dylan kessler

dylanikessler@yahoo.com

Abstract

Generate Massive amounts of Testing Data! Make data for tables! Make Streaming data! All of your test data needs in one place!

MsgGen

the Data Generator

# MsgGen – the Data Generator

Contents

[MsgGen – the Data Generator 1](#_Toc23006345)

[Introduction – the purpose of this program 3](#_Toc23006346)

[How the program works – high level summary 3](#_Toc23006347)

[Getting Started: How to run the program – invocation 3](#_Toc23006348)

[Modes of operation and the configuration file 5](#_Toc23006349)

[***Mode 0 – generate Social Security Numbers*** 6](#_Toc23006350)

[***Mode 1 – generate primary Keys*** 6](#_Toc23006351)

[***Mode 2 – not used*** 7](#_Toc23006352)

[***Mode 3 – (Disabled) generate people and addresses*** 7](#_Toc23006353)

[***Mode 4 – Main feature: generate files with records of random data based on an input spreadsheet that the user specifies*** 7](#_Toc23006354)

[***Mode 5 – HQL script generation*** 7](#_Toc23006355)

[***Mode 6 – HQL script generation*** 8](#_Toc23006356)

[***Mode 7 - Add primary keys*** 8](#_Toc23006357)

[***Mode 8 – Push to Kafka Topic*** 8](#_Toc23006358)

[Deeper Explanation of Mode 4(the MASTER MODE) 9](#_Toc23006359)

[Single Sentence summary of what this mode does 9](#_Toc23006360)

[Input spreadsheet – the end user’s data requirements 11](#_Toc23006361)

[Fields, data types, formatting rules, and qualifiers 12](#_Toc23006362)

[**Fields** 12](#_Toc23006363)

[**Data types** 12](#_Toc23006364)

[**Formatting Rules** 13](#_Toc23006365)

[**Qualifiers** 13](#_Toc23006366)

[Example 14](#_Toc23006367)

[More in-depth with the minor modes 19](#_Toc23006368)

[***Mode 5 and 6: Hive SQL generation*** 19](#_Toc23006369)

[Examples 20](#_Toc23006370)

[***Example 1:*** 20](#_Toc23006371)

[***Example 2:*** 21](#_Toc23006372)

[***Mode 7: Primary Key Functionality*** 22](#_Toc23006373)

[Examples 22](#_Toc23006374)

[***Example1:*** 22](#_Toc23006375)

[***Mode 8: Kafka Topic functionality*** 25](#_Toc23006376)

[Examples 26](#_Toc23006377)

[Building and Deployment 28](#_Toc23006378)

[Log Files 28](#_Toc23006379)

[In the Works: Future Enhancements 29](#_Toc23006380)

[Appendices 30](#_Toc23006381)

[Appendix 1 – Flow Diagrams 30](#_Toc23006382)

[Figure 12 - overall flow diagram for modes 4,5,7, and 8 31](#_Toc23006383)

[Figure 13 - How mode 7 works 32](#_Toc23006384)

[Appendix 2 – msggen Log File sample 32](#_Toc23006385)

# Introduction – the purpose of this program

The MsgGen program is a program that generates random test data. It has a number of different data generation features. The different data generation features are identified by specifying a particular mode. The main feature(mode 4) generates files of random data based upon properties specified in an input excel spreadsheet.

# How the program works – high level summary

Getting Started: How to run the program – invocation

There is an artifact for the current build of this program. This jar file is located in the artifact folder of the repository. For developers, the program can be built using either sbt or maven. Once you have built and packaged the program as an executable jar file, you can run it using the following convention:

**java -jar <jarname> <cf path >**

where:

jarname = full path to the jar file name

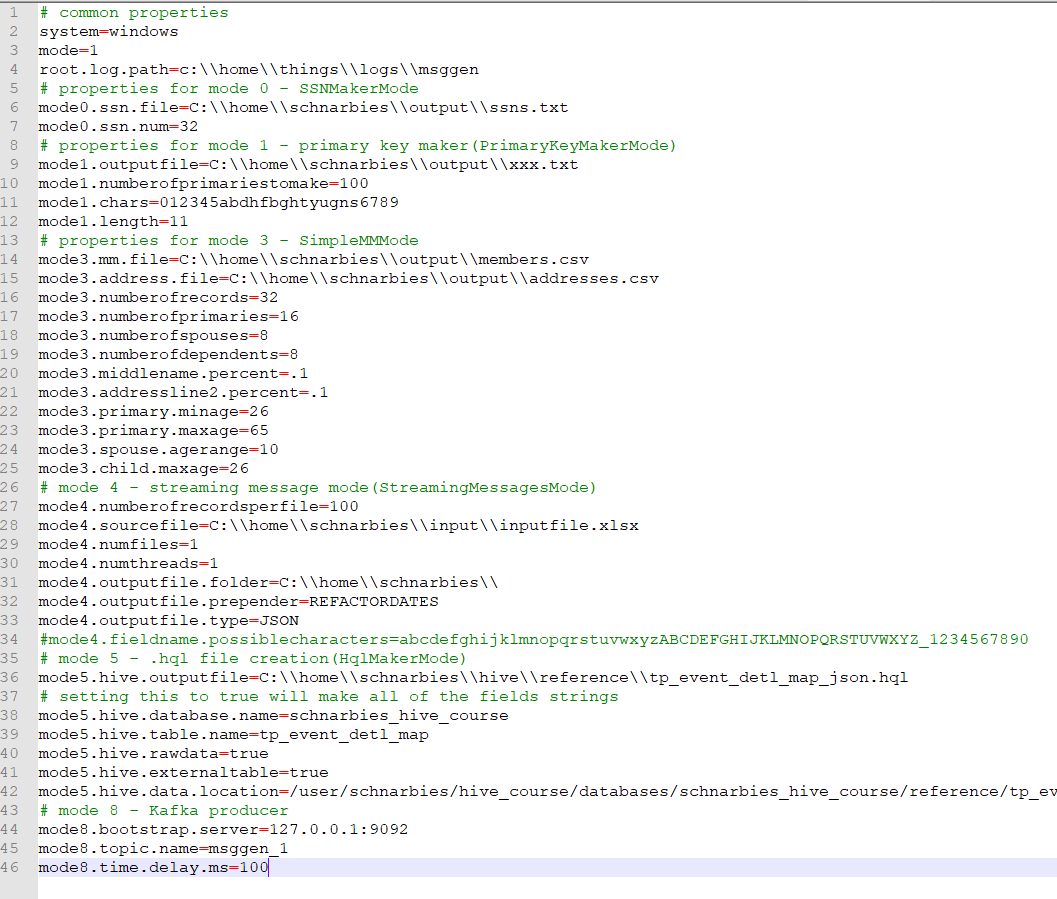
cf path = full path to the main configuration file name

The main configuration file

The program uses a main configuration file, which you can name however you want, since you will be giving this name to the program by specifying it as the cfpath in the jar invocation statement. For the purpose of this document, we will refer to this file as **config.properties**. In addition, mode 4(and the other modes associated with this mode) will use another input excel spreadsheet, whose name you will specify in the config.properties file(see mode 4 description for more details).

Three main config.properties properties: System, MODE, and root.log.path

Shown below is a sample of our main config.properties file. All the properties common to the application are listed first, then the mode-specific properties. Here we see there are only 3 common properties: **system, mode, and root.log.path**:



**System** – the operating system that the program is running on. Currently only “windows” and “linux” system values are accepted. Actually, what the program does is it checks to see if it is windows and if so, it handles file paths for a windows system, otherwise it handles file paths for a linux system.

**Mode** – the particular mode that you are running the program as. The modes of operation are discussed shortly.

**Root.log.path** – the root folder where all of your log files will be located. See the logging section for which log files are generated.

Modes of operation and the configuration file

The different modes and what they do are explained below:

Mode 0 – generate Social Security Numbers

Minor feature

This mode will simply generate a text file with a set number of random social security numbers, padded to 9 digits, without any hyphens (ex.: 123456789). All the SS numbers will be unique, there will be no duplicates.

The configuration properties for this mode are:

|  |  |
| --- | --- |
| Field name | mode0.ssn.num |
| explanation | how many SS numbers to make |
| example | 100 |
| notes | Must be a whole number greater than zero |

|  |  |
| --- | --- |
| Field name | mode0.ssn.file |
| explanation | full path to the output file to contain these social security numbers |
| example | C:\\home\\schnarbies\\ssn.txt for windows or  /home/schnarbies/ssn.txt for linux |
| notes | needs to be in the operating system format that the program is being run on; the program does not currently have a conversion feature for this |

Mode 1 – generate primary Keys

Minor feature

This mode is a little bit more in depth than Mode 0. For this mode, you specify a list of characters, a length, and how many keys to make, and this mode will randomly generate that many unique keys of the given length, using the characters given in mode1.chars. \*Note – because it generates unique keys, the program will not run if the total # of different possibilities is less than the number of keys to make that is specified! For example, you can’t say:

mode1.chars = a

mode1.numberofprimariestomake = 100

because with only 1 character, there is only 1 possible combination

The configuration properties for this mode are:

|  |  |
| --- | --- |
| Field name | mode1.outputfile |
| explanation | full path to the output file to contain these primary keys |
| example | C:\\home\\schnarbies\\output.txt for windows or  /home/schnarbies/output.txt for linux |
| notes | needs to be in the operating system format that the program is being run on; the program does not currently have a conversion feature for this |

|  |  |
| --- | --- |
| Field name | mode1.numberofprimariestomake |
| explanation | how many primary keys to make |
| example | 100 |
| notes | Must be a whole number greater than zero |

|  |  |
| --- | --- |
| Field name | mode1.chars |
| explanation | list of characters that can be in the primary key |
| example | 012345abdhfbghtyugns6789 |
| notes |  |

|  |  |
| --- | --- |
| Field name | mode1.length |
| explanation | length of each primary key |
| example | 10 |
| notes | Must be a whole number greater than zero |

Mode 2 – not used

Mode 3 – (Disabled) generate people and addresses

This mode is a legacy mode that I used when initially creating the program. Its disabled now, but it may be enhanced and brought back at some point in the future.

