Developing a python NETDATA collector

Introduction

So, you want to make a custom collector for NETDATA?
These are some basic guidelines for a python NETDATA collector. A python collector for NETDATA is a python script which can gather data from an external source and transform these data into charts to be displayed by NETDATA dashboard. The basic jobs of the collector are:

- Gather the data from the producer.
- Create the required charts.
- Parse the data to extract or create the actual data to be represented.
- Assign the correct values to the charts
- Set the order for the charts to be displayed.
- Give the charts data to NETDATA for visualization.

The basic elements of a NETDATA collector are:

- ORDER[] : A list containing the charts to be displayed.
- CHARTS{} : A dictionary containing the details for the charts to be displayed.
- data{} : A dictionary containing the values to be displayed.
- get_data() : The basic function of the collector which will retrun to NETDATA the correct values.

Data gathering

The input to the collector can be any program that can print to stdout. Common input sources for collectors can be logfiles, http requests, executables.

Chart creation

For the data to be represented in the NETDATA dashboard, the developer should first create the required charts. Charts (in general) are defined by several characteristics: Title, legend, units, type and presented values are some of these characteristics. Programmatically each chart is represented as a dictionary entry:

```
chart= {
        "chart_name":
        {
            “options”: [option_list],
            "lines": [
                        [dimension_list]
                    ]
        }
    }
```

To set the chart options, the "**options**" field is used which is a list in the form:
"**options**": [name, title, units, family, context, charttype] where:

- name: The name of the chart
- title : The title to be displayed in the chart
- units : The units for this chart
- family: An identifier used to group charts together (can be null)
- context: An identifier used to group contextually similar charts together (can be null)
- charttype: Either "line", "area" or "stacked". If null line is the default value.

Once the chart has been defined the dimensions of the chart should be defined as well. Dimensions are basically the metrics to be represented in this chart and each chart can have more than one dimension. In order to define the dimensions, the "**lines**" list should be filled in with the required dimensions. Each dimension is a list:
"**dimension**" : [id, name, algorithm , multiplier ,divisor]

- id : The id of the dimension. Mandatory unique field (string) required in order to set a value.
- name : The name to be presented in the chart. If null id will be used.
- algorithm : Can be absolute or incremental. If null absolute is used. Incremental shows the difference from the previous value.
- multiplier : an integer value to divide the collected value, if null, 1 is used
- divisor : an integer value to divide the collected value, if null, 1 is used
  The multiplier/divisor fields are used in cases where the value to be displayed should be decimal since NETDATA only gathers integer values.

Parse the data to extract or create the actual data to be represented.

Once the data is received, the collector should process it in order to get the values required. If for example the received data is a json string, a basic parsing should be performed to get the required data to be used for the charts.

Assign the correct values to the charts

Once we have processed our data and got the required values , we need to assign those values to the charts we have created. This is done using the **data** dictionary which is in the form:
'"**data**": {dimension_id: value } where:

- dimension_id: The id of a defined dimension in a created chart.
- value: The numerical value to associate with this dimension.

Set the order for the charts to be displayed.

Once our charts have been created, we need to set them in order of appearance. To do this we use the **ORDER** list which is in the form:
"**ORDER**": [chart_name_1,chart_name_2, …., chart_name_X] where
chart_name_x: is the chart name to be shown in X order

Give the charts data to NETDATA for visualization.

Our collector should just rerun the data dictionary. If everything is set correctly the charts should be updated with the correct values.

A simple example

Our first collector is called howto_basic.chart.py and is basically a glorified copy of the example.chart.py provided with NETDATA. A simple chart is created which has one dimension. Each time the value is randomly generated. Since almost every intro in programming languages has a “hello world” example, this is the way we can print debug messages in NETDATA collectors.:

```
self.debug(“Hello World”)
```

The source code of the collector can be found in APPENDIX A. By following the steps described in the previous section:

Data gathering

No data gathering, random values will be used.

Chart creation

For this collector the chart was created as follows:

```
CHARTS = {
   "demo_chart": {
      'options': ["demo_name", "demo_title", "demo_units", "demo_family", "demo_context", "line"],
      "lines": [
         ["demo_line_id","demo_line_name"]
      ]
   }
}
```

Parse the data to extract or create the actual data to be represented.

