Datagenerator

version 0.28

by: uwe geercken
web: www.datamelt.com
email: uwe.geercken@datamelt.com

last update: 2011-05-27

Table of Contents

Functional description

General

All programs that are created, need to be tested for completeness, functionality and quality. Most programs process data in one form or the other and so data is needed to check, if the program does what is was designed for.

Preferably there is data available from e.g. a source system that provides “real-life” data. Or there is a test system around, from which data can be taken to test. But sometimes that is either not the case, the amount of data is insufficient or the number of combinations within the data is too small to make up a representable test.

This is the scenario for the datagenerator tool. It generates mass data based on word lists, purely random data, datetime values or based on a regular expression patterns. The data is output to an ASCII file with fields of fixed length or field divided using a separator. From there it can be used for testing of programs or for feeding it into a tool that processes the data. The output format of the data is specified in a simple XML file, defining the content and the length of the fields which make up the rows in the output file.

How it works

Word lists provide a means to group similar items together such as e.g. “colors” or “weekdays” or “top 100 used english words”, just to name some. The tool randomly selects values from such a word list and uses the retrieved value for a specified field in the output row. As theses lists are text files – containing one value per row – they can easily be extended if necessary. You can – as an example - also create similar lists of words but in different languages. The advantage of word lists is, that you get real and sensefull data in your output file.

Another method of generating data, is to do it purely randomly. This is the easiest way of generating data. All that needs to be specified is the length of the data that shall be generated and the program starts to randomly generate text strings. Once the data is generated, you can e.g. mass load it into a database and test a program for performance or you could see if the tool can cope with an input file of 10 million lines. Of course the random data generated is senseless data. It has no meaning in real life.

There is a special type “datetime” that you can use to generate date and/or time values based on a given pattern. The format of the pattern follows the formatting of the Java SimpleDateFormat class.

If we would generate multiple datetime fields and with a different pattern then we would get two independent dates. That is probably not what we want. We want different date fields which are correctly related to each other: So maybe we want a date field showing “2011-01-01” and another field with the weekday name equal to “Saturday” and not a randomly picked weekday. The datagenerator allows you to do so by defining an id and then later referencing this id from other datetime fields.

The last way to generate data, is to use regular expression patterns. If you know regular expressions – which are quite common in many areas and across different operating systems – then you know that you can use them to test if data fits to a specified pattern. E.g. you have an email address and want to know if it is properly formatted. If you only have a few, you can do it in your head, but if you need to do it programmatically – e.g. on a web page - or if you have large amounts of email addresses, then regular expressions are a solution to do so: you have the data and test it against a pattern. The datagenerator program uses regular expression patterns exactly the other way around: you specify a pattern data should fit to and the tool generates masses of data according to it.

Output format

Below is an example of the XML file used to define how the output of the generated data should be formatted:

```
<xml>
       <row type="delimited" seperator=";">
              <field type="category" category="seasons" length="20" />
              <field type="category" category="months" length="20" />
              <field type="category" category="colors" length="15" />
              <field type="random" length="10" />
              <field type="regex" pattern="www.[a-pA-P]{8,16}.com" length="30" />
              <field type="regex" pattern="[abcdeSTU-W]{12}" length="30" />
              <field type="datetime" id=”date1” pattern="yyyy-MM-dd HH:mm:ss" length="19" />
              <field type="datetime" reference=”date1” pattern="w" length="10" />
       </row>
</xml>
```

The example that comes with the program, contains information of how the file should be structured. At the top, specify the [row] tag and its type. Possible values are “fixed” and “delimited”. If you choose “fixed” then all fields and thus all rows will have the same length: if the generated data is shorter than the specified length of the field – for example if a short word is taken from a word list – then the remaining space, up to the length of the field, is filled with spaces. On the other hand, if you specify “delimited” as row type, then you also need to specify the “separator” attribute, which defines the separator to be used between the individual fields.

Inside the [row] tag, you can have as many field tags as you require to make up a full row of output data. Fields are of three different types: for taking data from a word list (type=category), for generating random data (type=random) or for generating data based on a regular expression pattern (type=regex). For details see above.

Independent of the type you define, you always need to specify the “length” attribute of the field, defining what the maximum length of a field should be.

If you choose “category” as the type of a field, then you also need to specify the name of the file to choose in the attribute “category”. So if your category file on the filesystem is named “colors.category”, then you should specify: category=”colors”. The values for the field will thus be randomly taken from this file. Category files always need to have the extension “.category”.

If you choose “random” as field type, then all you need to specify is the attribute “length” and data will randomly be generated up to this length.

Lastly, if you want to get data that is according to a regular expression pattern, then you need to define the “pattern” attribute. It shall contain a valid regular expression – in terms of Java regular expressions. E.g. the regular expression pattern “[a-zA-z0-9]” defines to use any lowercase alphabetic characters, any uppercase alphabetic characters or a number to be used. Another example would be to generate data according to following pattern: “[s-zäüö]{8,16}”. This would generate data according from the range of characters (s to z lowercase, plus the special characters ä, ü or ö). And it would generate strings of data from length 8 up to length 16. So you would get data of e.g. length 8, 12, 13, etc. If you leave away the second value in the round brackets, then you define a fixed length of the data to be generated. So defining the pattern: “[s-zäüö]{8}” will always generate data of length eight (8).

But there are some minor limitations: In regular expressions you can define the characters “.”, “*” or “?” as placeholders for “any character”. The difference is, that they define the occurrence of none, one or many times of a character or group of characters. This is not possible with the datagenerator tool, as the tool would not know, how many characters to generate. Also – at the time being an “or” character (|) can not be used.