Mode 4 – Main feature: generate files with records of random data based on an input spreadsheet that the user specifies

MAJOR feature

This is the MASTER MODE that is the heart of the program. This program generates multiple files of records in a format specified by the user in an Excel spreadsheet. Before explaining how this MASTER mode works, here is a summary of the other minor modes that are coupled with this MASTER mode:

Mode 5 – HQL script generation

generates a basic Hive table creation script based off of the Spreadsheet specified for mode 4. Minor feature. More description about this feature follows the explanation of mode 4.

The configuration properties for this mode are:

|  |  |
| --- | --- |
| Field name | **mode5.hive.outputfile** |
| explanation | The name of the output hive file to generate, full path |
| example | C:\\home\\schnarbies\\hive\\reference\\myTable.hql for windows  /home/schnarbies/hive/reference/myTable.hql for linux |
| notes | needs to be in the operating system format that the program is being run on; the program does not currently have a conversion feature for this |

|  |  |
| --- | --- |
| Field name | **mode5.hive.database.name** |
| explanation | The name of the database to specify in the hive file |
| example | Schnarbies\_db |
| notes |  |

|  |  |
| --- | --- |
| Field name | **mode5.hive.table.name** |
| explanation | The name of the hive table specified in the hive file |
| example | Schnarbies\_table |
| notes |  |

|  |  |
| --- | --- |
| Field name | **mode5.hive.rawdata** |
| explanation | Should the fields in the table script be all Strings? |
| example | True or false values only |
| notes | Value must be true or false; if true, the program will make all of the table fields Strings, otherwise, the program will look each data type to determine which Hive type to make it |

|  |  |
| --- | --- |
| Field name | **mode5.hive.externaltable** |
| explanation | Is the table an external table? |
| example | True or false values only |
| notes | Value must be true or false; if false, the script will include a “load data inpath” clause in it |

|  |  |
| --- | --- |
| Field name | **mode5.hive.data.location** |
| explanation | Where is the undelying data for this table stored? |
| example | /user/myuser/path/database/mytable |
| notes | This data location is assumed to use the standard linux file path conventions. Currently, the location is assumed to be hdfs |

Mode 6 – HQL script generation

this mode ONLY generates the Hive script, and does not make records. Minor feature. More description about this feature follows the explanation of mode 4.

The configuration properties are the same as for mode 5.

Mode 7 - Add primary keys

this mode couples with mode 1 and mode 4. It will replace the first field of each output record with a primary key generated using mode 1. Minor feature.

NOTE – a condition of running this feature in mode 7 is that the **mode4.numfiles** must equal 1.

More description about this feature follows the explanation of mode 4.

There are no configuration properties specific to this mode; they use the ones specific to modes 1 and 4.

Mode 8 – Push to Kafka Topic

This feature will periodically push each of the records generated to a Kafka Topic. Major feature. More description about this feature follows the explanation of mode 4. An explanation of how Kafka works is not covered in this manual.

The configuration properties for this mode are:

|  |  |
| --- | --- |
| Field name | **mode8.bootstrap.server** |
| explanation | The kafka bootstrap server IP address and port number |
| example | 127.0.0.1:9092 |
| notes |  |

|  |  |
| --- | --- |
| Field name | **mode8.topic.name** |
| explanation | The name of the topic to push the messages to |
| example | msggen\_1 |
| notes |  |

|  |  |
| --- | --- |
| Field name | **mode8.time.delay.ms** |
| explanation | The amount of time delay(in milliseconds) to countdown before pushing each record to the kafka topic |
| example | 100 |
| notes | Must be a whole number. Zero indicates no time delay. |

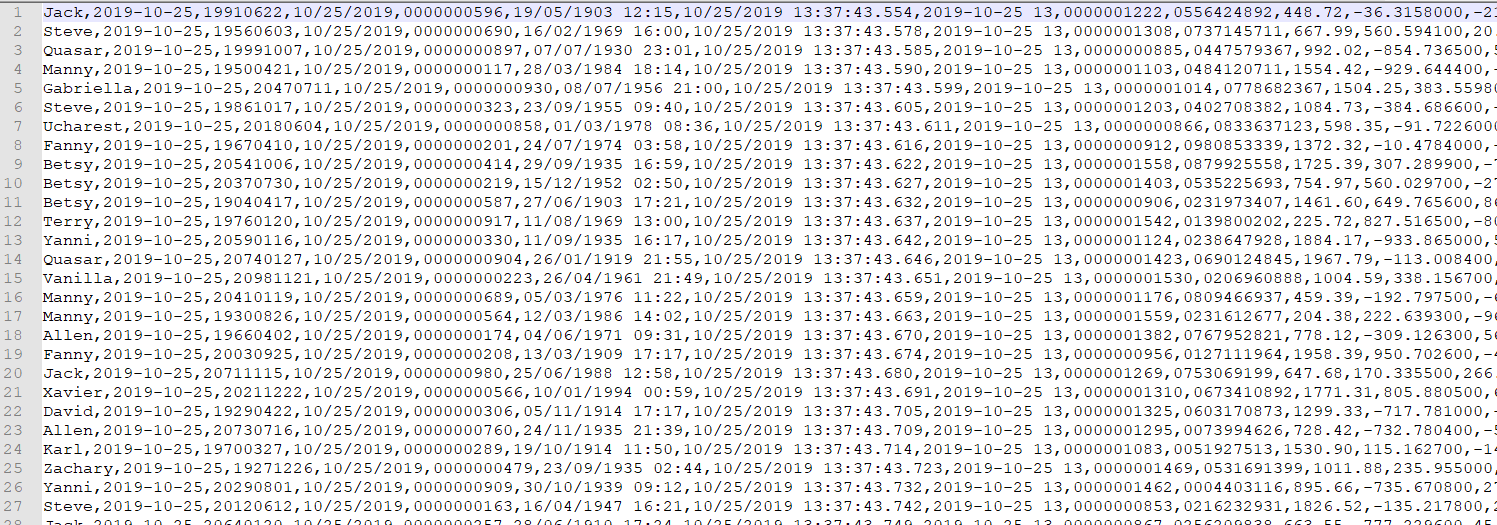
# Deeper Explanation of Mode 4(the MASTER MODE)

See Figure 12 in [Appendix 1](#_Appendices) for a flow of this feature.

Single Sentence summary of what this mode does

This program takes information contained in an Excel spreadsheet, and uses it to create files of records of random data, in either CSV or JSON format.

Here are examples of the output data that can be generated in both CSV and JSON:





We’ll start explaining how to use the MASTER feature by explaining the properties in **config.properties**.

Then we’ll explain how to fill out the input Excel spreadsheet.

Configuration properties

The only parameter that you have to specify when running the jar file is the full path to this configuration file. A list of the properties for mode 4 are as follows:

|  |  |
| --- | --- |
| Field name | **mode4.numberofrecordsperfile** |
| explanation | how many records per file do you want made? |
| example | 100 |
| notes | Must be a whole number greater than zero |

|  |  |
| --- | --- |
| Field name | **mode4.sourcefile** |
| explanation | where is the location of the input spreadsheet to use?(explained shortly) |
| example | C:\\home\\schnarbies\\input\\inputfile.xlsx for windows or  /home/schnarbies/input/inputfile.xlsx for linux |
| notes | needs to be in the operating system format that the program is being run on; the program does not currently have a conversion feature for this |

|  |  |
| --- | --- |
| Field name | **mode4.numfiles** |
| explanation | How many files to make |
| example | 4 |
| notes | Must be a whole number greater than zero. For mode 7, this must equal 1 |

|  |  |
| --- | --- |
| Field name | **mode4.numthreads** |
| explanation | How many threads to use when generating the records |
| example | 2 |
| notes | Must be a whole number greater than zero |

|  |  |
| --- | --- |
| Field name | **mode4.outputfile.folder** |
| explanation | The output folder to put the generated record files in |
| example | C:\\home\\schnarbies\\ for windows or  /home/schnarbies/ for linux |
| notes | needs to be in the operating system format that the program is being run on |

|  |  |
| --- | --- |
| Field name | **mode4.outputfile.prepender** |
| explanation | A prepender that you specify each of the output files to start with |
| example | myFile |
| notes | The name of each output file generated will start with this prepender, followed by “\_”, followed by the time in “HHmmss.SSS” format |

|  |  |
| --- | --- |
| Field name | **mode4.outputfile.type** |
| explanation | The formatting of the record data and the file type(extension) of the output files |
| example | CSV or JSON |
| notes | Currently the program only generates csv or json data |

|  |  |
| --- | --- |
| Field name | **mode4.fieldname.possiblecharacters** |
| explanation | A list of all of the possible characters that the fields names could contain |
| example | abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ |
| notes | This feature is not currently being used at this time |

Input spreadsheet – the end user’s data requirements

**\*\*\* 2 important notes about the input spreadsheet:**

1. **Before populating your spreadsheet** : make sure to format all of the cells in the spreadsheet as text.

