TSimulus Documentation

Release 0.1.13

Mathieu Goeminne

Oct 06, 2017

Overview

CHAPTER 1

About

TSimulus is a toolkit for generating random, yet realistic, time series. In this project, a time series is nothing but a orderly sequence of points in times, each of them being associated to at most a value. Time series are used in a wide variety of areas, including finance, weather forecasting, and signal processing.

While random-number generators can easily be used for producing sequences of unrelated (or, at least, hardly predictable) numbers, generating sequences of numbers that seem to respect some *obvious* patterns is also interesting in many circumstances, including the simulation of data acquisition in the aforementioned areas.

In order to make realistic time series, a *convincing* noise must generally be added to some specified patterns. In addition, the values of a time series may be related to those of an other time series.

The TSimulus project provides tools for specifying the shape of a time series (general patterns, cycles, importance of the added noise, etc.) and for converting this specification into time series values.

This library is part of the EAM-SDI research project, founded by the Walloon Region.

CHAPTER 2

Features

Generators: The TSimulus library provides a domain specific language (DSL) that can be used for specifying **generators** that describe the shape of the desired time series (evolutionary patterns, cycles, noise, etc.). Alternatively, these generators can be programmatically specified using a Java/Scala API. The generators can be converted into time series values for a considered time period.

Generator Combination: Generators can be **combined** in order to produce higher-level generators. Such generators can describe conditional and time-based time series, such that "if the value of this time series is higher than the value of that time series, then this value must be generated. Otherwise, that value must be generated instead."

Numeric and Binary Values: Generated time series values may be either numeric or binary. Bool operations can be applied on binary values, that can be used for describing conditional time series. Numeric values can be combined and compared in different ways, in order to create complex time series by combining simple ones.

Time Series Evaluation: Time series can be evaluated for any point of time. This evaluation is fast, side effect-free, and referentially transparent (in particular, the evaluation of a time series always provides the same value for a given timestamp).

Furthermore, the library supports the generation of time series values as a (potentially illimited) number stream. With such a structure,

Missing Values: A time series may not have a value to provide for any possible timestamp. Such cases are managed by the library as "missing" values. Missing values may be conditionally replaced by "default" values and can be discarded from a collection of values before operating an aggregation.

CHAPTER 3

Licence and Source Code

TSimulus is released under the Apache Licence (version 2.0) and has been initiated as part of EAM-SDI, a CWALity research project of the Walloon region.

The source code is available on Github: https://github.com/cetic/tsimulus

Please feel free to contribute by submitting pull requests, bug reports and feature requests.

Installation

The simplest way to install the TSimulus library is to add it as a dependency in your SBT file:

```
libraryDependencies += "be.cetic" %% "rts-gen" % "0.1.13"
```

Alternatively, you can use add the following in your pom document:

```
<dependency>
    <groupId>be.cetic</groupId>
    <artifactId>rts-gen_2.11</artifactId>
    <version>0.1.13</version>
</dependency>
```

You can also clone its source code repository from Github and package it using SBT:

```
> git clone https://github.com/cetic/tsimulus.git
> cd tsimulus
> sbt package
```

If you are not interested in the development of this library, you more likely want to know *How to Use the Library*.

Getting Started

A First Generation

The simplest way to test the TSimulus project is to download the last release of a runnable Jar file that contains an application based on the TSimulus library. You will also need a Java Virtual Machine installed on your environment.

For your first time series generation, we will specify a generator that propose a basic external temperature simulator for Uccle, Belgium. According to the Royal Meteorology Institute of Belgium, the mean temperatures are as follows:

Month	Temperature
January	3.3
February	3.7
March	6.8
April	9.8
May	13.6
June	16.2
July	18.4
Augustus	18.0
September	14.9
October	11.1
November	6.8
December	3.9

So, we create a generator that describes the evolution of the temperature month per month:

```
{
   "generators":[
   {
      "name": "monthly-basis",
      "type": "monthly",
      "points": {"january": 3.3, "february": 3.7, "march": 6.8, "april": 9.8, "may":␣
↪13.6, "june": 16.2,
                 "july": 18.4, "august": 18, "september": 14.9, "october": 11.1,
↪"november": 6.8, "december": 3.9}
   }],
   "exported":[
      {"name": "temperature", "generator": "monthly-basis", "frequency": 3600000}
   ],
   "from": "2016-01-01 00:00:00.000",
   "to": "2017-12-31 23:59:59.999"
}
```

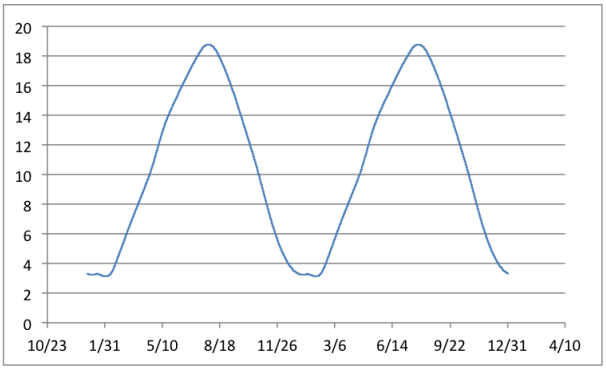
(Download this configuration)

The name and the attributes of each objects in this JSON document are described further in the rest of this documentation. At the moment, we will only focus on the created configuration and the resulting values.

