DataGen Main

Main Menu

Requirements

Definitions

General Discussion

Order of template building

ToDo

ToDo

1. Create a State Diagram for building Template from UI
2. Create a State Diagram for processing Template
3. Create a Sequence Diagram for interactions with UI, API, Server side code
4. Build a set of classes to hold the different parts of the template

Build a set of classes to hold the different parts of the template

Notes

Comments

Task Reference

Most of this is academic. It's not going to be seen by most users, I assume, since the normal user will prob just create the template then either be done or save and retrieve it later. If the choice is to keep a copy the ability to look at it is high, modify is less. So, what each section is called may not be something that matters at so yet. Nor the way I store it.

1. _id - the id is the template uuid and is used to distinguish this template over any other.
2. Description - Description is an optional area that is used to keep information about the template. For example the name, creation date, updated_by, modified date, version, description, meta data (template uuid?)
3. File_Directives - Directives is a required area that will give direction to the server on how to process the document as a whole. Total records, output type, delimiters etc.
4. Row_Definitions - defines the rows (if > 1) and columns and how they interact with each other.
5. Row_Directives - defines how the rows interact with other rows (TBA)

```
{
  "id":"",                // guid v4
  "description": {},       // Object with non directive information
  "file_directives":{},        // Object with document directives
  "row_definitions":[ ROW_DEFINITION, ROW_DEFINITION, ...,  ROW_DEFINITION]    // row definition objects array
  "row_directives":[ ROW_DIRECTIVE, ROW_DIRECTIVE, ..., ROW_DIRECTIVE]
}
```

Row_Directives

ROW_DIRECTIVE: the Row Directive gives direction to the server on how this row interacts with other rows.

```
{
  ”column_id”:0,
  ”directive_type”:0,
  ”directive_type_object”:{ }
}
```

<need types> **DIRECTIVE_TYPE_OBJECT**

```
{
  ”row_id”:0
}
// need to flesh this out.  How will this directive tell the system that row_id:x is acted upon
// or acts upon another row_id:n
```

Description

Description is an optional section of the document template. This has no directive that are used by the system. It will be saved by the system for the user to use as required. The user can put self defined attributes in this object and the system will not object.

```
{

 "description": {

     ”document_name”: ””,
     ”description”: ””,
     ”version”:””,
     ”creation_date”: ””,
     ”created_by”:””,
     ”last_modified”:””,
     ”last_modified_by”:””
  }
}
```

Row_Definitions

The row definitions is where the rubber meets the road. Here the rows and columns will be defined. Interactions between them will be created.

ROW-DEFINITION: A row will have a small description, and an array of COLUMNS, and a ROW DIRECTIVE object.

```
{
  ”name”:””,
  ”description”:””
  ”id”:””
  ”row_type”:ROW_TYPE,
  ”column_definitions”:[ COLUMN_DEFINITION, COLUMN_DEFINITION, ... ],

}
```

Attributes of a **ROW_DEFINITION**:

attribute	**description**
name	a generic name, human readable the user gives this row
description	the description of this row, again, human readable
id	A system generated v.4 uuid
row_type	The user can designate this row as a special, non body, which will make the system act according to the type

column_definitions	an array of 1..n COLUMN_DEFINITION objects which, in order, create the row

ROW_TYPE: designate the row so the system will implement it correctly.

type	description
HEADER	This row is designated as a header record. System will create ONLY 1 (one) record. It will be at the top of the file (first row or first file if multiple files).
DATA	This row is a data row and will be processed according to the
FOOTER	This row is designated as a footer record. System will create ONLY 1 (one) record. It will be the very last record of the file (last row of the last file if multiple files).