Otherwise, when specifying dates, Excel tends to change date entries after you enter them.

(ex.: if you enter “01/01/2000” excel will change it to “1/1/2000” and the generator program will not recognize this as valid.)

1. Date and DateTime data types can have complicated formatting. **This program does not attempt to manually perform its own Date and DateTime formatting**; rather, **it trusts that the format entered by the user conforms to the Java standard RandomDate and RandomDateTime formats**. It is dependent upon the user to correctly specify the format of the Date or DateTime field.

Rules for specifying a "format" for the RandomDate and RandomDateTime fields can be found at:

<https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>

To use this program, the user needs to populate an Excel spreadsheet with information about each of the different data fields that they want generated in each record. Each column in the sheet represent a particular “piece” of data, or field, that you want each record to have.

A layout of the spreadsheet is shown below:

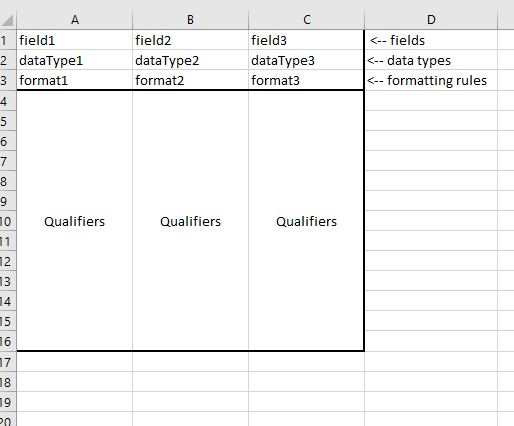


Figure - layout of the input spreadsheet

Fields, data types, formatting rules, and qualifiers

Below is a brief explanation of each of the parts to the sheet:

Fields

The name of each field. For JSON data, these will be the keys.

Data types

Each data type has two halves to it: it starts with the source type, and then ends with the actual type of data.

Source Type: each data type starts as either an Enum, an External, or a Random

1. Enum – the user will specify all of the different possible values that the field could be in the Qualifiers section
2. External – similar to the Enum, except for that the user will specify(in the Qualifiers section) the full path to an external file that contains all of the possible values that the field could be. The file itself will have 1 value per line.
3. Random – the program will generate the field, based on rules given in the formatting section.

Type of Data: currently can be one of the following:

String

Int

Long

Float

Double

Date

DateTime

Money

Examples of data types: EnumString, ExternalString, RandomString, EnumInt, ExternalLong, RandomMoney, EnumDate, RandomDateTime

Formatting Rules

The formatting rules field is a comma separated list of all of the rules to apply to a data type while generating it. Which rules are available to use depend on which data type is specified. Each formatting rule can be one of 2 types:

Rules that need qualifiers – ex.: min, max

Rules that need qualifiers will need to have additional information about them specified in the qualifiers section. For example, if you are generating a random integer between 1 and 10, for the format field, you will have “min,max” , and then the first two cells in the qualifier section will have 1 and 10, like this:

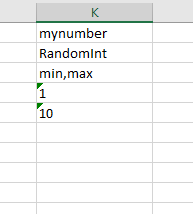


Figure - format and qualifiers for random number between 1 and 10

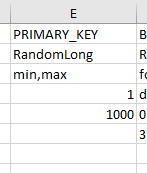
Rules that do not need qualifiers – ex.: upper, lower, rounddown

These rules do not need any other information about them in order for the program to use them. For example, “lower” is available for any of the String data types, and it tells the program that, for that field, after generating a random value for it, convert that value to lowercase.

**Qualifiers**

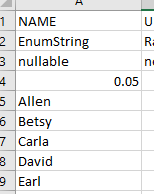
For each column in the sheet, rows 4 and below contain the qualifier data. There are 2 types of qualifier data:

**Values for the formatting rules that need qualifiers** – for each of the formatting rules that need qualifiers, the topmost qualifier rows contain the values for these rules. For example, below is a field for a RandomLong:



In this example, the formatting rules specified in row 3 are “min” and “max”, which need qualifier values. Therefore the 4th row must have the minimum value, and row 5 must have the maximum value, in that order.

Another example is for an EnumString data type, where the user wants there to be a 5% change that this field is missing a value(is null). We can achieve this using the “nullable” formatting rule. Below shows the configuration for this field:



Here, the formatting rule specified in row 3 is “nullable”, which needs a qualifier. The very first qualifier row(row 4) must contain the value for this nullable(the % chance of a null value), which is 0.05. The rest of the rows in this column contain all of the Enum Values.

**Enum Values** – these qualifiers are the actual values that an Enum data type could be. All of these types must occur after the qualifiers for the formatting rules.

There is a spreadsheet included in the repository’s documentation folder(/src/main/resources/documentation) called **VariableRules.xlsx**

This spreadsheet contains all of the various formatting rules and qualifier specifications for each of the data types.

Example

Let’s show a sample input spreadsheet, along with example output data to help clarify how the program works. Here is our example sheet:

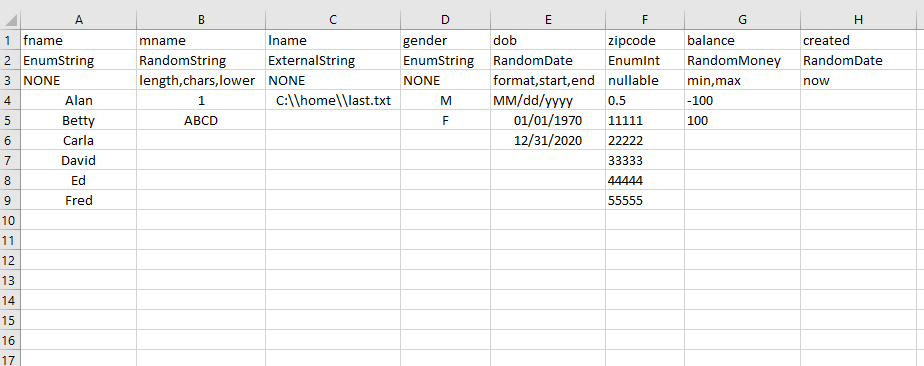


Figure – example.xlsx input spreadsheet

Let’s look at each field’s formats and qualifiers to see what they mean:

|  |  |  |
| --- | --- | --- |
| fname | | |
| Data type | EnumString | Generate a string; user will specify all possible values |
| Format | NONE | Nothing special about this field |
| Qualifiers | | Alan, Betty,Carla,David,Ed,Fred |
| End result | | This field will have a value randomly picked from the list “Alan, Betty,Carla,David,Ed,Fred” |
| mname | | |
| Data type | RandomString | Generate a string |
| Format | length,chars,lower | It will have a given length, there is a list of characters to chose from, the value will be converted to lowercase after it is generated |
| Qualifiers | 1 | The length will be = 1 |
|  | ABCD | The character values will be chosed from “A”,”B”,”C”,and “D” |
| End result | | This field will be a single character with a value of “a”,”b”,”c”, or “d” |
| lname | | |
| Data type | ExternalString | Generate a string |
| Format | NONE | Nothing special about this field |
| Qualifiers | C:\\home\\last.txt | This is where the external file is located that has all the values in it, see the file below: |
| End Result | | This field will have the values “Applewhite”, ”Booth”, Campra”, or “Dodd” |

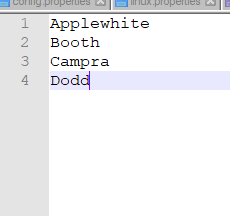
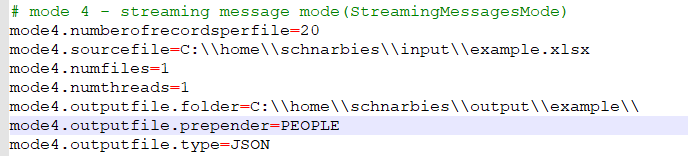


Figure - last.txt file with last names

|  |  |  |
| --- | --- | --- |
| gender | | |
| Data type | EnumString | Generate a string |
| Format | NONE | Nothing special about this field |
| Qualifiers | M, F | The list of possible values for this field |
| End Result | | This field will have the value “M” or “F” |
| dob | | |
| Data type | RandomDate | Generate a Date |
| Format | format,start,end | It will have a given format, with a value between start and end |
| Qualifiers | MM/dd/yyyy | It will have a “MM/dd/yyyy” format |
|  | 01/01/1970 | Its value will be including or after -1/01/1970 |
|  | 12/31/2020 | Its value will be before or including 12/31/2020 |
| End Result | | It will generate a random Date between 01/01/1970 and 12/31/2020 |
| zipcode | | |
| Data type | EnumInt | Generate an Integer |
| Format | nullable | The field could possible have a null value(missing data) |
| Qualifiers | 0.5 | The percentage chance(50%) that the field will have a null value |
|  | 11111,22222,33333,  44444,55555 | The different possible value the zipcode could have |
| End Result | | The field will be either of these values: 11111,22222,33333,  44444,55555, with a 50% chance of being empty |
| balance | | |
| Data type | RandomMoney | Generate a money value |
| Format | min,max | A minimum andmaximum values will be specified |
| Qualifiers | -100 | Minimum value |
|  | 100 | Maximum value |
| End Result | | This field will have a value between -100.00 and 100.00 |
| created | | |
| Data type | RandomDate | Generate a Date |
| Format | now | Insert the current date as this value |
| End Result | | The field’s value will be the current Date specified in the default(yyy-MM-dd) format |

Here are the configuration properties specified in our config.properties file:



The configuration properties tell us that:

1. The input spreadsheet is located at C:\\home\\schnarbies\\input\\example.xlsx
2. We will be creating 1 file with 20 records in it
3. We will only be using 1 thread to create the records
4. The output file will be located in the folder C:\\home\\schnarbies\\output\\example\\
5. The output file will start with the word “PEOPLE”
6. The output file will be of type JSON, and the data will be in json format

After running the program, we see our output file here:

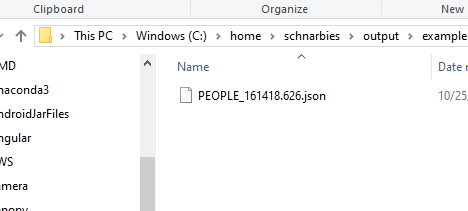


Figure - example output file

And in the file, we see 20 json records:

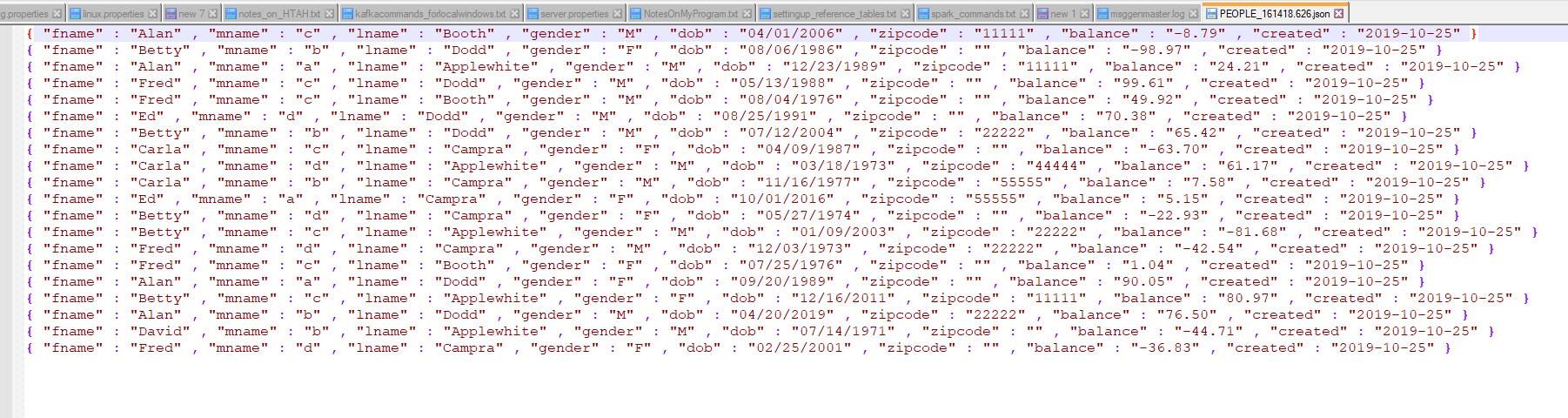


Figure - example output file json

Notice that half of the zipcode fields have the value “”

Also, notice that the mname fields are lowercased.

If we simply change the field **mode4.outputfile.type** to be CSV and rerun, now we get CSV data:

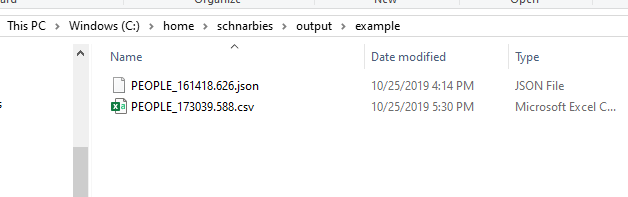


Figure - csv file generated

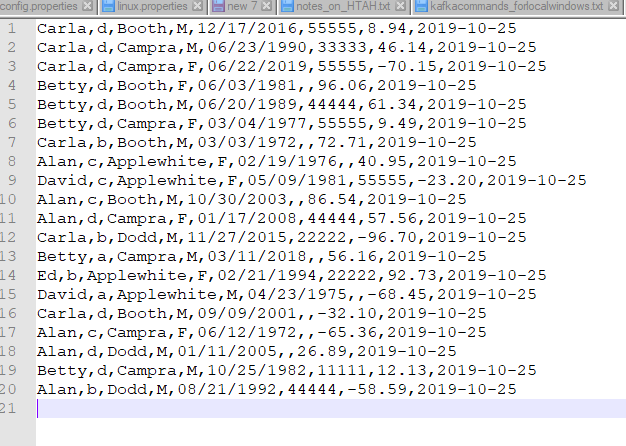


Figure - csv file data

# More in-depth with the minor modes

Mode 5 and 6: Hive SQL generation

If the data that you are generating is supposed to go into a Hive database table, then this functionality will auto-generate a script for creating the Hive table. The options when generating this table are:

1. Is the underlying data CSV or JSON data
2. Is it a managed table or an external table
3. Should all of the fields be represented in the table as strings(raw data), or should each field’s data type map to the data type specified in the input spreadsheet.

The main difference between mode 5 and 6 is that mode 5 performs all the actions of mode 4, and then will generate the Hive script for the data that is created. Mode 6, uses the same input excel spreadsheet, but only generates the Hive script; it does not generate data records.

Examples

Example 1:

Using the same input file examples.xlsx from the Mode 4 example before AND the same mode4.outputfile.type configuration(CSV), let’s look at the following configuration properties for running this feature in mode 6(Hive sql generation only):

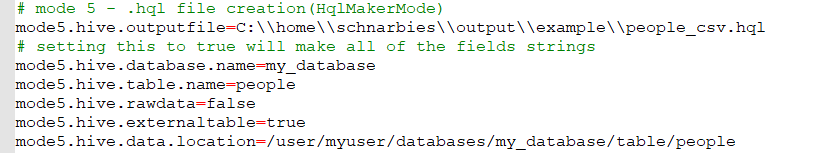


Figure - configuration for Hive sql generation

These configuration properties tell us:

1. The Hive script will be located at C:\\home\\schnarbies\\output\\example\\people\_csv.hql
2. The underlying data is of type CSV; the default Serializer/Deserializer specified will be org.apache.hadoop.hive.serde2.OpenCSVSerde
3. The name of the database is my\_database
4. The name of the table is people
5. The program will map the fields to their corresponding Hive data types
6. This will be an external table
7. The hdfs location of the data will be /user/myuser/databases/my\_database/table/people

The output file is shown below:

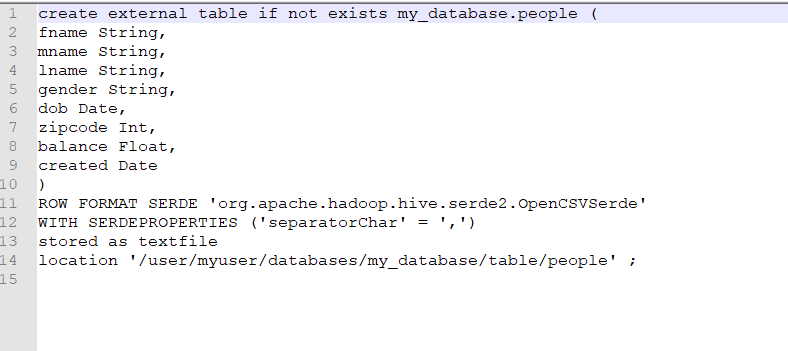


Figure - people\_csv.hql

Example 2:

Let’s change two fields:

mode4.outputfile.type=JSON

mode5.hive.outputfile=C:\\home\\schnarbies\\output\\example\\people\_json.hql

mode5.hive.externaltable=false

So, this tells us:

1. The underlying data is json formatted; the default Serializer/Deserialzer specified will be org.apache.hive.hcatalog.data.JsonSerDe
2. The new output file name will be people\_json.hql
3. This is NOT an external table, it is a managed table

The resulting HQL file is shown below:

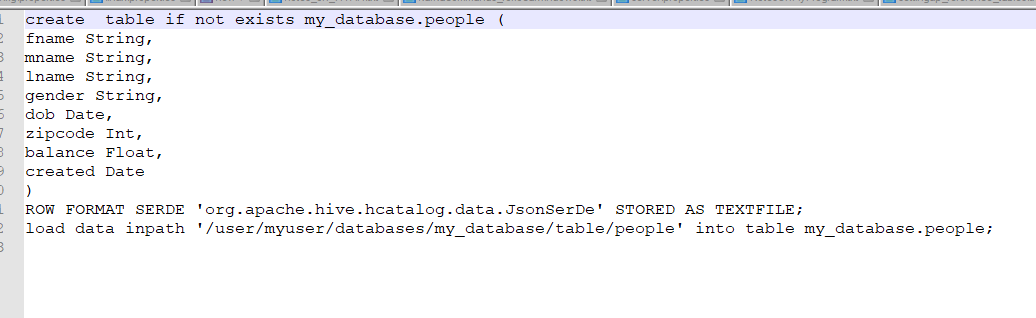


Figure - managed json table

Notice that there is a “load data” statement for this .hql script

Mode 7: Primary Key Functionality

<This feature is scheduled to be enhanced sometime in the near future>

Currently, the functionality of Mode 4 does not currently have an easy way to generate unique primary keys for each of the records made. The main purpose of this feature is to add a unique primary key to these records, without needing the individual threads to share mutable data structures, which can cause race conditions. In order to achieve this, we need to run the program twice:

1. Run the program in mode 1 to generate a file with the primary keys in it. If you want X records, you want to specify for this mode to generate X primary keys.
2. Run the program in mode 7 to generate your X records of data; the program will then substitute each of the primary keys from the first step for the first field in each of these records.

