
datarator Documentation

Release 0.2.2-dev

datarator team

Mar 18, 2017

Contents

1	Installation	3
1.1	Binaries	3
1.2	From source	3
2	Usage	5
2.1	Running	5
2.2	Command line options	5
2.3	Generate	5
2.4	curl way	6
2.5	wget way	6
2.6	httplib way	6
3	JSON API	7
3.1	Payload	7
3.2	Column specific payload:	8
4	Output template	9
4.1	Template: csv	9
4.2	Template: sql	10
4.3	Template: xml	11
5	Column	13
5.1	Column: address.continent	14
5.2	Column: address.country	15
5.3	Column: address.city	15
5.4	Column: address.phone	15
5.5	Column: address.state	16
5.6	Column: address.street	16
5.7	Column: address.zip	16
5.8	Column: color	16
5.9	Column: color.hex	17
5.10	Column: const	17
5.11	Column: copy	18
5.12	Column: credit_card.number	18
5.13	Column: credit_card.type	18
5.14	Column: currency	19
5.15	Column: currency.code	19

5.16	Column: date.day.of_week	19
5.17	Column: date.day.of_week	20
5.18	Column: date.day.of_month	20
5.19	Column: date.month	20
5.20	Column: date.month.name	21
5.21	Column: date.year	21
5.22	Column: date.of_birth	21
5.23	Column: join	22
5.24	Column: name.first	23
5.25	Column: name.first.female	23
5.26	Column: name.first.male	23
5.27	Column: name.full	24
5.28	Column: name.full.female	24
5.29	Column: name.full.male	24
5.30	Column: name.last	25
5.31	Column: name.last.female	25
5.32	Column: name.last.male	25
5.33	Column: regex	26
5.34	Column: row_index	26
6	Change Log	27
6.1	Unreleased	27
6.2	0.2.1 (2017-02-06)	27
6.3	0.2.0 (2016-10-10)	27
6.4	0.1.0 (2016-09-06)	28

Datarator is the stateless data generator with HTTP based JSON API.

Binaries

Download the binary, that targets your platform.

From source

- Make sure to have [go installed](#) (datarator has been tested with version 1.7) and directory with go binaries in your PATH environment variable
- afterwards run the following commands:

```
go get github.com/datarator/datarator
cd $GOPATH/src/github.com/datarator/datarator
go generate
go install
```


First make sure to have datarator installed (see page: *Installation*).

Running

Assuming datarator binary is in your path, simply run:

```
datarator
```

Command line options

Following are available:

```
Application Options:
-c, --chunk=    The count of generated data returned in one chunk (default: 1000)
-e, --embed     Use embedded rather than external static resources
-p, --port=     Port to listen on (default: 9292)
-t, --timeout=  Timeout in [ms] for maximum request processing (default: 3000)

Help Options:
-h, --help     Show this help message
```

Generate

Assuming datarator has been started with default command line options, to generate:

```
Hello world
```

Pick your tool of choice for sending HTTP POST requests:

curl way

```
curl -H 'Accept-Encoding: gzip, deflate' -H 'Content-Type: application/json' -X POST -
↳d '{"template":"csv","count":1,"columns":[{"name":"greeting","type":"const",
↳"payload":{"value":"Hello world!"}}]}' http://127.0.0.1:9292/api/schemas/say_hello
```

wget way

```
wget -qO - --header='Accept-Encoding: gzip, deflate' --header='Content-Type:
↳application/json' --post-data '{"template":"csv","count":1,"columns":[{"name":
↳"greeting","type":"const", "payload":{"value":"Hello world!"}}]}' http://127.0.0.
↳1:9292/api/schemas/say_hello
```

httpie way

```
echo '{"template":"csv","count":1,"columns":[{"name":"greeting","type":"const",
↳"payload":{"value":"Hello world!"}}]}' | http http://127.0.0.1:9292/api/schemas/say_
↳hello Accept-Encoding: gzip, deflate
```

Refer to *JSON API* page for full reference.

```
{
  "template": "<template_name>",
  "emptyValue": "<empty_value>",
  "count": <count>,
  "columns": [ <column> , <column> , ... ],
  "payload": <payload>
}
```

Legend:

- <template_name> - *Output template* name
- <empty_value> - empty value. By default is empty string
- <count> - generated rows count
- <column> - see *Column*
- <payload> - see *Payload*

Payload

Holds *Output template* (if present in root node) or *Column* specific options (if present in column node).