COLUMN_DEFINITION: A column will be what defines the data generated.

```
{
  "name":"",
  "description":"",
  "data_type":"",
  "column_size":"",
  "error": false,
  "rule":{
    "name":"",
    "dataset":"",
    "paramater":[]
    },
  "modifications":[],
  "persistents":[]
}
```

attributes of a **COLUMN_DEFINITION**:

attribute	description
name	the human readable name of a column, one that will also be the header for the column
description	The human readable description that is only used as a note for the user
data_type	the type of field this column will display (basic types)
column_size	numeric size of this column
error	binary to if this column can be used as an error for the row
rule	Object to define the data generation
modifications	array of mods that are applied to the data after generation and before it is applied to the column
persistents	An array of persistent values that is particular to this column and lets the system know if new data is generated.

File_Directives

Directive object is the overall way the server will handle this document. Some information will be the total records, total documents (if they need to be split up) total records per document, document type.

The document type will be different and have different information.

```
{
  "file_directives": {
    "file_name":"",
    "total_records":0,
    "error_percent":0,
    "output":{}
  }
}
```

attributes for FILE_DIRECTIVES

attribute	description
file_name	the output file name when sent to the system
total_records	total number of records generated not counting a header / trailer if designated
error_percent	calculated from the total_records, a % of the records will have a column with incorrect type of data or size. ***Note*:** This will not be implemented until later updates.
output	an object, described below, is different according to the **output.type**

OUTPUT object is according to the output type.

type	description
JSON	The output will be a set of JSON objects
DELIMITED	The output will be delimited by some given delimiter
TEXT	The output will be plain text, more than likely fixed width as set by the **COLUMN_DEFINITION.size**

JSON type object

```
{
  "type": JSON,
  "prettyPrint":false,
  "invalid_json_document": false
}
```

DELIMITED type object

```
{
  "type": DELIMITED,
  "delimiter": "",
  "output_header_record":false,
  "quote_string_column": false
}
```

TEXT type object

```
{
  "type":TEXT,
  "output_header_record": true,
  "quote_string_column": false
}
```

General Discussion

This is the 2.0 version of a Data Generator.

Some History:

I was working on a project which we saw the need for a lot of generated data since most of the real data was PII or PHI. I wanted to make it configurable and repeatable. I wanted to be able to add "rules" and have an output of JSON objects.

It was ok, I liked it and we used it multiple times. It saved us a lot of time not having to de-identify data and get it approved to remove from a secure system.

But, it had issues. As I updated and added new features I found it was harder to use than I wanted it to be. I gave the project to others and started dreaming of how to make it better.

Next Gen

The main requirements were to have the 'configuration' be created by a UI, to be able to create millions (I wrote that right) and not just hundreds or thousands.

The first part is solvable mainly because I am a TS /Angular fanboy. So, I will build this on an angular platform.

The second was a bit harder to deal with. I am thinking of all the work is to be done on the server. So, the UI builds the template and sends it to the server to run. I am thinking also, if the engine runs and stores the data into a db somehow then it can rack up all the rows of data it wants (there is the Million). But, there are issues in this.

1. I want a way to build the data and get it downloaded. You're not going to download xMillion record file. So, it needs to be broken up.
2. If you break them up how to you get them downloaded (zip?)
3. There is the ability to designate that a certain field is Unique. So, as the RULE builds the data it has to check that it was not already used for that field in a previous ROW.
    1. What does that do to speed?
    2. I have not even addressed time/speed
    3. There must be some intermediate tables/db files that hold the unique values and then searched EACH TIME another is built.
    4. Can the server run concurrent builds? If it has the ability to look at the unique fields of the other threads then ... maybe?
4. What happens to the template after the run is complete?
    1. I figured it could be stored
    2. I could be downloaded to the user
    3. can it be modified and uploaded?
    4. If it is stored in the system then there is login/user data ... ugh.

Requirements

1. System shall have a UI component
    1. Document configuration is done via UI
    2. Ability to store frequently used configuration
    3. Shall be able to download configuration
    4. Shall be able to upload configuration
        1. Shall be able to verify/validate configuration
        2. correctness
2. System shall have a API component
    1. login users? or just let them have a named space for templates.
    2. template retrieval
    3. document send
3. System shall have a db Component
    1. document configuration templates
    2. Store legit user's configuration
    3. document temporary data
4. Configuration Template shall contain
    1. Options
    2. **Documentation**
    3. 1..* named **ROW** types
    4. **ROW** shall have

    1. named **column with definitions**
5. **Column Definitions** shall have
    1. Name
    2. Data type
    3. Data size
    4. Rules
    5. Modifications
    6. Persistent

Persistent

Persistent is an array in the column definition.