See Figure 13 in [Appendix 1](#_Appendices) for a flow of this feature.

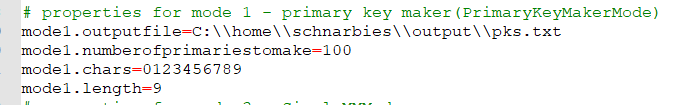
Examples

Example1:

We want to generate a list of 100 members, and we want to give each member a unique **suscriberid** number.

**Step 1 – run in mode 1**

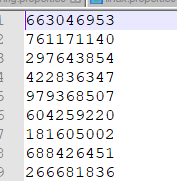
Here is the configuration for mode 1:



So this tells us:

1. Make 100 primary keys
2. Each primary key will be 9 digits long
3. Each primary key will contain any of the values “0123456789”
4. These primary keys will be stored in a file pks.txt

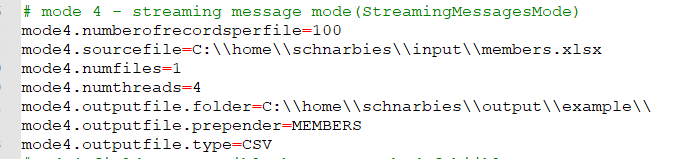
After running in mode 1, here is a sample of our pks.txt file:



**Step 2 – run in mode 2**

Keeping our settings for mode 1 the same, here is out configuration for mode 7:

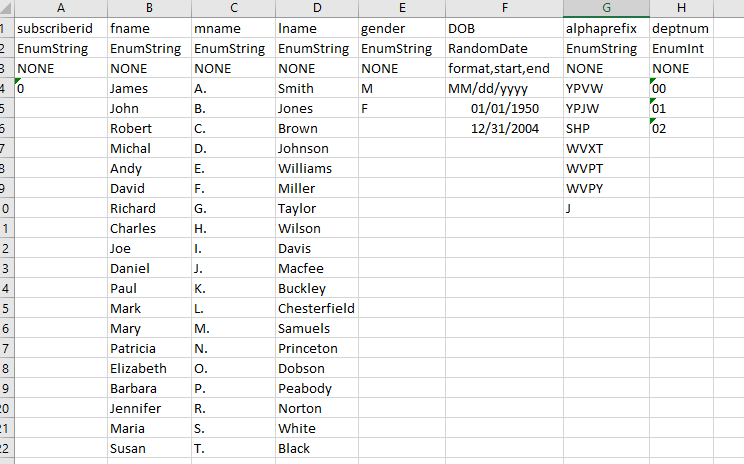
(note: there is no actual mode 7 configuration properties, the mode 4 configuration is used)

****

This tells us:

1. Out input spreadsheet is members.xlsx
2. We are making 100 member records
3. All of our records will be located in 1 file(this is a requirement of mode 7)
4. We will be using 4 threads to generate our data
5. Our output file will go in C:\\home\\schnarbies\\output\\example\\
6. Our output file will start with “MEMBERS”
7. Out data will be in csv format

Out input spreadsheet has been configured as such:

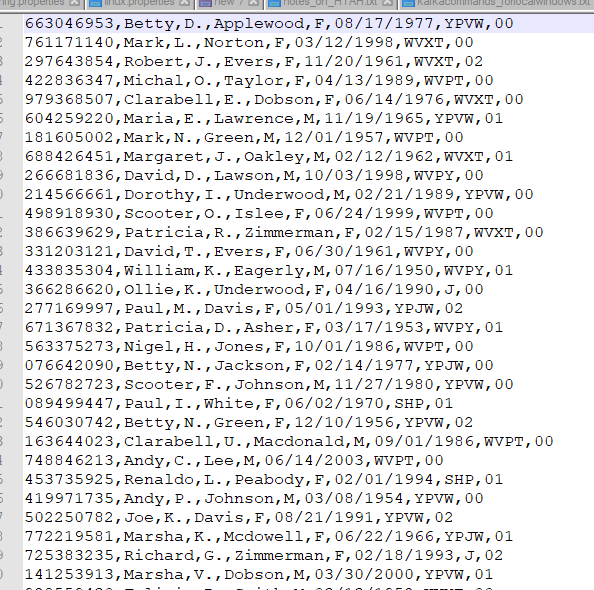


So, our members will all have a subscriberid, a first name, a middle initial, a last name, a gender, a date-of-birth, an alphaprefix, and a deptnum

**Note** that for the **subscriberid**, we are simply giving each record a value of **zero**, since this field’s value will be replaced anyway with the keys from **pks.txt**

After running, here is a sample of our “MEMBERS” data :

Note that the subscriberid field has been replaced with the primary keys



Mode 8: Kafka Topic functionality

This functionality assumes that the user is familiar with how to set up and use Kafka. We do include the kafka commands that we used to start zookeeper/broker/consumer on our local(Windows) environment.

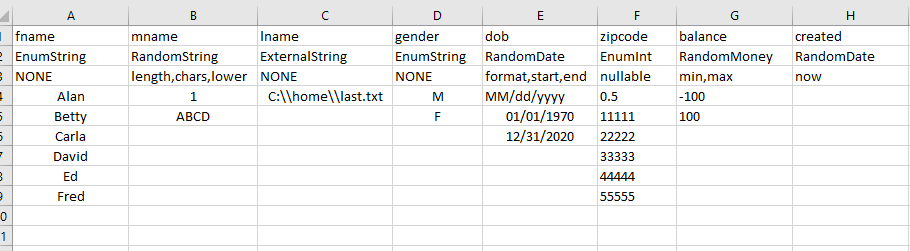
The version of Kafka we are using = 2.12-2.0.0

The version of Zookeeper that we are using = 3.3.6

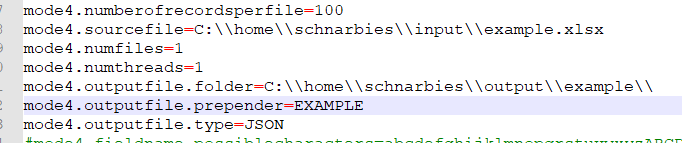
The purpose of this functionality is to simulate a streaming message system using our generated records. After each record is generated, the program will push it into a Kafka topic, with a configurable delay between each message push.

Examples

To demonstrate how to use this feature, we are going to use the same example spreadsheet that was used in the mode 4 example. A copy is below:

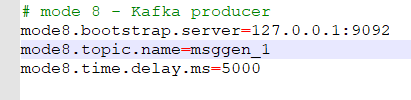


Our mode 4 configuration is shown below:



We are generating JSON for this example, just so that it is more noticeable in the kafka consumer command window.

Our mode 8 configuration properties are shown below:



This tells us that:

1. Our broker server is connected to port 9092
2. Our topic name is msggen\_1
3. There will be a 5 second delay between messages

Here are the commands that we used to start zookeeper, the broker, make the topic, and start the consumer:

# start zookeeper in one window

C:\kafka\_2.12-2.0.0\bin\windows\zookeeper-server-start.bat C:\kafka\_2.12-2.0.0\config\zookeeper.properties

# start a single broker(default port 9092 I believe) in another window

C:\kafka\_2.12-2.0.0\bin\windows\kafka-server-start.bat C:\kafka\_2.12-2.0.0\config\server.properties

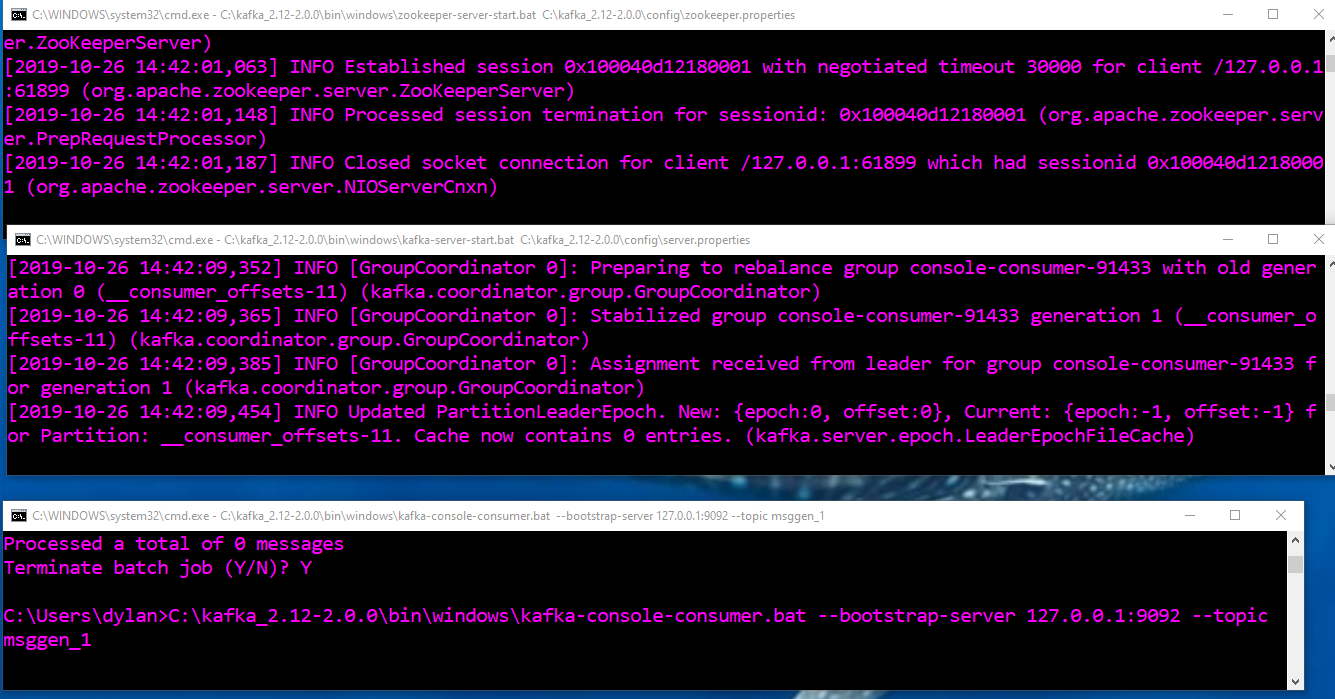
# create a new topic in a third window

C:\kafka\_2.12-2.0.0\bin\windows\kafka-topics.bat --zookeeper 127.0.0.1:2181 --topic msggen\_1 --create --partitions 1 --replication-factor 1