Save the configuration on a text file next to the downloaded application and run the application with the freshly created file as well as the limits of the time period of interest:

```
java -jar rst-gen-cli get_started_1.json
```

After a few seconds, you obtain a sequence of lines, each of them being a value entry made of a date, a series name, and a value separated by semicolons. In this demonstration, the series name is always “temperature”. If you plot the series values with your favourite tool, you should obtain something like the following plot:



Towards a more Realistic Model

So it works! The temperature continuously varies, and for the middle of each month it complies with the values specified in the configuration. But you may be frustated by the regularity of the obtained values: they increase or decrease monotoneously from a month to the next one, which is not a very realistic behaviour. Actually, for consecutive days, temperatures are globally higher during the day, and lower during the night.

We therefore create a new generator that expresses the variation of the temperature over the hours of a calendar day:

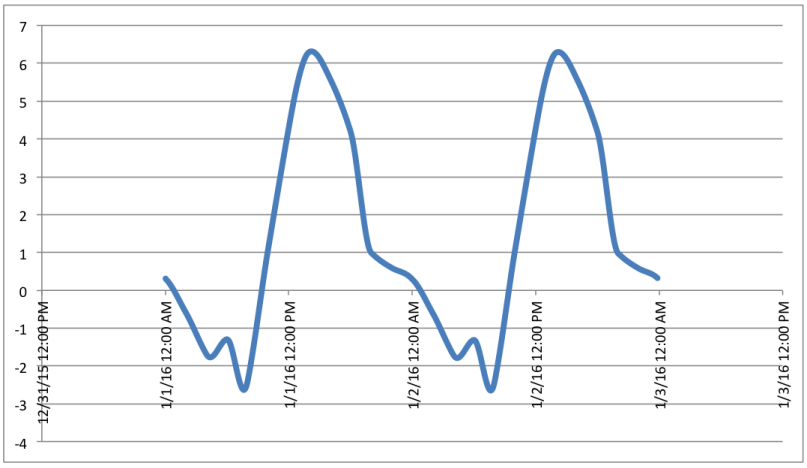
Hour	Relative Temperature
00:00	-3.0
02:00	-3.9
04:00	-5.0
06:00	-4.6
08:00	-5.7
10:00	-2.2
12:00	+1.0
14:00	+3.0
16:00	+2.3
18:00	+0.9
20:00	-2.3
22:00	-2.7

```
{
   "generators": [
       {
          "name": "monthly-basis",
          "type": "monthly",
          "points": {
             "january": 3.3,
             "february": 3.7,
             "march": 6.8,
             "april": 9.8,
             "may": 13.6,
             "june": 16.2,
             "july": 18.4,
             "august": 18,
             "september": 14.9,
             "october": 11.1,
             "november": 6.8,
             "december": 3.9
          }
       },
```

```
        {
            "name": "daily-variation",
            "type": "daily",
            "points": {
                "00:00:00.000": -3,
                "02:00:00.000": -3.9,
                "04:00:00.000": -5,
                "06:00:00.000": -4.6,
                "08:00:00.000": -5.7,
                "10:00:00.000": -2.2,
                "12:00:00.000": 1,
                "14:00:00.000": 3,
                "16:00:00.000": 2.3,
                "18:00:00.000": 0.9,
                "20:00:00.000": -2.3,
                "22:00:00.000": -2.7
            }
        },
        {
            "name": "result",
            "type": "aggregate",
            "aggregator": "sum",
            "generators": [
                "monthly-basis",
                "daily-variation"
            ]
        }
    ],
    "exported": [
        {
            "name": "temperature",
            "generator": "result",
            "frequency": 600000
        }
    ],
    "from": "2016-01-01 00:00:00.000",
    "to": "2017-12-31 23:59:59.999"
}
```

(Download this configuration)

Please note that the values of this second generator are relative to an arbitrary "neutral" value. Plotting these values is therefore not relevant. However, we sum the monthly temperatures with the daily ones in order to obtain a more complex behaviour, and the more realistic resulting time series can be displayed:



In this image, only values from the first and the second of January 2016 are displayed. A daily basis pattern is easily observable, while values are quite similar (although slightly different) from day to day.

However, a new further examination of the generated values reveals that the temperature variation remains unsatisfactory: during a calendar day, the temperatures varies unrealistically, and two identical days in different years (for instance, 2016-02-03 and 2017-02-03) have the same sequence of values. In the real life, the temperature slightly changes over time due to complex modifications of the atmospheric conditions.