No data processing, random values will be used.

```
random_value = random.randint(0,100)
```

Assign the correct values to the charts

Assign the correct values to the charts we have created.

```
data["demo_line_id"]  = random_value
```

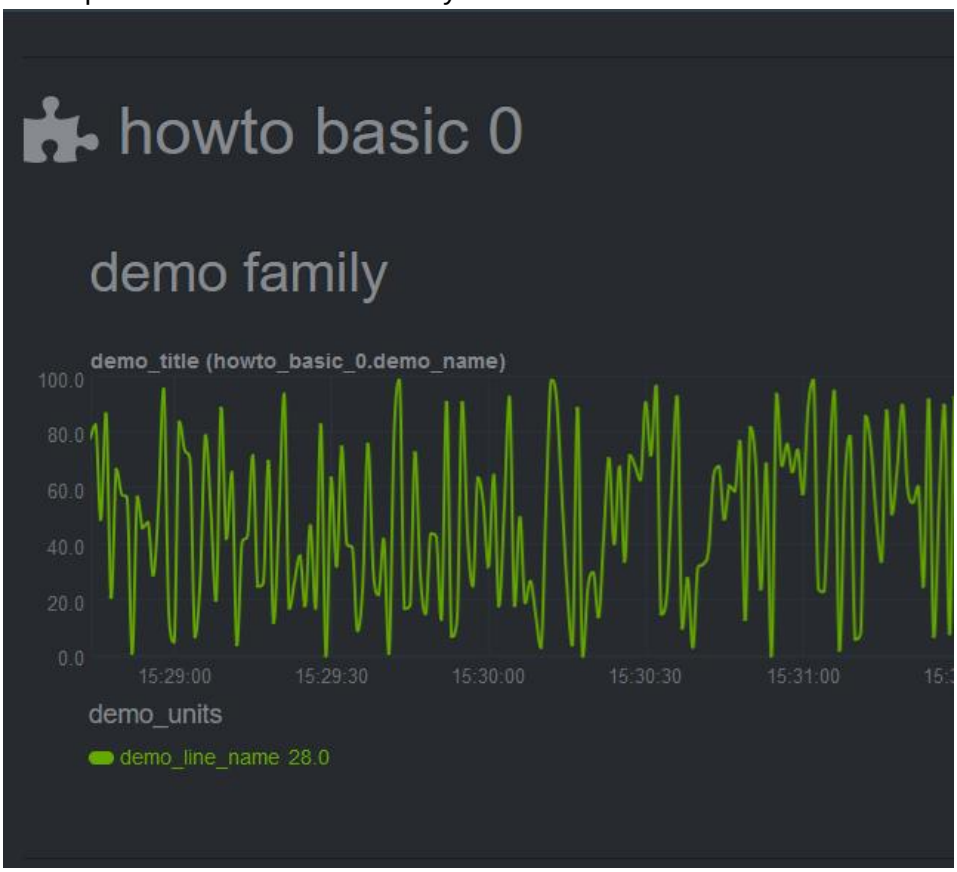
Set the order for the charts to be displayed.

```
ORDER = [
    "demo_chart"
]
```

Give the charts data to NETDATA for visualization.

```
return data
```

A snapshot of the chart created by this collector :



Weather station data visualization (1/3)

Let's assume that we want to visualize data regarding a weather metering station. We will assume that we use a web site that can provide us (through http) the data a monitoring station has gathered. We will assume that the data provided are numeric values regarding the temperature, humidity and pressure. Furthermore, we will assume that we also get min, max and average values for these metrics. The source code of the collector can be found in APPENDIX B.

```
weather_metrics=[
        "temp","av_temp","min_temp","max_temp",
        "humid","av_humid","min_humid","max_humid",
        "pressure","av_pressure","min_pressure","max_pressure",
        ]
```

Data gathering

The actual gathering is out of scope for this document. A mentioned we could use several services provided by NETDATA to get the data. In our scenario we have assumed an HTTP connection which returns the requested values.

Chart creation

For our first visualization we want to have a chart that only shows the latest temperature metric. For this collector the chart was created as follows:

```
CHARTS = {
   "temp_current": {
       "options": ["my_temp", "Temperature", "Celsius", "TEMP", "weather_station", "line"],
      "lines": [
         ["current_temp_id","current_temperature"]
      ]
   }
}
```

Parse the data to extract or create the actual data to be represented.