There are specific keywords that are allowed in persistent. Each keyword will allow the data generator to act differently for the specific column.

keyword	**Description**
Singular	Singular will use a column's RULE to build the first data for the column. After that, for each row in this column the same data value will be used.
Unique	Unique will insure that each time a value is generated it has not been already used in this column for this document. This will not ensure that the value is in another column in the document.
Increment	Increment will use the RULE to build the first data for the column (number) and then it will subsequently increment that value by 1 (one). The number can begin at any value according the the parameter
Tally	Tally will keep a running total (sum) of this column. It can be retrieved using the RULE: GrandTotal

Options

Options is a section of the template where it instructs the data generator system to act in a specific way, generally. Some options are

option	**description**
total	The total number of rows of generated data to be produced
format	The output format such as csv, JSON
delimiter	The delimiter used in csv (xsv if you will)
pretty print	If output is JSON make the output human readable
Error	Create records with errors in them. ***This will not be implemented in the first few minor versions of v2.0***

Error

This is a part of the Column definition which instructs the system to randomly create bad data in the field. This will either create no data when data is required or will create bad data (number in string field, string in number, bad date values).

This will not be implement in the first few minor implementations of v2.0.

Rules

1. The Rules will be created ahead of time and compiled into the system. Some rules would be:

   Singular = means the number will not be repeated for this column on this document

   Increment = the number will increase by 1 on all subsequent rows

Rule	**Description**	**Dataset**	**Parameter**	**Modification**	**persistent**
Number	Build a number according to the parameters specified. The output will be a number unless a Modification makes it a string	N/A	["startAt", n] ["between",n,m] if no parameter then it will default to ["startAt",0]		Singular Total
Date	Build a data according to the parameters specified	N/A	["past", n, m, timeframe] ["future", n, m, timeframe] n,m are integers between timeframe (days, months, years) date is calculated. ["now"] ["age", string] use a previously created attribute's number as the age to build date from. ["set", date] set a specific date if no parameters given then default is ["now"]	["format", string]	Singular
Parameter	Randomly select data given in the parameters	N/A	required: [value1, value2, ... valueN] values used by rule		
Character	Randomly choose characters. If parameters are given then they will focus the set. Character Rule will use Number, letters, SpecialCharacters to select from.	N/A	["includeNumbers"] ["includeSpecialChar"]		
Dataset	Select from the given Dataset a value	see list of Datasets to choose from	["limit",n]		Singular
Format	Create an output using random Characters and numbers according to the Parameter		required: [format String] %3s-%2d.%2d = xxx-nn.nn		Singular

SSN	Create an output using the SSA's rules for test SSN/TNN	N/A	["useDashes"] Will put dashes in format :999-99-9999		Singular
SameAs	use the result of an attribute which is already created on this row (Cannot use SameAs for a future attribute)	N/A	required: [attribute]		
Binary	Randomly select True/False	N/A	["yn"] output "yes"/"no" string ["10"] output "1" / "0" string with no parameters the default is true/false	[upperCase, lowerCase, capitalCase, limit]	
GUID	Build a 5 segment v4 guid in format "xxxxxxxx-xxxx-Mxx-Nxx-xxxxxxxxxxxx"	N/A	["version", 1..4] to 'force' the M to number but know that it will not conform to uuid standard. with no parameters the default is ["version", 4]		Singular
Gender	return Male, Female	N/A	["unk"] will add Unknown as a value randomly ["na", binary] will randomly put "N/A" as gender or if binary=true leave blank.		
Address	builds a typical address with the format of number+space+street name+space +street type	N/A uses streetNames, streetTypes			Singular
Email	Build an email address according to parameters local@hostname.domain	N/A uses: Characters, FirstName, LastName, Hostnames, Domains	["attributes", string1, string2, binary] builds an email addressing using data in attribute string1 and string2. if binary = true then use first letter in string1 and string2 as local. no parameters will generate a random string 9-13 char long as local		Singular Unique
GrandTotal	will put the running total of an attribute.	N/A	required: ["attribute"] the column which used		

			the **persistent.total** keyword.		