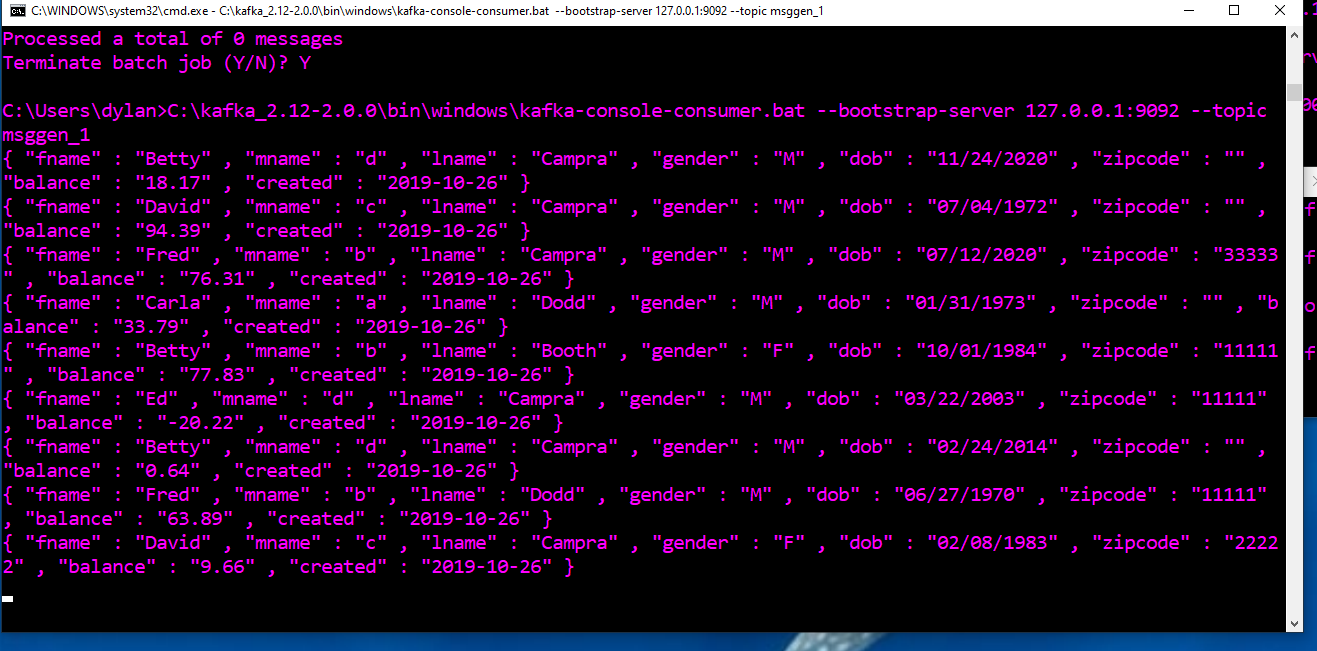
# create the consumer in a third window

C:\kafka\_2.12-2.0.0\bin\windows\kafka-console-consumer.bat --bootstrap-server 127.0.0.1:9092 --topic msggen\_

Below we have our zookeeper, broker and kafka consumer windows(the consumer is at the bottom)



After running in mode 8, we can now see messages populating slowly in the consumer window:



Additional Kafka configuration information:

server.properties configuration:

broker.id=0

listeners=PLAINTEXT://:9092

zookeeper.connect=localhost:2181

log.dirs=/tmp/kafka-logs

zookeeper.properties configuration:

dataDir=C:/kafka\_2.12-2.0.0/data/zookeeper

clientPort=2181

# Building and Deployment

For non-developers, the latest archive jar file of this application can be found in the repository at:

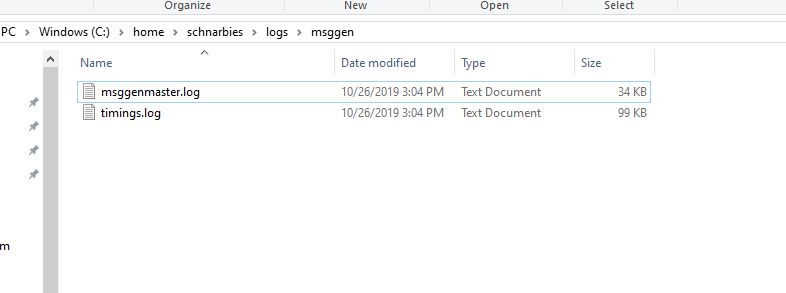
/artifact/msggen/jar

For developers, the artifact can be built using either maven or SBT. The application has both a pom.xml and build.sbt files.

# Log Files

The user specifies the root log file location with the **root.log.path** common property.

The two main log files are shown below:



**mssgenmaster.log** – this is the main log. All system messages and error messages are logged to this file. If there are issues with your input spreadsheet, this log file will have that information in it.

**Timings.log –** this log file is primarily used to monitor the programs run efficiency. It is mostly for development purposes.

There may be other log files that get generated. These are mostly for some of the minor features.

See Appendix 2 for a sample log file for the Mode 8 example.

# In the Works: Future Enhancements

The following is a List of minor and major features that I would like to add to this program sometime in the near future, stay tuned!

Minor Features

***TODO*** *- @1 adjust the Actors to pass in the current file #, mode4.numberofrecordsperfile property****TODO*** *- @2 add two new formats: baseValue and incrementValue so we can use @1 to have incrementing fields, primary keys****TODO*** *- add header functionality to the program when generating data and the Hive scripts****TODO*** *- add functionality for JSON to make the files valid JSON****TODO*** *- check the code around the mode4.outputfile.folder property, and have it add either a "\\" or "/" to the end if it is not given****TODO*** *- make line 89 of FileIO make the output folder if it doesn't exist!, or at least make the validator check/throw error if its not found****TODO*** *- make the data types case insensitive, make the formats case insensitive also****TODO*** *- make the format fields trimmed for whitespace before splitting****TODO*** *- extend mode 7 so that the PK array can be accessed from the template, so that the PKs can be added during the act of record generation, so that we don't need to single file requirement****TODO*** *- new format rules pre and post, that allow you to specify a constant value to prepend to the front of each value or add to the end of each value****TODO*** *- add functionality in HiveTableCreator to determine whether the data to load into a managed table is from HDFS or the local file system****TODO*** *- @3 SQL generation functionality for a MySQL database table****TODO*** *- @4 functionality to push records to MySQL table(from @3)****TODO*** *- @3 and @4 for MongoDB****TODO*** *- @3 and @4 for Cassandra****TODO*** *- @5 Array format feature*

MAJOR FEATURES  
***TODO*** *- new Dependent data types and dependent fields map; wrap Record make threads in a for loop to allow us to generate fields based on other fields(state/zip/county example)****TODO*** *- new mode: recursive feature that allows us to Take a parent root Java object, recursively parse through all of its child fields/objects, to dynamically generate an input spreadsheet; user will then fill in the format/qualifier info for this sheet, and rerun the program in Mode 4, dependent on @5*