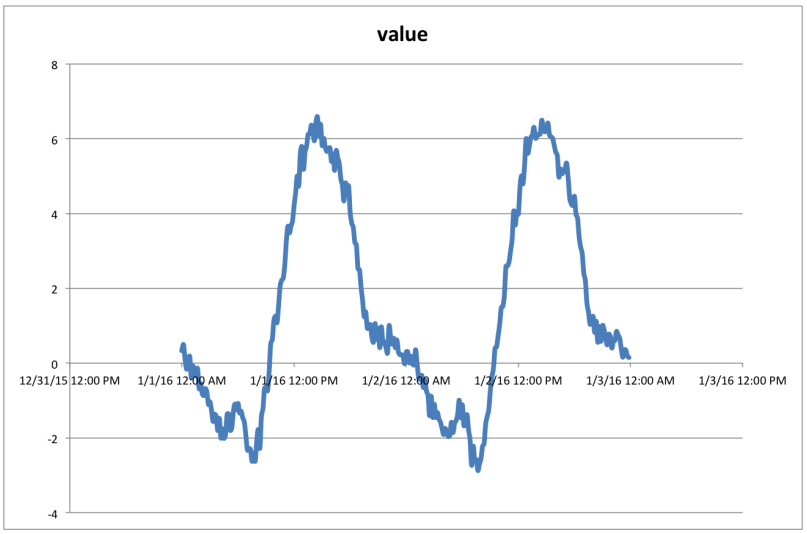
In order to simulate these small changes, we introduce a generator that describe a noisy time series, and we sum it with the previously defined generators.

```
{
   "generators": [
      {
         "name": "monthly-basis",
         "type": "monthly",
         "points": {
            "january": 3.3,
            "february": 3.7,
            "march": 6.8,
            "april": 9.8,
            "may": 13.6,
            "june": 16.2,
            "july": 18.4,
            "august": 18,
            "september": 14.9,
            "october": 11.1,
            "november": 6.8,
            "december": 3.9
         }
      },
      {
         "name": "daily-variation",
         "type": "daily",
         "points": {
            "00:00:00.000": -3,
            "02:00:00.000": -3.9,
            "04:00:00.000": -5,
            "06:00:00.000": -4.6,
            "08:00:00.000": -5.7,
            "10:00:00.000": -2.2,
            "12:00:00.000": 1,
            "14:00:00.000": 3,
```

```
            "16:00:00.000": 2.3,
            "18:00:00.000": 0.9,
            "20:00:00.000": -2.3,
            "22:00:00.000": -2.7
         }
      },
      {
         "name": "noise",
         "type": "arma",
         "model": {
            "std": 0.2,
            "c": 0,
            "seed": 1234
         },
         "timestep": 300000,
         "origin": "2016-01-01 00:00:00.000"
      },
      {
         "name": "result",
         "type": "aggregate",
         "aggregator": "sum",
         "generators": [
            "monthly-basis",
            "daily-variation",
            "noise"
         ]
      }
   ],
   "exported": [
      {
         "name": "temperature",
         "generator": "result",
         "frequency": 600000
      }
   ],
   "from": "2016-01-01 00:00:00.000",
   "to": "2017-12-31 23:59:59.999"
}
```

(Download this configuration)

The final result is now realistic enough for a basic simulation of the temperature over time. When observing the plot of its values, clear and realistic patterns emerge, while a realistic noise is also clearly present.



Not satisfied by the realism of the generated values? Don't hesitate to modify the parameters of the generators described in the configuration file or to try other *About Generators*.

Generators

About Generators

The main goal of generators is to describe the constraints that shape the generated time series.

Several kinds of generators are detailed in this section. Primary generators describe time-based or even simpler constraints and don't rely on any other generator. You can consider them as the building blocks of your time series.

Generators may be combined in various ways in order to produce more complex generators. Such composite generators express time series whose values are the result of the aggregation, the comparision, the correlation, etc. of the values generated by time series described by other generators.

While most generators describe time series producing numerical values, others can describe binary time series, i.e. time series that only generated true or false values. Such values are typically used in a condition, and can result from a test on one or many time series value(s). Composite binary generators support the basic and secondary boolean operations.

Configuration Document

The shape of the generated time series are defined using **generators**. These generators can be specified in a declarative fashion in a so-called **configuration document**, which is a JSON document respecting a given structure. This document is made of a *generators* section, in which generators are declared, a *exported* section containing the list of generators that must be converted into time series, as well as a pair of date delimiting the time period for which time series values must be generated.

generators

The use of the *generators* section in a configuration document, although optional, is highly recommended in order to describe the specify the generators describing the time series to generate. The *generators* section is therefore essentially a list of generators that could be converted into time series. In this section, **composite generators** (i.e: generators based on other generators) can be specified in two different ways:

- By writing nested objects in the document. When the specification of a generator requires a generator attribute, a JSON object describing this attribute can simply be used. There is no limits to such nested, **inline description**, in such a way specified generators can be as complex as desired.
- By using a reference to a top-level generator. When some generator attributes must be used in several places, or when the generator designer prefer to explicitly declare intermediate generators in order to obtain configuration documents that can be more easily be read, intermediate generators can be declared in the *generators* section with a unique reference id. Everywhere in the document, when a generator is required, the unique id can be used instead of an explicit JSON object representing the generator. This description by reference only works if referenced generators are defined at the top level of the generators section. In other words, **generators with online description in a nested object cannot be used by reference**.

exported

This section of the configuration document lists the generators that must be converted into time series. Again, this is essentially a list of objects containing the following attributes:

- **name**: the name that must be associated to the time series.
- **generator**: a description of the generator representing the time series to generate. This may be an online description.
- **frequency**: the period, in milliseconds, at which time series values must be generated.