A standard practice would be to either get the data on json format or transform them to json format. We use a dictionary to give this format and issue random values to simulate received data. We iterate through the names of the expected values and create a dictionary with key the name of each metric: and value a random value.

```
weather_data=dict()
    weather_metrics=[
                "temp","av_temp","min_temp","max_temp",
                "humid","av_humid","min_humid","max_humid",
                "pressure","av_pressure","min_pressure","max_pressure",
                ]

    def populate_data(self):
        for metric in self.weather_metrics:
            self.weather_data[metric]=random.randint(0,100)
```

Assign the correct values to the charts

Our chart has a dimension called “current_temp_id” which should have the temperature value received.

```
data['current_temp_id'] = self.weather_data["temp"]
```

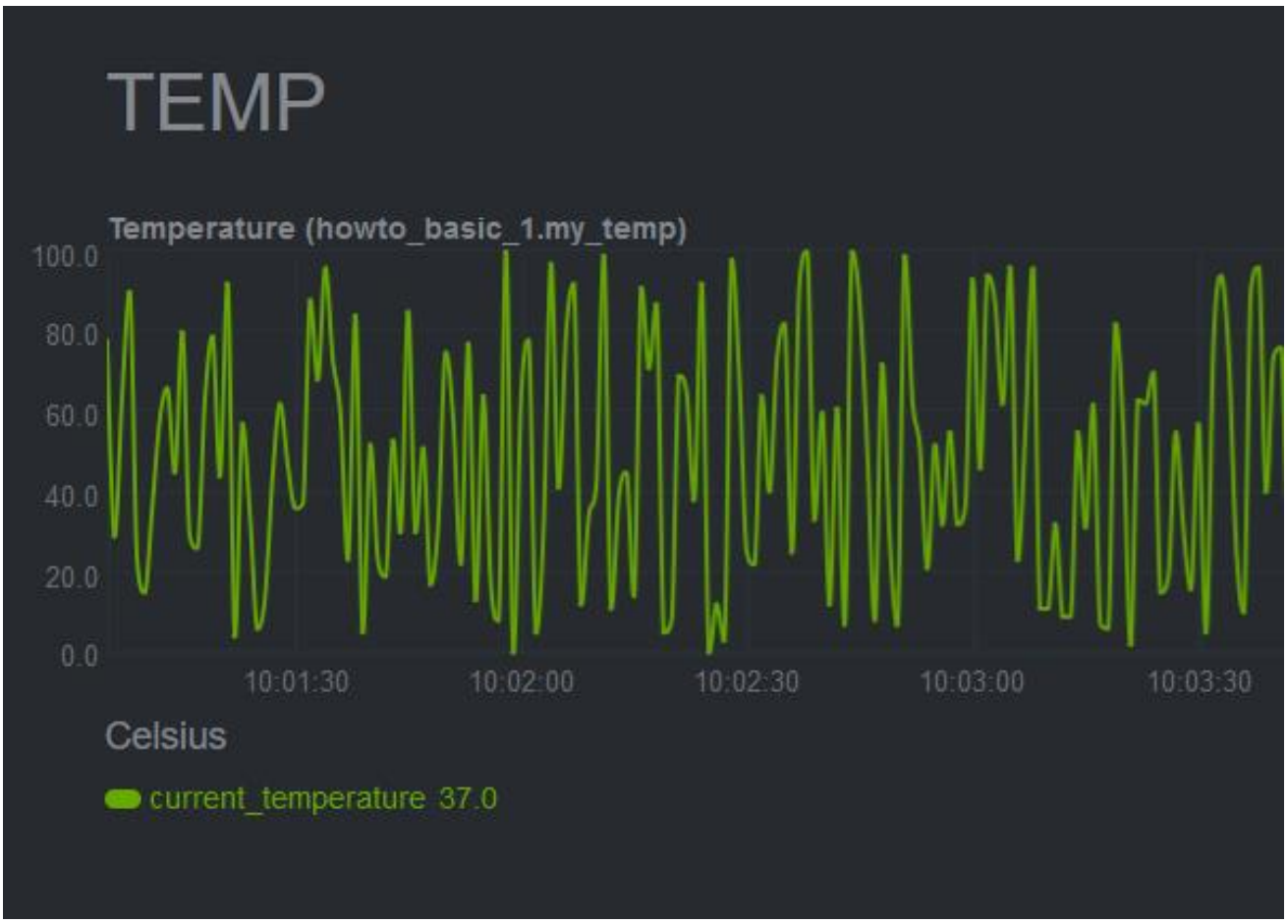
Set the order for the charts to be displayed.

```
ORDER = [
    " "temp_current""
]
```

Give the charts data to NETDATA for visualization.

```
return data
```

A snapshot of the chart created by this collector :



Weather station data visualization (2/3)

We will continue working on our weather station data visualization example. In order to enrich our example, we would like to add another chart in our collector which will present the humidity metric. (In **bold** will be the changes from the previous example)

Data gathering

Same as previous example.

