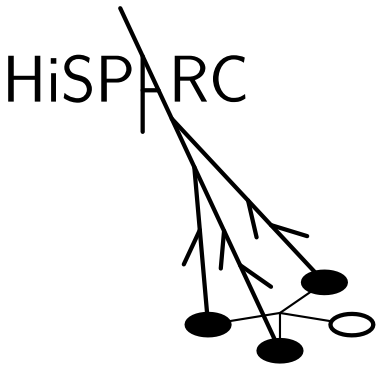


Public Database Documentation

Release 0.3

Arne de Laat

June 17, 2015

CONTENTS

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

Contents:

CHAPTER

ONE

DATA ACCESS

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

- Via the Public database download forms.
- Via Python using the SAPPHiRE framework (see SAPPHiRE Tutorial).
- Via the jSparc web applications.

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 downloads 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.

This csv file can be read by many programs, including Excel. Use the Import option in Excel to make it recognize tabs as delimiter between columns.

1.2 Downloading via Python

Note: An easy to use module has been added to SAPPHiRE to download data from the ESD. The following is an old (but working) example.

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 datetime import datetime
>>> from urllib import urlencode
>>> from urllib2 import urlopen
>>> 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', '4int32'),
              ('n1', 'float32'), ('n2', 'float32'),
              ('n3', 'float32'), ('n4', 'float32'),
              ('t1', 'float32'), ('t2', 'float32'),
              ('t3', 'float32'), ('t4', 'float32'),
              ('t_trigger', '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, 345.0)
```

CHAPTER

TWO

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 Codeacademy, 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>
<head>
<script src="http://code.jquery.com/jquery-1.11.1.min.js"></script>
<script>
    $(function() {
        // Get an up-to-date list of HiSPARC stations
        $.getJSON(
            'http://data.hisparc.nl/api/stations/',
            function(data) {
                // Create the drop-down menu
                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);});

        // Attach a function to the Get info button
        $('#get_station').on('click', function() {
            var id = $('#station_list').find('select').val();
            // Get info for selected station and display it in a nice way
            $.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 perpare 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 (or copy/paste without the :code:'>>> ') 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()
```

2.3.1 SAPPHiRE

The HiSPARC Python framework SAPPHiRE includes an API module. This module simplifies access to the API. See the SAPPHiRE documentation for more information: http://docs.hisparc.nl/sapphire/.

CHAPTER

THREE

HISPARC MAPS

Each HiSPARC station is equipped with a GPS antenne. This GPS is used to for time synchronization between stations and to determine the exact location of each station. All these locations are stored in our database. The GPS positions can be found on the data pages of stations, via the API (*API Reference*) and in the raw data.

3.1 OpenStreetMap

Using the free OpenStreetMap service and the OpenLayers library we are able to visualize the detector network by showing the locations of the stations on a map.

Here is an overview of the network: Stations on map

3.1.1 Controls

To keep the map clean we do not show any controls on the map. However, navigation is very intuitive and similar to what you may be familiar with from other map services. The controls are as follows:

- **Zooming**
  - Double click on a location you want to zoom in on
  - Scroll with your mouse or trackpad to zoom in and out
- **Moving**
  - Click and drag the map to move it around

Note: When you zoom in far enough the station numbers will be shown above the station indicators.

3.1.2 Status

The stations on the map can have one of 4 colors to indicate the current status of that station. The status is retrieved from the HiSPARC Monitoring System when the page loads.

- Green: when a station is operating properly

- Yellow: it is responding to the server but has a problem
- Red: it is completely unresponsive (offline)
- Blue: it is is no longer active or the status can't be determined.

3.2 Embedding

On several public database pages a map is embedded to show the station location or provide an overview of multiple stations. Moreover, maps are also used on our main website to give an overview of the station organization and clustering (e.g. Bristol, Science Park). This is accomplished by placing an iframe on those pages that shows another page which only has the map as its content. Example code:

```
<iframe src="http://data.hisparc.nl/maps/Netherlands/Amsterdam/Science%20Park/"
        scrolling="no" frameborder="0" width="600" height="300"></iframe>
```

Result:

3.2.1 Syntax

To show a map of a specific region or location, use the syntax explained here. First start with the base url:

```
http://data.hisparc.nl/maps/
```

When no extra options are given the page zooms and positions the map such that all stations fit in the window. But you can also focus on a specific region or station. Several levels of regions are possible:

```
http://data.hisparc.nl/maps/[Country]/[Cluster]/[Subcluster]/
http://data.hisparc.nl/maps/[Station id]/
```

An overview of countries, clusters and subclusters can be found on http://data.hisparc.nl/ . First you can choose to focus on a Country:

- http://data.hisparc.nl/maps/Netherlands/
- http://data.hisparc.nl/maps/Denmark/

Then focus more closely on a cluster, note that you also need to give the country:

- http://data.hisparc.nl/maps/Netherlands/Enschede/
- http://data.hisparc.nl/maps/Netherlands/Utrecht/
- http://data.hisparc.nl/maps/United%20Kingdom/Bristol/