'from' and 'to'

While the entire library can generate values for any valid moment, and despite the fact that time series are internally considered as potentially unlimited streams of values, such a boundless generation cannot be processed in a limited time. Consequently, two extra fields, expressing the beginning and the end of the time period for which values must be generated, are required in the configuration document in order to be able to generate time series.

However this specificity can be expected to change in the future.

Primary Generators

Primary time series generate values having no relation with any other time series.

A user can constrain the general shape of a primary time series by specifying some data points (which correspond to points in times associated to specific values), and by specifying that the generated time series must contain these data points.

The values for the points in time that don't correspond to a declared data point are interpolated by using a cubic spline. The approach followed for computing this interpolation implies that the user must specify at least three points for constraining the time series.

Daily Generator

A daily time series is based on a daily cycle. The user specifies the values the time series must generated for some points in time in the day, and these points are replicated in any day for which the user could desire a value.

In the meteorological domain, a typical daily generator would consist in a set of times (for instance, 11AM) associated to a temperature of 6°. The time series generated based on this constraint will be such that every day, the temperature associated to 11AM will be 6°.

Representation in the configuration document:

name The name associated to the generator describing a daily time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'daily'.

points Mandatory. Contains the points in time in the day, as well as the values associated to these instants. At least three points must be specified. If an instant is present several times in the points, only the last declaration of the point associated to this instant will be retained.

Example:

```
{
  "name": "daily-generator",
  "type": "daily",
  "points": {"11:00:00" : 6, "17:00:00" : 8, "07:00:00": 2}
}
```

Weekly Generator

A weekly time series is based on a weekly cycle. The user specifies the values of the time series for some days of the week.

In the meteorological domain, a typical weekly generator would consist in a set of days (for instance, Monday) associated to a temperature of 6°. The corresponding time series will be such that every week, the temperature associated to Monday, 12AM will be 6°.

Representation in the configuration document:

name The name associated to the generator describing a weekly time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'weekly'.

points Mandatory. Contains the days of the week, as well as the values associated to these days. At least three points must be specified. If an day is present several times in the points, only the last declaration of the point associated to this day will be retained. The days must be expressed in English, with lowercase letters.

Example:

```
{
  "name": "weekly-generator",
  "type": "weekly",
  "points": {"monday": 6.0, "friday": -3.6, "sunday" : 10.9}
}
```

Monthly Generator

A Monthly time series is based on a monthly cycle. The user specifies the values of the time series for some months in the year.

In the meteorological domain, a typical monthly generator would consist in a set of months (for instance, January), associated to a temperature of -6.3°. The time series generated based on this constraint will be such that every year, the temperature associated to mid-January will be -6.3°. The selected instant is exactly the mid-term of the month, so that the actual time may depend on the fact that the considered year is a leap year or not.

Representation in the configuration document:

name The name associated to the generator describing a monthly time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'monthly'.

points Mandatory. Contains the months of the year, as well as the values associated to these months. At least three points must be specified. If a month is present several times in the points, only the last declaration of the point associated to this month will be retained. The months must be expressed in English, with lowercase letters.

Example:

```
{
  "name": "monthly-generator",
  "type": "monthly",
  "points":  {"january": -6.3, "february": -6.9, "june" : -2.7, "october" : -3.4}
}
```

Yearly Generator

A yearly time series is based on a yearly cycle. The user specifies the values of the time series for some.

Contrary to daily, weekly, and monthly time series, a yearly time series cannot present any natural cycle, since the number of the year is on an open scale. Consequently, a cycle is artificially created by considering that the value of the year following the last mentioned one, will be equal to the value associated to the first mentioned year.

Similarly, the value of the year preceding the first mentioned year will be equal to the value associated to the last mentioned year.

Representation in the configuration document:

name The name associated to the generator describing a yearly time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'yearly'.

points Mandatory. Contains some years, as well as the values associated to these years. At least three points must be specified. If a year is present several times in the points, only the last declaration of the point associated to this year will be retained.

Example:

```
{
    "name": "yearly-generator",
    "type": "yearly",
    "points": {"2015": 42.12, "2016": 13.37, "2017": 6.022}
}
```

Constant Generator

A time series generator whose values are constant over time.

Representation in the configuration document:

name The name associated to the generator describing a constant time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'constant'.

value Mandatory. Specifies the value of the time series.

Example:

```
{
  "name": "cst",
  "type": "constant",
  "value":  17.5
}
```

Stochastic Generator

In order to improve the realism of time series, a generator can be used to describe a random walk, based on an ARMA model. This model is made of four modules:

- The 'auto-regression' module, represented by ϕ a parameter vector which represents the auto-correlation of the model.
- The 'mobile average' module, represented by θ a parameter vector which represents the evolution of the average over time.
- The 'noise' module, represented by σ the standard deviation and μ the expectation of a normal distribution. This distribution is used for generating a white noise : from this distribution is generated an independent variable which is added to the model.
- A constant c.

From these four modules, a series of values is generated according to the following formula :

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}.$$

(image from Wikipedia)

Where ϵ represents the noise module.

In practice, only a few parameters are enough for obtaining a convincing noise : the standard deviation and a 1D autoregression vector provide, in most cases, a satisfying pseudo-random time series.