Please note the last two lines. It shows how to use datetime fields where one field references the other. This means that the field with the “id” will generate a datetime and other fields will use this datetime as a reference (but maybe with a different pattern).

Program properties file

There are several parameters that can be passed to the datagenerator program. Alternatively, all parameters can be defined in the properties file “datagenerator.properties”. In this case, no parameter shall be passed to the program, as it takes all arguments from the properties file.

Below find the parameters that can be specified in the properties file:

Name	*Description*	*Example*
numberofoutputlines	optional.specifies how many output lines should be generated	numberofoutputlines=100000
outputinterval	optional. specifies how often the number of generated rows will be displayed	outputinterval=1000
categoryfilesfolder	required. path to the folder where the category files are located	categoryfilesfolder=/home/generator/categories
rowlayoutfile	required. name and path of the file, describing the output format	rowlayoutfile=/home/generator/rowlayout.xml
outputfile	optional. name and path of the file which contains the generated data	outputfile=/home/generator/output.txt
verbose	optional. output status during processing	verbose=true
format	optional. specifies the output format:	format=2

	0=regular (mixed) case, 1=lowercase only or 2=uppercase only	
possiblevalues	optional. defines which character set should be used when generating random data	possiblevalues=ABCDEFGHIJKLMNOPQRST 1234567890!$+*%&
maximumyear	Optional. Defines the maximum year used during date generation. Default is 2199	maximumyear=2013
minimumyear	Optional. Defines the minimum year used during date generation. Default is 1970	minimumyear=2006

Program arguments

If you do not want to use a properties file, then you can pass all required arguments directly to the program.

Name	*Description*	*Example*
-n	optional. specifies how many output lines should be generated	-n=52000
-e	optional. specifies how often the number of generated rows will be displayed.	-e=1000
-c	required. path to the folder where the category files are located	-c=/home/generator/categories
-l	required. name and path of the file, describing the output format	-l=/home/generator/rowlayout.xml
-o	optional. name and path of the file which contains the generated data	-o=/home/generator/output.txt
-v	optional. output status during processing	-v
-f	optional. specifies the output format: 0=regular (mixed) case, 1=lowercase only or	-f=2

	2=uppercase only	
-p	optional. defines which character set should be used when generating random data	-p=ABCDEFGHIJKLMNOPQRST1234567890!$+*%&
-m	optional. Defines the maximum year used during date generation. Default is 2199	-m=2016
-i	optional. Defines the miximum year used during date generation. Default is 1970	-i=2013

Running the program

Running the program is straightforward. As discussed above, either specify the required arguments or define the arguments in a properties file. As a prerequisite you need to have Java installed on your system.

When you use a properties file then run it as follows:

```
java -cp .:datagenerator.jar com.datamelt.datagenerator.DataCreator
```

When you specify all arguments on the command line:

```
java -cp .:datagenerator.jar com.datamelt.datagenerator.DataCreator
-c=/home/generator/categories -l=/home/generator/rowlayout.xml -v
```

Make sure that you adjust the above parameters according to your needs. If the *datagenerator.jar* file is located somewhere else on your filesystem, add the appropriate path.

Note*: The GNU interpreter for Java bytecode - gij – does not work properly with random values. You should use the Sun Java implementation instead.*

Sample data

Below find some sample data generated by the program. The rowlayout file, which defines the structure of the output data, is as follows:

```
<xml>
	<row type="delimited" seperator=";">
		<field type="category" category="seasons" length="20" />
		<field type="category" category="weekdays" length="20" />
		<field type="category" category="months" length="20" />
		<field type="category" category="colors" length="15" />
		<field type="random" length="10" />
		<field type="regex" pattern="www.[a-pA-P]{8,16}.com" length="30" />
		<field type="regex" pattern="[abcdeSTU-W]{12}" length="30" />
	</row>
</xml>
```

So the first four fields in the generated data comes from category files. The fifth field generates random data and the last two fields are generated from the given regular expression patterns.

Here is now an example of the generated data according to the rowlayout file described above:

```
summer;friday;may;orange;cmsa743qjb;www.fjmhnhlgme.com;ccuewwsweaba
summer;thursday;march;purple;mxlmqtmlml;www.hjfkdkdjdgnflg.com;ebttsbteauaw
autumn;monday;october;black;5q8lrg6wsy;www.opdnemingog.com;dvtsweaubceb
summer;tuesday;october;orange;pq4abkmc8c;www.fhnjbhchafgbgah.com;edbducbeaeea
summer;thursday;april;red;6brjz6hdin;www.gnbmblclg.com;taucvvtvstac
spring;saturday;october;orange;ceesawtpxu;www.jkhmfmeiofnfbhpg.com;atescddbbduv
summer;saturday;august;black;ylmonmr8nc;www.fjcicmdl.com;tdewwewbstss
autumn;monday;january;pink;qcbfxqzqtb;www.makaingcpkohek.com;cstcabbvutdv
autumn;friday;november;pink;gb2j162vwo;www.dmgkejekco.com;udbteteaudcw
spring;wednesday;october;violet;bbuw9bb4d4;www.cglfjcod.com;wcbtsbttbbte
spring;wednesday;march;yellow;74ck15wy9v;www.jndmbkpldcm.com;wdsadvswwvue
summer;thursday;march;white;dsyfvvse6d;www.bjmifpfooda.com;vvwsvdcetvvw
```