Chart creation

In our previous example we simple have to add a new entry in the CHARTS dictionary with the definition for the new chart.

```
CHARTS = {
    'temp_current': {
        'options': ['my_temp', 'Temperature', 'Celsius', 'TEMP', 'weather_station', 'line'],
        'lines': [
            ['current_temperature']
        ]
    },
    'humid_current': {
        'options': ['my_humid', 'Humidity', '%', 'HUMIDITY', 'weather_station', 'line'],
        'lines': [
            ['current_humidity']
        ]
    }
}
```

Parse the data to extract or create the actual data to be represented.

Same as previous example.

Assign the correct values to the charts

Our chart has a dimension called “current_temp_id” which should have the temperature value received.

```
data['current_temperature'] = self.weather_data["temp"]
data['current_humidity'] = self.weather_data["humid"]
```

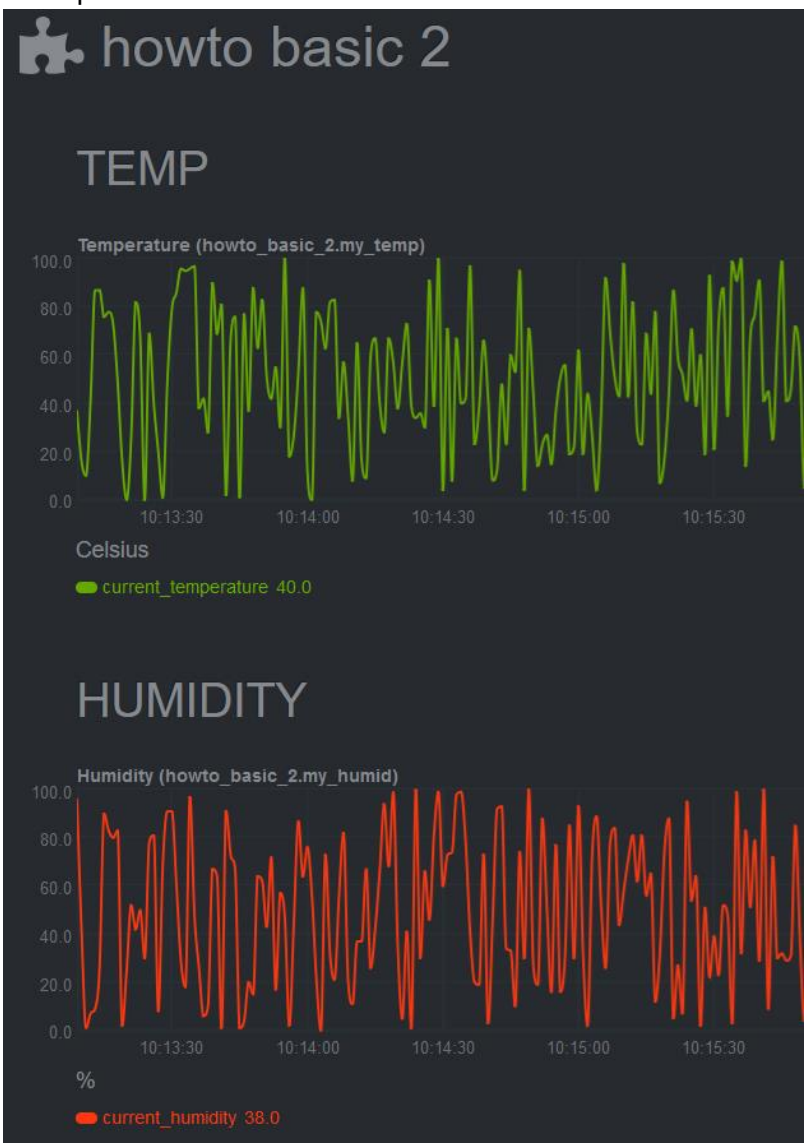
Set the order for the charts to be displayed.

```
ORDER = [
    'temp_current',
    'humid_current'
]
```

Give the charts data to NETDATA for visualization.

Same as previous example.