The random walk proposed by an ARMA model being discrete by nature, the user has to precise the time interval separating two successive steps. The value of the time series between two successive steps is computed by a linearly interpolation of these steps.

Representation in the configuration document:

name The name associated to the generator describing a constant time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'arma'.

model Mandatory. Must contain the parameter of the ARMA model used in this generator.

phi A list representing the parameter vector of the 'autoregression' module.

theta A list representing the parameter vector of the 'mobile average' module.

std The standard deviation of the normal distribution associated to the model.

c The constant value associated to the model.

seed An arbitrary integer value, which will be used in order to ensure the deterministic nature of the generated time series.

timestep Mandatory. The duration, in milliseconds, between two successive steps.

origin Mandatory. The moment at which the time series is starting.

Example:

```
{
  "name": "g3",
  "type": "arma",
  "model": {
    "phi": [1,2,3],
    "theta": [4,3,2,1],
    "std": 0.5,
    "c": 4.2,
    "seed": 1809
  },
  "timestep": 180000,
  "origin": "2016-01-01 12:34:56.789"
}
```

Gaussian Noise Generator

The gaussian noise generator is an alternative to the stochastic generator. It produces, for any time, a random value the distribution of which is close to the gaussian, or normal, distribution.

Contrary to the stochastic generator, this generator produces independent values. The gaussian noise generator is therefore less realistic than the stochastic generator, but in return the generator of gaussian values is in constant time, while the stochastic generator must produce all the intermediate values between the generator origin to the requested time.

Representation in the configuration document:

name The name associated to the generator describing a constant time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'gaussian'.

seed Mandatory. The seed of the random number generator.

std Mandatory. The standard deviation of the Gaussian distribution followed by the generated values.

Example:

```
{
  "name": "generator",
  "type": "gaussian",
  "seed": 42,
  "std": 0.5
}
```

Composite Generators

Time series generators can be composed in order to form composite generators. Two types of composite generators can be distinguished:

- **Transformation generators**, which are based on an other generator and transform the values of its time series.
- **Combined generators**, which are based on multiple generators and combine the values of their time series in order to produce new values.

Two methods are available for mentioning the generators describing a composed time series : **references** and **inline definitions**.

A reference is a text label corresponding to the name of the referred generator. There must be a generator having this name in the configuration document, otherwise an error occurs when converting the generators into time series. A generator does not need to be defined before being referred, but referred generators need to be first-level generators, i.e., they cannot be defined as a part of a composite generator.

The inline definition of a generator consists in writing a JSON object representing this generator.

Affine Function

This is a transformation time series based on the application of a function on the underlying time series. More specifically, at any time, the value of the new time series is the application of a function on the value of the underlying time series for the considered time.

While the generator is design to apply any arbitrary function to time series values, using the configuration document one can specify only affine functions.

This time series can be described as an *aggregation* of several time series (including constant time series), and should therefore be considered as a shortcut for a definition of this specific composition.

Representation in the configuration document:

name The name associated to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'function'.

generator Mandatory. Describes, by reference or inline definition, the generator of the time series on which the new time series is based.

slope Mandatory. The first derivative of the linear function belonging to the affine function. This coefficient is applied to any value provided by the underlying time series.

intercept Mandatory. Specifies the y-intercept of the affine function applied to the values generated by the underlying time series. The relation between the new time series and the underlying one is therefore: $y=slope*x + intercept$, where y represents the values generated by the new time series, and x represents the values of the underlying time series.

Example:

```
{
   "name": "function-generator",
   "type": "function",
   "generator": { "type" : "constant", "value" : 42 },
   "slope": 1.4,
   "intercept" : 9.2
 }
```

Aggregation

A new time series can be obtained by aggregating several time series. More precisely, the values of an aggregate time series are computed by aggregating the values of the values of the underlying time series.

While the generator is design to apply any arbitrary aggregation function to the provided values, only some predefined aggregation functions can be specified in the configuration document.

The following aggregation functions are available:

sum The sum of all the elements.

product The product of all the elements.

min The minimal value of all the elements.

max The maximal value of all the elements.

mean The mean value of all the elements.

Median The median value of all the elements.

Representation in the configuration document:

name The name associated to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'aggregate'.

aggregator Mandatory. Must be either 'sum', 'product', 'min', 'max', 'mean', or 'median'.

generators Mandatory. A list of the generators describing the underlying time series.

Example:

```
{
  "name": "aggregate-generator",
  "type": "aggregate",
  "aggregator": "sum",
  "generators": ["daily-generator", "monthly-generator"]
}
```

Division

A special kind of aggregation consisting in the division of the values of a time series by the values of an other one. Contrary to the other proposed aggregation, a time series division can only be based on exactly two time series.

If the value of the numerator or the denominator is not defined, the result of the division is not defined.

Representation in the configuration document:

name The name associated to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'divide'.

numerator Mandatory. The generator that must be used as numerator.

denominator Mandatory. The generator that must be used as denominator.

Example:

```
{
  "name": "division-generator",
  "type": "divide",
  "numerator": "generator-a",
  "denominator": "generator-b"
}
```

Correlation

When a time series correlated to an other one is desired, but the exact relation between the series is not required, one can use a generator based on a correlation coefficient.