Syntax:

```
{
  "<name>": "<value>",
  "<name>": "<value>",
  ...
}
```

Legend:

- <name> - name of the option

- `<value>` - value of the option

Column specific payload:

All of these are optional:

```
"emptyPercent": <empty_percent>,  
"xml": <attribute|element|cdata|comment|value>
```

Legend:

- `<empty_percent>` - indicates how much percent of the column values are to be empty. Valid values are: `<0-100>`. Default value is 0.
- `xml` - considered for the *Template: xml* only. Indicating the type of xml node to generate the column value to. (i.e.: `xml: comment` generates value wrapped in xml comment). Default value is `element`.

Following output templates are available:

- *Template: csv*
- *Template: sql*
- *Template: xml*

Template: csv

Enabled via: “*template*”: “*csv*”.

Optional *Payload* available:

- “*header*”: “*true*” / “*header*”: “*false*” - whether names of the columns should included (as the 1.st row) or not. By default is false.
- “*separator*”: “<separator>” - the separator string to be used for joining values.

For **example**, input JSON:

```
{
  "template": "csv",
  "count": 3,
  "columns": [{
    "name": "name1",
    "type": "const",
    "payload": {
      "value": "value1"
    }
  }, {
    "name": "name2",
    "type": "const",
    "payload": {
      "value": "value2"
    }
  }
}
```

```

    }, {
      "name": "name3",
      "type": "const",
      "payload": {
        "value": "value3"
      }
    }
  ]],
  "payload": {
    "header": true,
    "separator": ", "
  }
}

```

results in:

```

name1,name2,name3
value1,value2,value3
value1,value2,value3
value1,value2,value3

```

Template: sql

Enabled via: "template": "sql".

For **example**, input JSON:

```

{
  "template": "sql",
  "count": 3,
  "columns": [{
    "name": "name1",
    "type": "const",
    "payload": {
      "value": "value1"
    }
  }, {
    "name": "name2",
    "type": "const",
    "payload": {
      "value": "value2"
    }
  }, {
    "name": "name3",
    "type": "const",
    "payload": {
      "value": "value3"
    }
  }
]}

```

results in:

```

INSERT INTO foo (name1,name2,name3) VALUES ('value1','value2','value3');
INSERT INTO foo (name1,name2,name3) VALUES ('value1','value2','value3');
INSERT INTO foo (name1,name2,name3) VALUES ('value1','value2','value3');

```

Template: xml

Enabled via: "template": "xml".

Optional *Payload* available:

- "pretty_print": "true" / "pretty_print": "false" - whether pretty printing should be enabled or not. By default is false.
- "pretty_print_tabs": "true" / "pretty_print_tabs": "false" - whether to use tabs (or spaces) for pretty print. By default is false (=> uses spaces).
- "pretty_print_spaces_count": <count>- the count of spaces in case of pretty print enabled. By default is 4.

Moreover optional column-specific *Payload* available:

- "xml": "<xml_type>" - column to be used as a specific xml type, available values follow

xml_type options:

- "attribute" - column name is beeing used as a xml attribute name and column value as xml attribute value
- "cdata" - column value is beeing used as a xml cdata (<![CDATA[...]]) contents
- "comment" - column value is beeing used as a xml comment (<!--...-->) contents
- "element" - column name is beeing used as a xml element name
- "value" - column value is beeing used as a xml element value

For **example**, input JSON:

```
{
  "template": "xml",
  "count": 3,
  "columns": [
    {
      "name": "name1",
      "type": "const",
      "payload": {
        "value": ""
      },
      "columns": [
        {
          "name": "name2",
          "type": "const",
          "payload": {
            "value": "value2",
            "xml": "attribute"
          }
        }
      ],
    },
    {
      "name": "name3",
      "type": "const",
      "payload": {
        "value": ""
      },
      "columns": [{
        "name": "name3value",
        "type": "const",
        "payload": {
```

```
        "value": "value3",
        "xml": "value"
      ]}
    ]
  },
  "payload": {
    "pretty_print": true
  }
}
```

results in:

```
<name1 name2="value2">
  <name3>value3</name3>
</name1>
<name1 name2="value2">
  <name3>value3</name3>
</name1>
<name1 name2="value2">
  <name3>value3</name3>
</name1>
```


Syntax:

```
{
    "name": "<name>",
    "type": "<type>",
    "payload": <payload>
}
```

Legend:

- *<name>* - name of the column
- *<type>* - type of the column
- *<payload>* - *Payload*

Following column types are available:

- **address:**
 - *Column: address.continent*
 - *Column: address.country*
 - *Column: address.city*
 - *Column: address.phone*
 - *Column: address.state*
 - *Column: address.street*
 - *Column: address.zip*
- **color:**
 - *Column: color*
 - *Column: color.hex*
- *Column: const*

- *Column: copy*
- **credit card:**
 - *Column: credit_card.number*
 - *Column: credit_card.type*
- **currency:**
 - *Column: currency*
 - *Column: currency.code*
- **date:**
 - *Column: date.day.of_week*
 - *Column: date.day.of_week*
 - *Column: date.day.of_month*
 - *Column: date.month*
 - *Column: date.month.name*
 - *Column: date.year*
 - *Column: date.of_birth*
- *Column: join*
- **name:**
 - *Column: name.first*
 - *Column: name.first.female*
 - *Column: name.first.male*
 - *Column: name.full*
 - *Column: name.full.female*
 - *Column: name.full.male*
 - *Column: name.last*
 - *Column: name.last.female*
 - *Column: name.last.male*
- *Column: regex*
- *Column: row_index*

Column: address.continent

Generates the random continent name.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "address.continent"
}]
```

could result in value:

```
Europe
```

Column: address.country

Generates the random country name.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "address.country"
}]
```

could result in value:

```
Slovakia
```

Column: address.city

Generates the random city name.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "address.city"
}]
```

could result in value:

```
London
```

Column: address.phone

Generates the random phone number.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "address.phone"
}]
```

could result in value:

```
3-456-437-63-83
```

Column: address.state

Generates the random state name.

For **example**, input JSON:

```
"columns": [{  
  "name": "name1",  
  "type": "address.state"  
}]
```

could result in value:

```
North Carolina
```

Column: address.street

Generates the random street name.

For **example**, input JSON:

```
"columns": [{  
  "name": "name1",  
  "type": "address.street"  
}]
```

could result in value:

```
Eagle Crest Drive
```

Column: address.zip

Generates the random zip name.

For **example**, input JSON:

```
"columns": [{  
  "name": "name1",  
  "type": "address.zip"  
}]
```

could result in value:

```
9393157
```

Column: color

Generates the random color name.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "color"
}]
```

could result in value:

```
Green
```

Column: color.hex

Generates the random hexadecimal value of the color.

Optional *Payload* available:

- "short": "true" / "short": "false" - whether short version of the hexadecimal value should be generated or not. By default is false.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "color.hex",
  "payload": {
    "short": true
  }
}]
```

could result in value:

```
390
```

Column: const

Generates constant value provided in payload.

Mandatory *Payload* available:

- "value": <value> - the constant value to generate

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "const",
  "payload": {
    "value": "foo"
  }
}]
```

results in value:

```
foo
```

Column: copy

Generates the same value as the column referred.

Mandatory *Payload* available:

- "from": "<column_name>" - the column name whose value is to be copied.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "const",
  "options": {
    "value": "foo"
  }
}, {
  "name": "name2",
  "type": "copy",
  "options": {
    "from": "name1"
  }
}]
```

results (for columns: name1 as well as name2) in value:

```
foo
```

Column: credit_card.number

Generates the random credit card number value.

Optional *Payload* available:

- "type": "<column_name>" - the type of credit card to generate number of. Valid values are: amex, discover, mastercard and visa.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "credit_card.number",
  "payload": {
    "type": "amex"
  }
}]
```

could result in value:

```
4771761587281649
```

Column: credit_card.type

Generates the random credit card type value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "credit_card.type"
}]
```

could result in value:

```
American Express
```

Column: currency

Generates the random currency value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "currency"
}]
```

could result in value:

```
New Zealand Dollars
```

Column: currency.code

Generates the random currency code value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "currency.code"
}]
```

could result in value:

```
GBP
```

Column: date.day.of_week

Generates the random weekday number value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "date.day.of_week"
}]
```

could result in value:

2

Column: date.day.of_week

Generates the random weekday name value.

Optional *Payload* available:

- "short": "true" / "short": "false" - whether short version of the weekday name should be generated or not. By default is false.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "date.day.of_week.name",
  "payload": {
    "short": true
  }
}]
```

could result in value:

Thu

Column: date.day.of_month

Generates the random day of month value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "date.day.of_month"
}]
```

could result in value:

21

Column: date.month

Generates the random month value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "date.month"
}]
```


could result in value:

```
11
```

Column: date.month.name

Generates the random month name value.