A snapshot of the modified chart:



Weather station data visualization (3/3)

We will continue working on our weather station data visualization example. In order to enrich our example, we would like to add another chart in our collector which will the average and min max values for temperature. (In **bold** will be the changes from the previous example)

Data gathering

Same as previous example.

Chart creation

In our previous example we simply add a new entry in the CHARTS dictionary with the definition for the new chart. Since we want 3 values represented in this this chart, we add 3 dimensions. We also use the same FAMILY value in the charts (TEMP) so that those two charts can be grouped together.

```
CHARTS = {
    'temp_current': {
        'options': ['my_temp', 'Temperature', 'Celsius', 'TEMP', 'weather_station', 'line'],
        'lines': [
            ['current_temperature']
        ]
    },
    'temp_stats': {
        'options': ['stats_temp', 'Temperature', 'Celsius', 'TEMP', 'weather_station', 'line'],
        'lines': [
            ['min_temperature'],
            ['max_temperature'],
            ['avg_temperature']
        ]
    },
    'humid_current': {
        'options': ['my_humid', 'Humidity', '%', 'HUMIDITY', 'weather_station', 'line'],
        'lines': [
            ['current_humidity']
        ]
    }

}
```

Parse the data to extract or create the actual data to be represented.

Same as previous example.

Assign the correct values to the charts

Our new chart has 3 dimensions called min_temperature, max_temperature, avg_temperature which should be assigned a value.

```
        data['current_temperature']  = self.weather_data["temp"]
        data['min_temperature']=self.weather_data["min_temp"]
        data['max_temperature']=self.weather_data["max_temp"]
        data['avg_temperature']=self.weather_data["av_temp"]
        data['current_humidity']=self.weather_data["humid"]
```

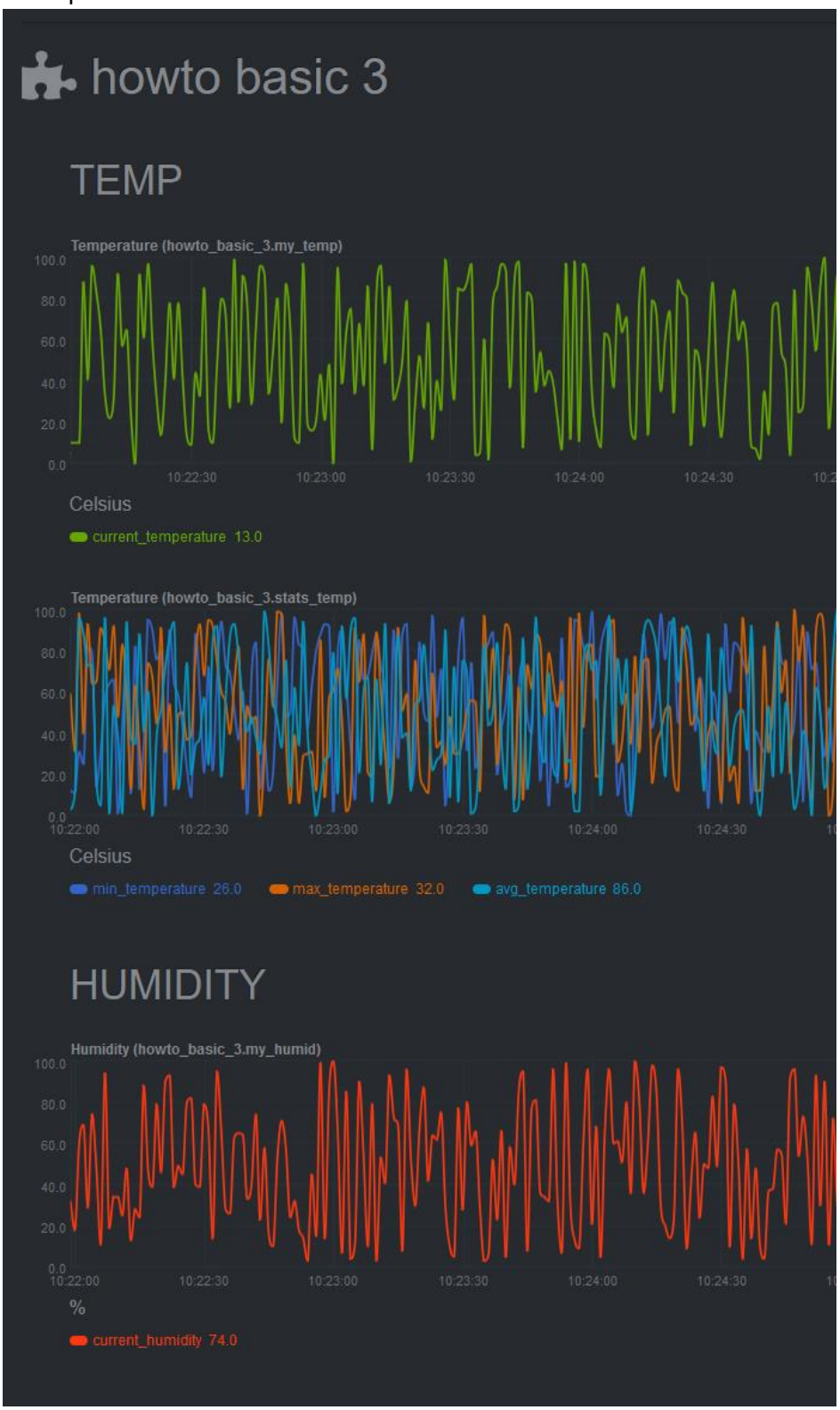
Set the order for the charts to be displayed.

```
ORDER = [
    'temp_current',
    'temp_stats'
    'humid_current'
]
```

