IOT Data Generator

Introduction

This tool allows the configuration and generation of synthetic IOT sensor data. The user can configure IOT Device Types, specify number of copies of each device type and configure each device with multiple sensors.

Design Approach

The problem of synthetic data generation for IOT devices could be broken up into three parts.

a. ***Creating the Device definitions and message format definitions***
A schema definition for a Device and its sensors could exist in an xml schema definition file conforming to ODF standards. However, currently there are not too many standard definitions available. Also, when new devices and domains emerge in the future, or if standards develop for current domains, the xsd definitions may change.

This tool takes in the specification in a json format and hence is not tied into the xsd definition. JSON was chosen as it is self-describing and also easy to understand because it is textual and not binary.

b. ***Generating data***
The tool outputs the generated data in the form of records in json lines format - this is one json record per line. As millions of records could potentially be generated in an IOT data generator, all of these in one giant json array in the file will not scale. If there are a 100 million records in a json array, it would require a great deal of memory to load it using a single API - and may not succeed. Otherwise, we would have to work out a hack to load the file part by part to satisfy the requirement. **JSON streaming is a good solution to this problem.** By using json lines format, one could potentially stream out the individual data records - a json record could easily be transformed into the required ODF xml format. This is demonstrated in a demo program supplied with the toolset.

Reference: This wikipedia entry explains the json lines format and json streaming.
https://en.wikipedia.org/wiki/JSON_streaming

c. ***Creating the xml file from the python objects***
The json file could be transformed into an ODF xml file as required based on the mapping between the json record and the xml record.
A demo program is provided that takes the iot-1-demo json and translates it into the demo.xml file.

Model

A Device Definition file could consist of multiple devices. A Device consists of multiple sensors.

Multiple copies of a device could be present in a Device Definition file.
You could create multiple device definition files.
The exact syntax is specified in the Usage section.

Functions provided by the tool

DatagenIOT is a synthetic data generator for IOT devices.

It can be used for:

- Generating test data for complex IOT devices with multiple configurable sensors
- The data generated could be stored in a database or directly used by an IOT simulator to send data on the network.
- As the tool supports multiple configurations for a device and the message format could also be specified, it could easily be translated into any ODF format as and when it is defined.

USAGE

A Device Definition file could consist of multiple devices.
A Device" type" could be "simple" or "complex".
A "complex" type should have a "message_template.json" where the message format to output is specified. This could have nested structures and is akin to the complex type of xml schema.
A device could have two types of "sampling". - "random" and "fixed"
In "random" sampling, a random number is chosen in the "interval" and it is added to the time at which the previous message was generated for this device.
All "interval" units are in seconds.
"num_copies" specifies the number of copies of this device present.
Two Device identifier types are supported.

1. dev.uuid - A fully numeric string with formats like {:02d}
2. dev.uuid2 - An alphanumeric string with formats like dev-{:06d}-{:1d}

All uuids for the devices are automatically generated based on the uuid type.
Ex*: If there are 5000 devices then, the uuid2 numbers will be "dev-000000-0" to "dev-000499-9"*

Constraints and restrictions on the types produced by the sensors are specified in the device definition file. This consists of
{datatype, variable name, optional attributes based on the data type}.
We will discuss each of the types here.

The basic types supported are:

1. IntRange - This will generate a random integer in the range specified by the attributes "min" and "max".
2. DecRange - This will generate a random decimal number with "dec" precision and in the range specified by "min" and "max"
3. selFromList - selects a random item from the list provided in the "random" attribute. The list could be strings, integers or decimals.
4. timestamp - outputs a string version of the datetime format. All interval units are seconds.

The message format for a "simple" device type is just the json formatted structure of the variables named in the "sensors" attribute.

Ex:
{"uuid": "02", "ts": "2015-01-01 14:01:10", "levelstr": "g", "temp": -3.8, "level": 9.4}

The message format for a “complex” Device is discussed in detail below.

====
Device Definition file

```
[
   {
      "type":"complex ",      #
      "uuid":"dev-{:06d}-{:1d}",
      "sampling":{"type":"random", "interval":5},
      "num_copies":2,
      "message_template":"template.json",
      "sensors":[
         {"type":"dev.uuid2",   "name":"uuid"},
         {"type":"IntRange", "name":"light_value",     "min":0,     "max":300000},
         {"type":"IntRange", "name":"temp_value",      "min":20000, "max":35000},
         {"type":"IntRange", "name":"pressure_value",  "min":30000, "max":110000},
         {"type":"IntRange", "name":"humidity_value",  "min":10,    "max":90}
      ]
   },
   {
      "type":"simple",
      "uuid":"{:02d}",
      "sampling":{"type":"fixed", "interval": 10},
      "num_copies":3,
      "sensors":[
         {"type":"timestamp",    "name":"ts"},
         {"type":"dev.uuid",         "name":"uuid"},
         {"type":"DecRange",   "name":"temp", "dec": 1, "min":-15.0, "max":3.0},
         {"type":"selFromList",  "name":"level", "random": [1.1,3.2,8.3,9.4]},
         {"type":"selFromList",        "name":"levelstr", "random":
["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o"]}
      ]
   }
]
```