Optional *Payload* available:

- "short": "true" / "short": "false" - whether short version of the month name should be generated or not. By default is false.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "date.month.name",
  "payload": {
    "short": true
  }
}]
```

could result in value:

```
Aug
```

Column: date.year

Generates the random year value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "date.year"
}]
```

could result in value:

```
1448
```

Column: date.of_birth

Generates the random date of birth value.

Optional *Payload* available:

- "age": <age> - the age that date of birth should be generated for. If not specified, random age in interval 0-120 is used.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "date.of_birth",
  payload {
    "age": 18
  }
}]
```

could result in value:

```
1998-02-22 22:08:28 +0100 CE
```

Column: join

Joins nested column values with the separator (optional) provided.

Optional *Payload* available:

- "separator":<separator> - the separator string to be used for joining values.

For **example** (without separator), input JSON:

```
"columns": [{
  "name": "name1",
  "type": "join",
  "columns": [{
    "name": "name1",
    "type": "const",
    "payload": {
      "value": "value1"
    }
  }, {
    "name": "name2",
    "type": "const",
    "payload": {
      "value": "value2"
    }
  }
}]
}]
```

would result in value:

```
value1value2
```

For **example** (with separator), input JSON:

```
"columns": [{
  "name": "name1",
  "type": "join",
  "columns": [{
    "name": "name1",
    "type": "const",
    "payload": {
      "value": "value1"
    }
  }, {
```

```
    "name": "name2",
    "type": "const",
    "payload": {
      "value": "value2"
    }
  }
}, "payload": {
  "separator": ", "
}]
```

would result in value:

```
value1,value2
```

Column: name.first

Generates the random first name value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "name.first"
}]
```

could result in value:

```
Malcolm
```

Column: name.first.female

Generates the random female first name value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "name.first.female"
}]
```

could result in value:

```
Sherly
```

Column: name.first.male

Generates the random male first name value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "name.first.male"
}]
```

could result in value:

```
Brandon
```

Column: name.full

Generates the random full name value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "name.full"
}]
```

could result in value:

```
Katrina Vanhamlin
```

Column: name.full.female

Generates the random female full name value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "name.full.female"
}]
```

could result in value:

```
Katrina Vanhamlin
```

Column: name.full.male

Generates the random male full name value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "name.full.male"
}]
```

could result in value:

```
Stephan Mciltrot
```

Column: name.last

Generates the random last name value.

For **example**, input JSON:

```
"columns": [{  
  "name": "name1",  
  "type": "name.last"  
}]
```

could result in value:

```
Vanhamlin
```

Column: name.last.female

Generates the random female last name value.

For **example**, input JSON:

```
"columns": [{  
  "name": "name1",  
  "type": "name.last.female"  
}]
```

could result in value:

```
Vanhamlin
```

Column: name.last.male

Generates the random male last name value.

For **example**, input JSON:

```
"columns": [{  
  "name": "name1",  
  "type": "name.last.male"  
}]
```

could result in value:

```
Mciltrot
```

Column: regex

Generates the random string matching the specified regular expression (to examine full capabilities, refer to project: [lucasjones/regex](#) being used under the hood),

Mandatory *Payload* available:

- "pattern": <pattern> - the pattern to match.

Optional *Payload* available:

- "limit": <limit> - the maximum number of times *, '+' should repeat.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "regex",
  "payload": {
    "pattern": "z{1,3}",
    "limit": 10
  }
}]
```

could result in value:

```
zzz
```

Column: row_index

Generates the current row index value.

For **example**, input JSON:

```
"columns": [{
  "name": "name1",
  "type": "row_index"
}]
```

results in values:

```
0
1
2
3
...
```

All notable changes to this project will be documented in this file.

Datarator is in a pre-1.0 state. This means that its APIs and behavior are subject to breaking changes without deprecation notices. Until 1.0, version numbers will follow a [Semver-ish](#) `0.y.z` format, where `y` is incremented when new features or breaking changes are introduced, and `z` is incremented for lesser changes or bug fixes.

Unreleased

0.2.1 (2017-02-06)

Fixes:

- binaries release fixed

0.2.0 (2016-10-10)

Known Issues:

- binaries release failed (no binaries provided)

Features:

- CLI:
 - options: configurable chunk size
- API:
 - column payload: `emptyPercent`
 - response gzip support
 - response header: `Content-Encoding`

- timeout on data generation
- removed api: GET /
- template:sql removed whitespaces
- json schema updated for usage with [jdorn/json-editor](#)

Fixes:

- using proper sql mime type

0.1.0 (2016-09-06)

- Initial release