The user must provide a Pearson coefficient (a value between 0 and 1, such that 0 indicates the total lack of correlation and 1 indicates a perfect correlation), plus the generator of the time series that must be correlated.

The correlated time series tries to generate values in such a way the Pearson coefficient between this series and the underlying one is as close as possible to the provided one. However, a strict equality between these two values cannot be guaranteed.

Representation in the configuration document:

name The name associated to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'correlated'.

generator Mandatory. Describes the generator used for representing the underlying time series that must be correlated to the new one.

coef Mandatory. The value of the Pearson coefficient that must be approached.

Example:

```
{
  "name": "corr-generator",
  "type": "correlated",
  "generator": "daily-generator",
  "coef": 0.8
}
```

Time Shift

This transformation time series causes a time shift in an baseline series: a constant time shift is applied to each time point of a time series, so that the baseline time series seems to be 'shifted' on the time axis.

Representation in the configuration document:

name The name given to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'time-shift'.

generator Mandatory. A description of the generator used for describing the underlying time series.

shift the time, in milliseconds, to add to each time point belonging to the original time series in order to evaluated to the time series.

Example:

```
{
  "name": "time-shift-generator",
  "type": "time-shift",
  "generator": "g1",
  "shift": 60000
}
```

Transition

A transition time series may be used for representing a behavior that fluctuates over time. More precisely, the time series produced by this generator mimic a baseline time series until a given time, after which they mimic an other baseline time series.

A transition period can be specified. During this period, the values generated by the transition time series are defined as the interpolation of the values generated by the two baseline time series. In other words, the generated values progressively switch from those generated by the first baseline time series to those generated by the second baseline time series.

Three interpolation functions are provided for describing how the transition values must be computed:

- **linear**: a function that provides a linear transition between the considered time series.
- **sigmoid**: a function that provides a S-shaped, smoothly transition from a time series to the other one. The variation of the interpolation is mainly observable at the middle of the transition, while the interpolation is almost constant at the beginning and the end of the transition.
- **exponential**: a function that gives the second time series an importance that increases as the exponential of the transition progress.

Representation in the configuration document:

name The name given to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'transition'.

first Mandatory. The generator describing the baseline time series that is mimicked before the transition.

second Mandatory. The generator describing the baseline time series that is mimicked after the transition.

time Mandatory. The moment at which the transition starts.

duration Optional. The transition duration, in milliseconds. If not specified, the transition is immediate.

transition Optional. The interpolation function to use for the transition. If defined, must be either 'linear', 'sigmoid', or 'exponential'. If 'duration' is not specified, this parameter has no effects. Default is 'linear'.

Example:

```
{
  "name": "transition-generator",
  "type": "transition",
  "first": "daily-generator",
  "second": "noise-generator",
  "time": "2016-04-03 12 :34 :56 .000",
  "duration": 300000,
  "transition" : "linear"
}
```

Sliding Window

Time series values can be aggregated by using a sliding window. At any time, the value of a sliding window time series is defined as the aggregation of the *recent* values of a baseline time series. A typical use case of this time series is a mobile average.

If no values are defined by the underlying time series for the considered time window, the produced value is undefined.

While the generator is designed to process any arbitrary aggregation function to the elements belonging to the time window, only some predefined aggregation functions can be specified in a configuration document.

The following aggregation functions are available :

- **sum**: the sum of all the elements.
- **product**: the product of all the elements.
- **min**: the minimal value of all the elements.
- **max**: the maximal value of all the elements.
- **mean**: the mean value of all the elements.
- **median**: the median value of all the elements.

Representation in the configuration document:

name The name associated to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'window'.

aggregator Mandatory. Must be either 'sum', 'product', 'min', 'max', 'mean', or 'median'.

generator Mandatory. The generator describing the underlying time series.

window-length The length, in milliseconds, of the considered time window.

n The number of evenly spaced moments in the time window to take into account.

Example:

```
{
  "name": "window-generator",
  "type": "window",
  "aggregator": "sum",
  "window-length" : 5000,
  "generator": "daily-generator",
  "n": 5
}
```

Binary Generators

In addition to continuous time series (i.e., time series producing continuous values), binary time series can be generated by transforming continuous values to binary values.

Probabilistic binarization

A logistic distribution model may be used for generating a binary time series. This model associates to each value of a continuous time series a probability that the produced binary value is true.

The logistic distribution model is characterized by two parameters:

- **location**: represents the position of the model on the continuous variable axis. Its value corresponds to the median of the continuous values. In other words, it corresponds to the value associated to a probability of 50% that the binary variable is true.
- **scale**: A strictly positive value representing the scale of the model, ie the « speed » at which the binary values switch from the state 'false' to the state 'true' when the associated continuous values increase. Common values for this parameter are between 1 and 5.

Representation in the configuration document:

name The name given to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be “threshold”.

generator Mandatory. The generator describing the underlying continuous time series.

location Mandatory. Represents the model location.

scale Mandatory. Represents the model scale.

seed Optional. An arbitrary integer used in order to get a deterministic time series.

Example:

```
{
  "name": "logistic-generator",
  "type": "logistic",
  "generator": "g1",
  "location": 6,
  "scale": 2.4,
  "seed": 1809
}
```

True

A True time series is a binary time series that always produces the value ‘true’.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be “true”.