And to focus on a subcluster, also specifying the country and cluster:

- http://data.hisparc.nl/maps/Netherlands/Amsterdam/Zaanstad/
- http://data.hisparc.nl/maps/Netherlands/Enschede/Enschede/

Finally you can also focus on one specific station by simply giving its station number:

- http://data.hisparc.nl/maps/8005/

- http://data.hisparc.nl/maps/8103/

CHAPTER

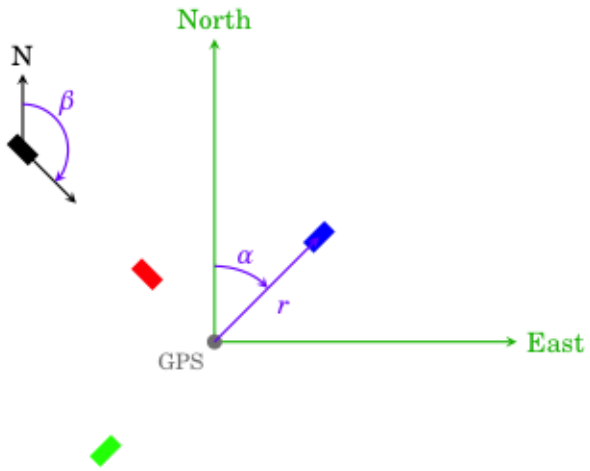
FOUR

HISPARC STATION LAYOUT

Each HiSPARC station has a GPS antenne and either 2 or 4 scintillator detectors. This GPS is used to determine the exact location of the station. However, the GPS is often not at the center of the station, moreover, not all stations are oriented the same way, and the distances between detector may differ. For reconstruction of the measured data it is important to know the location of each detector, this section explains how the location of the detectors can be measured and communicated to us.

4.1 Compass coordinates

The coordinate system we have chosen for describing the position of detectors is illustrated in the following figure.



For each detector 3 (or 4) coordinates need to be determined:

- First the distance (`r`) from the GPS to the center of the scintillator (in meters).
- Next the alpha angle (α) which is the clock-wise turning angle between North and the detector as seen from the GPS (in degrees).
- A height (`z`) coordinate can be measured if there is a significant altitude difference between the GPS antenna and the detectors, or if the detectors are not at the same height. Up is positive (in meters).
- The rotation of the detector is described by the beta angle (β), which is the clock-wise turning rotation of the long side of the detector relative to North (in degrees).

For more information about the coordinate systems used in HiSPARC see: Coordinate systems and units in HiSPARC.

For Dutch schools we have an assignment sheet in the infopakket which walks students through the process of measuring the station layout: De stationsplattegrond.

4.2 Submitting the measurements

New layouts can be submitted via the layout submit form. The submitted measurements will be reviewed before they are accepted and stored in the database.

4.3 Accessing the data

If you are analysing data or are making a schematic drawing of the station layout you can access the detector coordinates via the *API*.

CHAPTER

FIVE

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:

5.1 API Views Reference

class Nagios

critical = (2, 'CRITICAL')

ok = (0, 'OK')

unknown = (3, 'UNKNOWN')

warning = (1, 'WARNING')

clusters(*request*, *country_number=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_number** – 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.

config(*request*, *station_number*, *year=None*, *month=None*, *day=None*)
Get station config settings

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

Parameters

- **station_number** – 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.

countries(*request*)
Get country list

Retrieve a list of all countries.

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

get_cluster_dict(*country=None*)

get_country_dict()

get_event_traces(*request*, *station_number*, *ext_timestamp*)
Get the traces for an event

Parameters

- **station_number** – a station number identifier.
- **ext_timestamp** – extended timestamp (nanoseconds since UNIX epoch).
- **raw** – (optional, GET) if present get the raw trace, i.e. without subtracted baseline.

Returns two or four traces.

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

get_pulseheight_drift_last_14_days(*request*, *station_number*, *plate_number*)

get_pulseheight_drift_last_30_days(*request*, *station_number*, *plate_number*)

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 date

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 meesage 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

get_station_dict (*subcluster=None*)
Return list of station numbers and names

For all non-test stations in the given subcluster

get_subcluster_dict (*cluster=None*)

has_data (*request*, *station_number*, *type=None*, *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_number** – a stationn number identifier.
- **type** – the data type, either events or weather.
- **month, day** (*year,*) – the date, this has to be within the time HiSPARC has been operational and can be as specific as you desire.

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

json_dict (*dict*)
Create a json HTTPResponse

man (*request*)
Give overview of the possible urls

num_events (*request*, *station_number*, *year=None*, *month=None*, *day=None*, *hour=None*)
Get number of events for a station

Retrieve the number of events that a station has measured during its entire operation or during a specific period, which can be a year, month, day or an hour.

Parameters

- **station_number** – a stationn number identifier.
- **month, day, hour** (*year,*) – the date, this has to be within the time HiSPARC has been operational and can be as specific as you desire.

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

station(*request*, *station_number*, *year=None*, *month=None*, *day=None*)
Get station info

Retrieve important information about a station. If no date if given the latest valid info will be sent, otherwise the latest on or before the given date.