# Appendices

Appendix 1 – Flow Diagrams

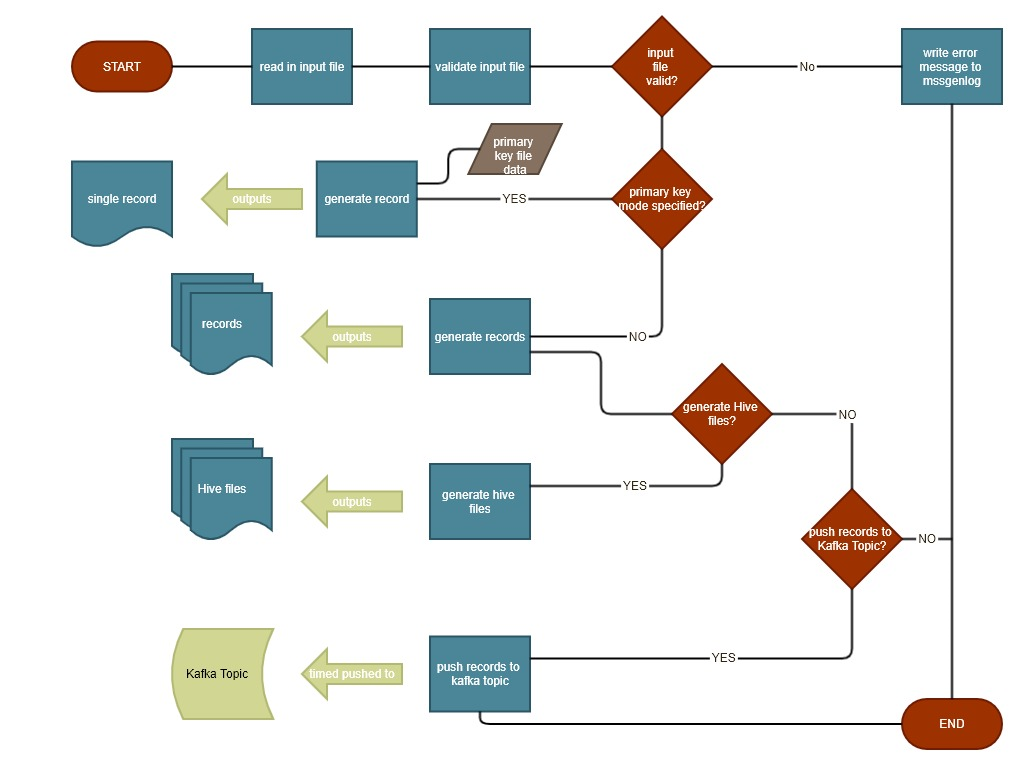


Figure - overall flow diagram for modes 4,5,7, and 8

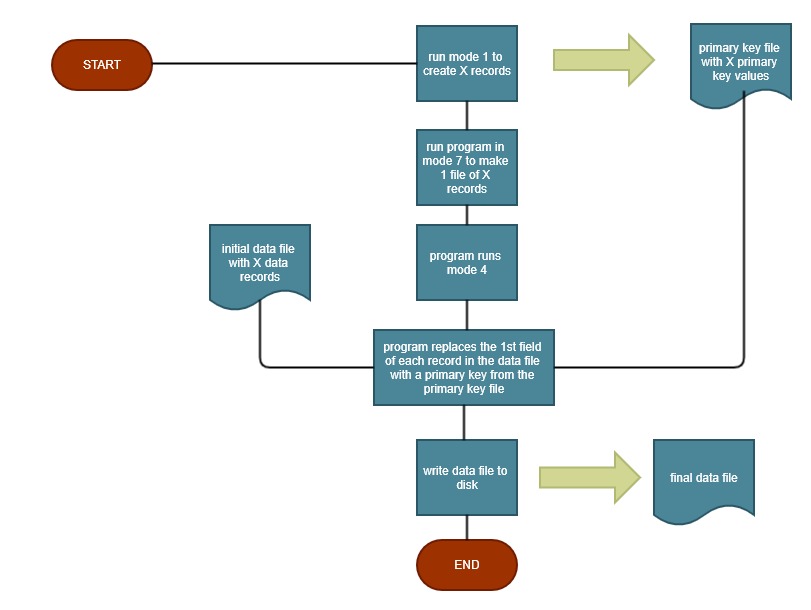


Figure - How mode 7 works

Appendix 2 – msggen Log File sample

The following log files sample comes from running the Mode 8 example above:

10/26/19 15:40:49.813 DEBUG Mode = 8

10/26/19 15:40:49.848 DEBUG inside StreamingMessagesMode

10/26/19 15:40:49.853 DEBUG entering readInSpreadsheet() method

10/26/19 15:40:49.854 DEBUG attempting to open workbook

10/26/19 15:40:50.586 DEBUG opened workbook

10/26/19 15:40:50.588 DEBUG reading in the Column names

10/26/19 15:40:50.620 DEBUG reading in the Data Types of each Column

10/26/19 15:40:50.624 DEBUG reading in the Data Formatting Information for each Column

10/26/19 15:40:50.627 DEBUG reading in the Data Value Information for each Column

10/26/19 15:40:51.600 DEBUG entering ExcelDataSheetValidator.validate() method, validating the input template

10/26/19 15:40:51.600 DEBUG header rows validated

10/26/19 15:40:51.679 DEBUG data types validated

10/26/19 15:40:51.686 DEBUG index 0 , EnumString format NONE , is valid

10/26/19 15:40:51.689 DEBUG index 1 , RandomString format length,chars,lower , is valid

10/26/19 15:40:51.690 DEBUG index 2 , ExternalString format NONE , is valid

10/26/19 15:40:51.690 DEBUG index 3 , EnumString format NONE , is valid

10/26/19 15:40:51.693 DEBUG index 4 , RandomDate format format,start,end , is valid

10/26/19 15:40:51.694 DEBUG index 5 , EnumInt format nullable , is valid

10/26/19 15:40:51.695 DEBUG index 6 , RandomMoney format min,max , is valid

10/26/19 15:40:51.696 DEBUG index 7 , RandomDate format now , is valid

10/26/19 15:40:51.697 DEBUG data formats validated

10/26/19 15:40:51.712 DEBUG all of the qualifiers for field : fname are valid

10/26/19 15:40:51.716 DEBUG all of the qualifiers for field : mname are valid

10/26/19 15:40:51.719 DEBUG all of the qualifiers for field : lname are valid

10/26/19 15:40:51.719 DEBUG all of the qualifiers for field : gender are valid

10/26/19 15:40:51.742 DEBUG all of the qualifiers for field : dob are valid

10/26/19 15:40:51.746 DEBUG all of the qualifiers for field : zipcode are valid

10/26/19 15:40:51.751 DEBUG all of the qualifiers for field : balance are valid

10/26/19 15:40:51.753 DEBUG all of the qualifiers for field : created are valid

10/26/19 15:40:51.754 DEBUG data values validated

10/26/19 15:40:51.757 DEBUG template validated

10/26/19 15:40:51.758 DEBUG entering readInSpreadsheet() method

10/26/19 15:40:51.775 DEBUG reading in .txt file

10/26/19 15:40:51.781 DEBUG inside RecordMakerController.generateRecords main method

10/26/19 15:40:51.782 DEBUG initializing the actor system

10/26/19 15:40:52.581 DEBUG creating the primary record making controller

10/26/19 15:40:52.586 DEBUG sending a StartMessage to the primary record making controller

10/26/19 15:40:52.587 DEBUG

10/26/19 15:40:52.588 DEBUG numTheads = 1

10/26/19 15:40:52.589 DEBUG numFiles = 1

10/26/19 15:40:52.590 DEBUG polling the ArrayBlockingQueue

10/26/19 15:40:52.591 DEBUG numRecordsPerFile = 100

10/26/19 15:40:52.596 DEBUG

10/26/19 15:40:52.609 DEBUG

10/26/19 15:40:52.609 DEBUG inside initiateRecordMaking

10/26/19 15:40:52.615 DEBUG made 1 records

10/26/19 15:40:52.618 DEBUG

10/26/19 15:40:52.681 DEBUG inside incrementCheckAndCount, records made = 1

10/26/19 15:40:52.683 DEBUG sending another MakeRecordMessage(1)

10/26/19 15:40:52.691 DEBUG inside incrementCheckAndCount, records made = 2

10/26/19 15:40:52.692 DEBUG sending another MakeRecordMessage(2)

10/26/19 15:40:52.713 DEBUG inside incrementCheckAndCount, records made = 3

10/26/19 15:40:52.713 DEBUG sending another MakeRecordMessage(3)

10/26/19 15:40:52.723 DEBUG inside incrementCheckAndCount, records made = 4

10/26/19 15:40:52.725 DEBUG sending another MakeRecordMessage(4)

10/26/19 15:40:52.734 DEBUG inside incrementCheckAndCount, records made = 5

10/26/19 15:40:52.735 DEBUG sending another MakeRecordMessage(5)

10/26/19 15:40:52.743 DEBUG inside incrementCheckAndCount, records made = 6

10/26/19 15:40:52.744 DEBUG sending another MakeRecordMessage(6)

10/26/19 15:40:52.755 DEBUG inside incrementCheckAndCount, records made = 7

10/26/19 15:40:52.756 DEBUG sending another MakeRecordMessage(7)

10/26/19 15:40:52.762 DEBUG inside incrementCheckAndCount, records made = 8

10/26/19 15:40:52.769 DEBUG sending another MakeRecordMessage(8)

10/26/19 15:40:52.785 DEBUG inside incrementCheckAndCount, records made = 9

10/26/19 15:40:52.787 DEBUG sending another MakeRecordMessage(9)

10/26/19 15:40:52.792 DEBUG inside incrementCheckAndCount, records made = 10

10/26/19 15:40:52.793 DEBUG sending another MakeRecordMessage(10)

10/26/19 15:40:52.799 DEBUG inside incrementCheckAndCount, records made = 11

10/26/19 15:40:52.800 DEBUG sending another MakeRecordMessage(11)

10/26/19 15:40:52.806 DEBUG inside incrementCheckAndCount, records made = 12

10/26/19 15:40:52.810 DEBUG sending another MakeRecordMessage(12)

10/26/19 15:40:52.815 DEBUG inside incrementCheckAndCount, records made = 13

10/26/19 15:40:52.816 DEBUG sending another MakeRecordMessage(13)

10/26/19 15:40:52.822 DEBUG inside incrementCheckAndCount, records made = 14

10/26/19 15:40:52.823 DEBUG sending another MakeRecordMessage(14)

10/26/19 15:40:52.829 DEBUG inside incrementCheckAndCount, records made = 15

10/26/19 15:40:52.834 DEBUG sending another MakeRecordMessage(15)

10/26/19 15:40:52.840 DEBUG inside incrementCheckAndCount, records made = 16

10/26/19 15:40:52.841 DEBUG sending another MakeRecordMessage(16)

10/26/19 15:40:52.848 DEBUG inside incrementCheckAndCount, records made = 17

10/26/19 15:40:52.849 DEBUG sending another MakeRecordMessage(17)

10/26/19 15:40:52.857 DEBUG inside incrementCheckAndCount, records made = 18

10/26/19 15:40:52.858 DEBUG sending another MakeRecordMessage(18)