Modifications

- MOdification

Modification	use	Description
pre	["pre", string]	prepend 'string' on the data element
post	["post", string]	append 'string' on the data element.
peri	["peri", string1, string2]	a combination of Pre/Post where string1 is prepended and string2 is appended to the data element.
format	['format', string]	Format the output according to specification for the rule. ie date use YYYY,etc, Number us 999.99
multiplicity	["multiplicity", n, string]	create multiple output for this data as an array of data or quoted csv. The string parameter is separate value. ie ["multiples", 4, "|"] creates the data as a pipe delimited string like "Joe|Carl|Bill|Mike"
lowerCase	["lowerCase"]	for String/Character output the data in all lowercase letters
upperCase	["upperCase"]	for String/Character output the data in all uppercase letters
capitalCase	["capitalCase"]	For String/ multiple Characters, have the first letter uppercase and the remaining lowercase
limit	["limit", n]	Limit the output to n characters. This is mainly used for dataset like lorem-ipsum where the output may be too long.
toString?	["toString"]	Mainly used on Number output to format the output number as a string
pad	["pad", n]	Pad the output with n spaces. On numbers the spaces will be prepended the number, on string/char the spaces will be appended.
money	["money", char]	format the output (number) to be a money value pre-pending char. ["money", "$"] make 12345 => $123.45 String

Order of template building

1. Create the new Template
2. Define Document Options
3. Create a ROW
4. Create COLUMN
5. Define the COLUMN Options

Definitions

Term	**Definition**
Row	This is a row of data that has a set of columns of data. There can be `n` number of types of rows. They can inter-relate to other rows.

	Note: the term Row is interchangeable with JSON object (if JSON is the selected output option). This document will use the term ROW.
Column	This is the data in a row. There can be x number of columns in a **row**. Data is defined by a **rule**. the Data is manipulated by **parameters** (params). Data is augmented by **modifications** (mods). **Note**: the term Column is interchangeable with attribute in a JSON object. This document will use the term COLUMN.
Column Definition	A Column Definition defines how data is generated and how it will be presented. Column Definition has the **name**, the **rule**, rule **parameters** (params) , rule **datasets**, final **modifications** (mods), p**ersistent**, and other **triggers**.
Name	The Name is a required part of the column definition. It is the column name if a header record is required, and the attribute of a JSON object if the output is JSON data.
Rule	The Rule is a required part of the column definition. A Rule defines what is put into a column of data. The rule defines how the data is generated or calculated. Rules may use a **Dataset** for the data. Rules may also use **Parameters** when building or selecting the data.
Dataset	the Dataset is an optional part of the Rule. A Dataset is a predefined set (1 .. *) of values a Rule can use to select and use as the output for the column. Datasets include Names, Cities, Counties, Languages, Letters, Special Characters etc.
Parameter	The Parameter is an optional part of the Rule. A Parameter is a limiting or further defining factor(s) for a rule when generating data.
Modification	The Modification is an optional part of the column definition. Modifications (mods) are manipulations set to the data after the Rule has completed. Mods are predefined and allowed/limited to a Rule but not part of the Params. Many Mods can be used on all Rules.
Persistent	The Persistent is an optional part of the column definition. It is an array of keywords which will tell the data generator to handle this column in a very special way. The keywords are **Singular**, **Increment**, **Unique**, **Tally**. These keywords are by default off. By including them in the Persistent activates them.
Trigger	Trigger is an optional part of the column definition. Is a way for the data designer to add a small snippet of javaScript code as a modification to the generated data. ***This feature will not be implement in the first few minor releases of 2.0.***

Exported from Living Spec on Sun Feb 06 2022