Parameters

- **station_number** – 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 info about the station. Most importantly, this contains information about the position of the station, including the position of the individual scintillators.

stations(*request*, *subcluster_number=None*)
Get station list

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

Parameters **subcluster_number** – 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).

stations_with_data(*request*, *type=None*, *year=None*, *month=None*, *day=None*)
Get stations with event or weather data

Retrieve a list of all stations which have recorded events or weather data in the given year, month, day or at all.

Parameters

- **type** – data type to check for, either weather or events.
- **month, day** (*year,*) – the date, this has to be within the time HiSPARC has been operational and can be as specific as you desire.

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

subclusters(*request*, *cluster_number=None*)
Get subcluster list

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

Parameters **cluster_number** – 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.

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:

CHAPTER

SIX

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:

6.1 Status Display Views Reference

create_histogram(*type*, *station*, *date*)
Create a histogram object

create_histogram_network(*type*, *date*)
Create a histogram object

create_plot_object(*x_values*, *y_series*, *x_label*, *y_label*)

get_barometer_dataset_source(*request*, *station_number*, *year*, *month*, *day*)

get_coincidencenumber_histogram_source(*request*, *year*, *month*, *day*)

get_coincidencetime_histogram_source(*request*, *year*, *month*, *day*)

get_config_source(*station_number*, *type*)

get_current_config_source(*request*, *station_number*)

get_dataset_source(*year*, *month*, *day*, *type*, *station_number*)

get_detector_timing_offsets(*station_number*)

get_detector_timing_offsets_source(*request*, *station_number*)

get_eventtime_histogram_source(*request*, *station_number*, *year*, *month*, *day*)

get_gps_config_source(*request*, *station_number*)

get_gpspositions(*configs*)
Get all unique gps positions from the configs

get_histogram_source(*year*, *month*, *day*, *type*, *station_number=None*)

get_pulseheight_histogram_source(*request*, *station_number*, *year*, *month*, *day*)

get_pulseintegral_histogram_source(*request*, *station_number*, *year*, *month*, *day*)

get_station_layout_source(*request*, *station_number*)

get_temperature_dataset_source(*request*, *station_number*, *year*, *month*, *day*)

get_voltage_config_source(*request*, *station_number*)

help(*request*)
Show the static help page

nav_calendar(*theyear*, *themonth*, *station=None*)
Create a month calendar with links

nav_months(*theyear*, *station*)
Create list of months with links

nav_months_network(*theyear*)
Create list of months with links

nav_years(*station=None*)
Create list of previous years

nav_years_network()
Create list of previous years

network_coincidences(*request*, *year=None*, *month=None*, *day=None*)
Show daily coincidences histograms for the entire network

plot_config(*type*, *configs*)
Create a plot object from station configs

plot_dataset(*type*, *station*, *date*, *log=False*)
Create a dataset plot object

plot_timing_offsets(*station_number*)
Create a plot object from station configs

station(*request*, *station_number*)
Show most recent histograms for a particular station

station_config(*request*, *station_number*)
Show configuration history for a particular station

station_data(*request*, *station_number*, *year*, *month*, *day*)
Show daily histograms for a particular station

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.

station_latest(*request*, *station_number*)
Show daily histograms for a particular station

station_status(*request*, *station_number*)
Show Nagios status for a particular station

stations(*request*)
Show the default station list

stations_by_country(*request*)
Show a list of stations, ordered by country, cluster and subcluster

stations_by_name(*request*)
Show a list of stations, ordered by station name

stations_by_number(*request*)
Show a list of stations, ordered by number

stations_on_map(*request*, *country=None*, *cluster=None*, *subcluster=None*)
Show all stations from a subcluster on a map

stations_with_data()
Get list of station numbers with valid event or weather data

Returns list with station numbers for stations that recorded data, either weather or shower, between 2002 and now.

Contents:

6.2 Nagios Status Reference

down_list()
Get Nagios page which lists DOWN hosts

Returns set of station number of stations that are DOWN.

get_station_status(*station_number*, *down*, *problem*, *up*)
Check if station is in down, problem or up list.

Parameters

- **station_number** – station 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.

get_status_counts(*down*, *problem*, *up*)
Get the lengths of the status lists

Parameters

- **down** – set of stations that are DOWN.
- **problem** – set of stations that have a service CRITICAL (but are UP).
- **up** – set of stations that are UP and have no services CRITICAL.

Returns dictionary containing the counts of stations with a status.

pc_name_to_station_number(*shortnames*)
Convert list of pc names to station numbers

Parameters **shortnames** – list of pc names.

Returns station numbers that have a pc with name in the shortnames.

problem_list()
Get Nagios page which lists hosts with a problem

Returns set containing station number of stations for which the host has status OK, but some services are CRITICAL.

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.

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.

up_list()
Get Nagios page which lists UP hosts

Returns set of station numbers of stations that are OK.

Contents:

CHAPTER

SEVEN

INDICES AND TABLES

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

d

C

D

G

H

J

M

N

O

P

R

S

U

V

W