Give the charts data to NETDATA for visualization.

Same as previous example.

A snapshot of the modified chart:



APPENDIX A: Source code of howto_basic_0.chart.py

```
# -*- coding: utf-8 -*-
# Description: howto  python.d module
# Author: Panagiotis Papaioannou (papajohn-uop)
# SPDX-License-Identifier:  GPL-3.0-or-later


from bases.FrameworkServices.SimpleService  import SimpleService



import random


NETDATA_UPDATE_EVERY=1
priority = 90000


ORDER = [
   "demo_chart"
]

CHARTS = {
   "demo_chart": {
      'options':  ["demo_name",  "demo_title",  "demo_units",  "demo_family",  "demo_context",
"line"],
      "lines": [
         ["demo_line_id","demo_line_name"]
      ]
    }
}

class Service(SimpleService):
   def __init__(self, configuration=None, name=None):
      SimpleService.__init__(self, configuration=configuration, name=name)
      self.order = ORDER
      self.definitions = CHARTS
      #values to show at graphs
      self.values=dict()


   @staticmethod
   def check():
      return True


   def logMe(self,msg):
      self.debug(msg)
```

```
def get_data(self):
    #The data dict is basically all the values to be represented
    # The entries are in the format: { "dimension": value}
    #And each "dimension" shoudl belong to a chart.
    data = dict()
    self.logMe("Demo collector")
    random_value = random.randint(0,100)
    data["demo_line_id"] = random_value

    return data
```

APPENDIX B: Source code of howto_basic_1.chart.py

```
# -*- coding: utf-8 -*-
# Description: howto weather station netdata python.d module
# Author: Panagiotis Papaioannou (papajohn-uop)
# SPDX-License-Identifier: GPL-3.0-or-later


from bases.FrameworkServices.SimpleService import SimpleService



import random


NETDATA_UPDATE_EVERY=1
priority = 90000


ORDER = [
    "temp_current"
]

CHARTS = {
    "temp_current": {
        "options": ["my_temp", "Temperature", "Celsius", "TEMP", "weather_station", "line"],
        "lines": [
            ["current_temp_id","current_temperature"]
        ]
    }
}

class Service(SimpleService):
    def __init__(self, configuration=None, name=None):
        SimpleService.__init__(self, configuration=configuration, name=name)
        self.order = ORDER
        self.definitions = CHARTS
        #values to show at graphs
        self.values=dict()


    @staticmethod
    def check():
        return True

    weather_data=dict()
    weather_metrics=[
                "temp","av_temp","min_temp","max_temp",
                "humid","av_humid","min_humid","max_humid",
```

```
            "pressure","av_pressure","min_pressure","max_pressure",
            ]

    def logMe(self,msg):
        self.debug(msg)


    def populate_data(self):
        for metric in self.weather_metrics:
            self.weather_data[metric]=random.randint(0,100)


    def get_data(self):
        #The data dict is basically all the values to be represented
        # The entries are in the format: { "dimension": value}
        #And each "dimension" shoudl belong to a chart.
        data = dict()

        self.populate_data()


        data['current_temp_id'] = self.weather_data["temp"]

        return data
```