====

Message template for complex device type

Ex: template.json

====

```
{
      "sn": "$uuid",
      "sensors": {
            "light": {
                  "value": "<$light_value>",
                  "unit": "mLux"
            },
            "temp": {
                  "value": "<$temp_value>",
```

```
                "unit": "mCelsius"
            },
            "pressure": {
                "value": "<$pressure_value>",
                "unit": "Pascal"
            },
            "humidity": {
                "value": "<$humidity_value>",
                "unit": "units"
            }
        }
}
====
```

The message template allows you to create models for different sensors as shown above. Here, four sensors - light, temperature, pressure and humidity are modelled. THe variable names here should correspond with those mentioned in the Device definition file. If a variable is a number, it should be enclosed in <> brackets.

Example message format for the above template:

```
====
{"sn": "dev-000000-1", "sensors": {"humidity": {"unit": "units", "value": 87}, "light": {"unit": "mLux", "value": 226556}, "temp": {"unit": "mCelsius", "value": 28046}, "pressure": {"unit": "Pascal", "value": 53393}}}
====
```

Steps to run the program.

This program is written in python3. It uses the json module.

Step 1.
Create the device definition file.
Ex: config/Device-simple.json
This consists of a simple device and a complex device.

Step 2.
Create the message template files if required.
Ex: config/template.json

Step 3.
Create the configuration file.
Ex: iot-1.properties

RUN

```
-python DataGenIOT.py iot-1.properties
====
Using configuration file iot-1.properties
Generated 2 sets of records to output/iot-1-out.jsonl
```

====

This creates the output file in the output directory specified in the properties file.
Ex: output/iot-1-out.jsonl

To Verify the file contents
To count the number of records and verify that they are valid json records.
-python testDataGen.py output\iot-1-out.jsonl
====
Number of records in the file output\iot-1-out.jsonl is 10
====

Run the demo to convert the json to ODF xml
-python convXml.py
This takes the iot-1-demo.json and converts it into a demo.xml file.
====
generated xml file is output/demo.xml
====

Example of transforming a json lines into a ODF xml.

A cryogenic refrigerator in the starship Enterprise (to be invented in the future) may not have a ODF definition yet. However, when the definition is available, the json lines could easily be transformed into the xml as shown in the example.

The JSON lines 1 and 2 -
====

```
{"object": {
            "temp": {
               "unit": "mCelsius",
                     "value": 32874
                     },
            "humidity": {
                     "unit": "numbers",
                     "value": 23
                     }
             },
             "sn": "dev-000021-1",
             "name":"Refrigerator Assembly Product"
}
{"object": {
     "temp": {
                "unit": "mCelsius",
                     "value": 32875
                     },
            "humidity": {
                     "unit": "numbers",
                     "value": 24
                     }
```

```
            },
            "sn": "dev-000021-2",
            "name":"Refrigerator Assembly Product"
}
====
```

is transformed into

```
====
{
   "XML": {
      "version": 1.0,
      "encoding": "UTF-8"
   },
   "Comment": " Example of a simple odf structure for a refrigerator. ",
   "Objects": {
      "xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
      "xsi:noNamespaceSchemaLocation": "odf.xsd",
      "Object": {
         "type": "Refrigerator Assembly Product",
         "id": "dev-000021-1",
         "InfoItem": {
            "name": "temperature",
            "description": "Temperature",
            "value": [
                                                {
                                                        "unit": "mCelsius",
                                                        "value": 32874
                                                }
            ]
         },
         "InfoItem": {
            "name": "humidity",
            "description": "Humidity",
            "value": [
                                                {
                                                        "unit": "numbers",
                                                        "value": 23
                                                }
            ]
         }
      },
      "Object": {
         "type": "Refrigerator Assembly Product",
         "id": "dev-000021-2",
         "InfoItem": {
            "name": "temperature",
            "description": "Temperature",
            "value": [
                                                {
                                                        "unit": "mCelsius",
                                                        "value": 32874
```

```
                                        }
            ]
        },
        "InfoItem": {
            "name": "humidity",
            "description": "Humidity",
            "value": [
                                        {
                                            "unit": "numbers",
                                            "value": 23
                                        }
            ]
        }
    }
  }
}
====
```