Example:

```
{
  "name": "true-generator",
  "type" : "true"
}
```

False

A False time series is a binary time series that always produces the value ‘false’.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be “false”.

Example:

```
{
  "name": "false-generator",
  "type" : "false"
}
```

And

Two binary time series can be combined by a logical AND. The values of the resulting time series will be true if, and only if, the values of both baseline time series are true. If the value of at least one of the baseline time series is not defined, then the value of the resulting time series is not defined.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'and'.

a Mandatory. A description of one of the binary generators used for generating the binary time series.

b Mandatory. A description of the other binary generators used for generating the binary time series.

Example:

```
{
  "name": "and-generator",
  "type": "and",
  "a": "binary-generator-A",
  "b": "binary-generator-B"
}
```

Or

Two binary time series can be combined by a logical OR. The values of the resulting time series will be true if the value of at least one of the baseline time series is true.

If the value of at least one of the baseline time series is not defined, then the value of the resulting time series is not defined.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'or'.

a Mandatory. A description of one of the binary generators used for generating the binary time series.

b Mandatory. A description of the other binary generators used for generating the binary time series.

Example:

```
{
  "name": "or-generator",
  "type": "or",
  "a": "binary-generator-A",
  "b": "binary-generator-B"
}
```

XOR

Two binary time series can be combined by a logical XOR. The values of the resulting time series will be true if, and only if, the values of the baseline time series are different.

If the value of at least one of the baseline time series is not defined, then the value of the resulting time series is not defined.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be "xor".

a Mandatory. A description of one of the binary generators used for generating the binary time series.

b Mandatory. A description of the other binary generators used for generating the binary time series.

Example:

```
{
  "name": "xor-generator",
  "type": "xor",
  "a": "binary-generator-A",
  "b": "binary-generator-B"
}
```

Implies

Two binary time series can be combined by a logical "implies" (=>). The values of the resulting time series will be true if, the values of the first time series *implies* the values of the second time series. If the value of at least one of the baseline time series is not defined, then the value of the resulting time series is not defined.

A => B is equivalent to not(A) OR B

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'implies'.

a Mandatory. A description of one of the binary generators used for generating the binary time series.

b Mandatory. A description of the other binary generators used for generating the binary time series.

Example:

```
{
  "name": "implies-generator",
  "type": "implies",
  "a": "binary-generator-A",
  "b": "binary-generator-B"
}
```

Equiv

Two binary time series can be combined by a logical equivalence. The values of the resulting time series will be true if, and only if, the values of both baseline time series are defined and identical. If the value of at least one of the baseline time series is not defined, then the value of the resulting time series is not defined.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'equiv'.

a Mandatory. A description of one of the binary generators used for generating the binary time series.

b Mandatory. A description of the other binary generators used for generating the binary time series.

Example:

```
{
  "name": "equiv-generator",
  "type": "equiv",
  "a": "binary-generator-A",
  "b": "binary-generator-B"
}
```

Not

A binary time series can be defined as the negation of an other binary time series. The values of the resulting time series will be true if the values of the baseline time series are false, and vice versa.

If the value of the baseline time series is not defined, then the value of the resulting time series is not defined.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be "not".

generator Mandatory. A description of the binary generator used for generating the new binary time series.

Example:

```
{
  "name": "not-generator",
  "type": "not",
  "generator": "binary-generator"
}
```

Conditional time series

The values produced by a time series can be influenced by the values of a binary time series. The conditional time series takes a binary time series, called the condition, as a parameter, as well as one or two arbitrary time series.

The value produced by the conditional time series is determined by the following rules:

- If the condition value is not defined, the value of the conditional time series is also not defined.
- If the condition value is true, the value of the conditional time series is the value of the first arbitrary time series.
- If the condition value is false, the value of the conditional time series is the value of the second arbitrary time series.

Representation in the configuration document:

name The name given to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be "conditional".

condition Mandatory. The generator describing the condition time series.

success Mandatory. The generator describing a time series, the values of which are used if the condition is verified.

failure Optional. The generator describing a time series, the values of which values are used if the condition is not verified. If not specified, a undefined generator is used instead.

Example:

```
{
  "name": "conditional-generator",
  "type": "conditional",
  "condition": { "type" : "true" },
  "success": { "type": "constant", "value": 17 },
  "failure": { "type": "constant", "value": 42 }
}
```

Because the values of the tested time series are always true, all the values generated by this conditional time series will be 17.

Greater than

The values of a binary time series may be defined as the result of the test of an inequality between the values of two numeric time series. The values of the resulting time series will be true if, and only if, the values of the first numeric time series are greater than those of the second numeric time series. If the value of at least one of the numeric time series is not defined, then the value of the resulting time series is not defined.

This time series is based on a more generic one that allows the use of an arbitrary comparator. However, this generic time series can not be expressed by using configuration document, and can therefore be only expressed programmatically.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'greater-than'.

a Mandatory. A description of one of the numeric generators used for generating the binary time series.

b Mandatory. A description of the other numeric generators used for generating the binary time series.

strict Optional. Specifies of the inequality is strict or not. If not specified, the inequality will be considered as non strict.

Example:

```
{
  "name": "gt-generator",
  "type": "greater-than",
  "a": "binary-generator-A",
  "b": "binary-generator-B",
  "strict": true
}
```

Lesser than

This time series is similar to the precious one, except that the generated binary values are true if the values of the first numeric time series are lesser than those of the second numeric time series (and false otherwise). If the value of at least one of the numeric time series is not defined, then the value of the resulting time series is not defined.

Similarly to the "greater-than" time series, this time series is based on a more generic one that allows the use of an arbitrary comparator. However, this generic time series can not be expressed by using a configuration document, and can therefore be only expressed programmatically.

Representation in the configuration document:

name The name given to the generator describing the time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'lesser-than'.

a Mandatory. A description of one of the numeric generators used for generating the binary time series.

b Mandatory. A description of the other numeric generators used for generating the binary time series.

strict Optional. Specifies of the inequality is strict or not. If not specified, the inequality will be considered as non strict.

Example:

```
{
  "name": "lt-generator",
  "type": "lesser-than",
  "a": "binary-generator-A",
  "b": "binary-generator-B",
  "strict": true
}
```

Missing values

A time series essentially produces a sequence of values associated to equality spaced points in time. However, in practice, some values may be missing, because the measured values do not exist before or after a given date. When simulating the measure of a phenomenon, one could also desire to simulate the fact that the simulated sensor misses a measure, so that 'holes' appear in the time series.

The generator presented in this document support missing values by generating time series that can associate undefined value to any submitted timestamp. *Composite* time series take into account the fact that the baseline time series may produce such undefined values in a conservative way: a transformation time series produce an undefined value if its baseline time series produces an undefined value. The transformation applied may transform a defined value into an undefined one.

Aggregation time series discard all the undefined values produced by the baseline time series, so that the aggregation only applies on defined values. If no defined values are produced by the underlying time series, an undefined value is retrieved, regardless the aggregation function used on the time series.

While any time series can produce undefined values, some of them are specifically designed to transform a defined value into an undefined one, and *vice versa*.

Undefined time series

This time series always produces undefined values.

Representation in the configuration document:

name The name given to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be "undefined".

Example:

```
{
  "name": "undefined-generator",
  "type": "undefined"
}
```

Limited Time Series

This time series removes the values generated by a baseline time series and replaces them by missing values.

Representation in the configuration document:

name The name given to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be "limited".

generator Mandatory. The generator describing the baseline time series.

from Optional. Specifies the date before which the time series must not produce any value. By default, there is no limit date and no values are generated before the higher limit.

to Optional. Specifies the date at which the time series must start to generate values again. By default, there is no limit date and no values are generated after the lower limit. If neither the lower nor the higher limit are specified, the time series will generate no values.

Example:

```
{
  "name": "limited-generator",
  "type": "limited",
  "generator": "daily-generator",
  "from": "2016-01-01 00:00:00.000",
  "to": "2016-04-23 00:00:00.000"
}
```

Partial Time Series

Instead of specifying the periods during which a time series must not generate any values, one can specify the periods during which the time series must generate these values.

This transformation time series is the complement of the limited time series.

Representation in the configuration document:

name The name given to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be "partial".

generator Mandatory. The generator describing the baseline time series.

from Optional. If specified, indicates the date from which the values generates by the baseline time series must by generated. By default, there is no lower limit, and values must be generated until the higher limit.

to Optional. If specified, indicates the date to which the values generated by the baseline time series must be generated. By default, there is no higher limit, and values must be generated from the lower limit.

missing-rate Optional. Must be a number between 0 and 1. If specified, indicates the probability that a generated value will be considered as missing, despite the fact that the lower and higher time constraints are verified. A value of 0 means that no value will be missed ; a value of 1 means that all the values will be missed. By default, this parameter is set to 0.

Example:

```
{
  "name": "partial-generator",
  "type": "partial",
  "generator": "daily-generator",
  "from": "2016-01-01 00:00:00.000",
  "to": "2016-04-23 00:00:00.000",
  "missing-rate": 0.001
}
```

Default Time Series

A common behavior time series pattern consists in retrieving the first defined value among those produced by multiple time series. The default time series is based on an ordered list of baseline time series, and produces the first defined value of these time series.

If the values of all time series are undefined, an undefined value is produced.

Representation in the configuration document:

name The name given to the generator describing a time series. This name must be unique among all generators in the configuration document.

type Mandatory. Must be 'first-of'.

generators Mandatory. A list of generators describing the baseline time series.

Example:

```
{
  "name": "default-generator",
  "type": "first-of",
  "generators": ["daily-generator", "random-generator"]
}
```

How to Use the Library

Time Series Generation

Time series can be generated according to the rules described in a configuration document. You may consider to use our CLI application that reads configuration documents from a regular JSON file.

Alternatively, you could consider to use our microservice that listen for HTTP requests. In that case, the configuration document is submitted as a parameter of a POST request.

A First Example

Examples of complete configuration documents, as well as the way to convert them in order to generate time series values, are available in the *Getting Started* section of this documentation.

Use Cases

A sensor for Colectme

Colectme is a set of tools developed in the EAM-SDI project for collecting, storing, and processing log data from data centers.

The TSimulus library has been used in this project as a data source exploited by a gateway in order to simulate log events representing the history of resources consumption.