10/26/19 15:40:52.863 DEBUG inside incrementCheckAndCount, records made = 19

10/26/19 15:40:52.864 DEBUG sending another MakeRecordMessage(19)

10/26/19 15:40:52.872 DEBUG inside incrementCheckAndCount, records made = 20

10/26/19 15:40:52.873 DEBUG sending another MakeRecordMessage(20)

10/26/19 15:40:52.884 DEBUG inside incrementCheckAndCount, records made = 21

10/26/19 15:40:52.885 DEBUG sending another MakeRecordMessage(21)

10/26/19 15:40:52.895 DEBUG inside incrementCheckAndCount, records made = 22

10/26/19 15:40:52.896 DEBUG sending another MakeRecordMessage(22)

10/26/19 15:40:52.906 DEBUG inside incrementCheckAndCount, records made = 23

10/26/19 15:40:52.912 DEBUG sending another MakeRecordMessage(23)

10/26/19 15:40:52.919 DEBUG inside incrementCheckAndCount, records made = 24

10/26/19 15:40:52.920 DEBUG sending another MakeRecordMessage(24)

10/26/19 15:40:52.926 DEBUG inside incrementCheckAndCount, records made = 25

10/26/19 15:40:52.927 DEBUG sending another MakeRecordMessage(25)

10/26/19 15:40:52.931 DEBUG inside incrementCheckAndCount, records made = 26

10/26/19 15:40:52.932 DEBUG sending another MakeRecordMessage(26)

10/26/19 15:40:52.938 DEBUG inside incrementCheckAndCount, records made = 27

10/26/19 15:40:52.939 DEBUG sending another MakeRecordMessage(27)

10/26/19 15:40:52.944 DEBUG inside incrementCheckAndCount, records made = 28

10/26/19 15:40:52.945 DEBUG sending another MakeRecordMessage(28)

10/26/19 15:40:52.951 DEBUG inside incrementCheckAndCount, records made = 29

10/26/19 15:40:52.952 DEBUG sending another MakeRecordMessage(29)

10/26/19 15:40:52.957 DEBUG inside incrementCheckAndCount, records made = 30

10/26/19 15:40:52.958 DEBUG sending another MakeRecordMessage(30)

10/26/19 15:40:52.965 DEBUG inside incrementCheckAndCount, records made = 31

10/26/19 15:40:52.967 DEBUG sending another MakeRecordMessage(31)

10/26/19 15:40:52.971 DEBUG inside incrementCheckAndCount, records made = 32

10/26/19 15:40:52.975 DEBUG sending another MakeRecordMessage(32)

10/26/19 15:40:52.978 DEBUG inside incrementCheckAndCount, records made = 33

10/26/19 15:40:52.979 DEBUG sending another MakeRecordMessage(33)

10/26/19 15:40:52.982 DEBUG inside incrementCheckAndCount, records made = 34

10/26/19 15:40:52.987 DEBUG sending another MakeRecordMessage(34)

10/26/19 15:40:52.999 DEBUG inside incrementCheckAndCount, records made = 35

10/26/19 15:40:53.003 DEBUG sending another MakeRecordMessage(35)

10/26/19 15:40:53.008 DEBUG inside incrementCheckAndCount, records made = 36

10/26/19 15:40:53.009 DEBUG sending another MakeRecordMessage(36)

10/26/19 15:40:53.014 DEBUG inside incrementCheckAndCount, records made = 37

10/26/19 15:40:53.016 DEBUG sending another MakeRecordMessage(37)

10/26/19 15:40:53.023 DEBUG inside incrementCheckAndCount, records made = 38

10/26/19 15:40:53.024 DEBUG sending another MakeRecordMessage(38)

10/26/19 15:40:53.027 DEBUG inside incrementCheckAndCount, records made = 39

10/26/19 15:40:53.031 DEBUG sending another MakeRecordMessage(39)

10/26/19 15:40:53.036 DEBUG inside incrementCheckAndCount, records made = 40

10/26/19 15:40:53.037 DEBUG sending another MakeRecordMessage(40)

10/26/19 15:40:53.043 DEBUG inside incrementCheckAndCount, records made = 41

10/26/19 15:40:53.044 DEBUG sending another MakeRecordMessage(41)

10/26/19 15:40:53.048 DEBUG inside incrementCheckAndCount, records made = 42

10/26/19 15:40:53.049 DEBUG sending another MakeRecordMessage(42)

10/26/19 15:40:53.056 DEBUG inside incrementCheckAndCount, records made = 43

10/26/19 15:40:53.057 DEBUG sending another MakeRecordMessage(43)

10/26/19 15:40:53.061 DEBUG inside incrementCheckAndCount, records made = 44

10/26/19 15:40:53.062 DEBUG sending another MakeRecordMessage(44)

10/26/19 15:40:53.069 DEBUG inside incrementCheckAndCount, records made = 45

10/26/19 15:40:53.071 DEBUG sending another MakeRecordMessage(45)

10/26/19 15:40:53.075 DEBUG inside incrementCheckAndCount, records made = 46

10/26/19 15:40:53.076 DEBUG sending another MakeRecordMessage(46)

10/26/19 15:40:53.083 DEBUG inside incrementCheckAndCount, records made = 47

10/26/19 15:40:53.085 DEBUG sending another MakeRecordMessage(47)

10/26/19 15:40:53.089 DEBUG inside incrementCheckAndCount, records made = 48

10/26/19 15:40:53.093 DEBUG sending another MakeRecordMessage(48)

10/26/19 15:40:53.096 DEBUG inside incrementCheckAndCount, records made = 49

10/26/19 15:40:53.097 DEBUG sending another MakeRecordMessage(49)

10/26/19 15:40:53.101 DEBUG inside incrementCheckAndCount, records made = 50

10/26/19 15:40:53.105 DEBUG sending another MakeRecordMessage(50)

10/26/19 15:40:53.109 DEBUG inside incrementCheckAndCount, records made = 51

10/26/19 15:40:53.110 DEBUG sending another MakeRecordMessage(51)

10/26/19 15:40:53.116 DEBUG inside incrementCheckAndCount, records made = 52

10/26/19 15:40:53.118 DEBUG sending another MakeRecordMessage(52)

10/26/19 15:40:53.122 DEBUG inside incrementCheckAndCount, records made = 53

10/26/19 15:40:53.123 DEBUG sending another MakeRecordMessage(53)

10/26/19 15:40:53.128 DEBUG inside incrementCheckAndCount, records made = 54

10/26/19 15:40:53.129 DEBUG sending another MakeRecordMessage(54)

10/26/19 15:40:53.134 DEBUG inside incrementCheckAndCount, records made = 55

10/26/19 15:40:53.135 DEBUG sending another MakeRecordMessage(55)

10/26/19 15:40:53.141 DEBUG inside incrementCheckAndCount, records made = 56

10/26/19 15:40:53.141 DEBUG sending another MakeRecordMessage(56)

10/26/19 15:40:53.146 DEBUG inside incrementCheckAndCount, records made = 57

10/26/19 15:40:53.147 DEBUG sending another MakeRecordMessage(57)

10/26/19 15:40:53.152 DEBUG inside incrementCheckAndCount, records made = 58

10/26/19 15:40:53.154 DEBUG sending another MakeRecordMessage(58)

10/26/19 15:40:53.158 DEBUG inside incrementCheckAndCount, records made = 59

10/26/19 15:40:53.159 DEBUG sending another MakeRecordMessage(59)

10/26/19 15:40:53.164 DEBUG inside incrementCheckAndCount, records made = 60

10/26/19 15:40:53.165 DEBUG sending another MakeRecordMessage(60)

10/26/19 15:40:53.179 DEBUG inside incrementCheckAndCount, records made = 61

10/26/19 15:40:53.180 DEBUG sending another MakeRecordMessage(61)

10/26/19 15:40:53.189 DEBUG inside incrementCheckAndCount, records made = 62

10/26/19 15:40:53.189 DEBUG sending another MakeRecordMessage(62)

10/26/19 15:40:53.193 DEBUG inside incrementCheckAndCount, records made = 63

10/26/19 15:40:53.194 DEBUG sending another MakeRecordMessage(63)

10/26/19 15:40:53.200 DEBUG inside incrementCheckAndCount, records made = 64

10/26/19 15:40:53.202 DEBUG sending another MakeRecordMessage(64)

10/26/19 15:40:53.206 DEBUG inside incrementCheckAndCount, records made = 65

10/26/19 15:40:53.209 DEBUG sending another MakeRecordMessage(65)

10/26/19 15:40:53.213 DEBUG inside incrementCheckAndCount, records made = 66

10/26/19 15:40:53.214 DEBUG sending another MakeRecordMessage(66)

10/26/19 15:40:53.218 DEBUG inside incrementCheckAndCount, records made = 67

10/26/19 15:40:53.219 DEBUG sending another MakeRecordMessage(67)

10/26/19 15:40:53.226 DEBUG inside incrementCheckAndCount, records made = 68

10/26/19 15:40:53.227 DEBUG sending another MakeRecordMessage(68)

10/26/19 15:40:53.230 DEBUG inside incrementCheckAndCount, records made = 69

10/26/19 15:40:53.231 DEBUG sending another MakeRecordMessage(69)

10/26/19 15:40:53.237 DEBUG inside incrementCheckAndCount, records made = 70

10/26/19 15:40:53.238 DEBUG sending another MakeRecordMessage(70)

10/26/19 15:40:53.240 DEBUG inside incrementCheckAndCount, records made = 71

10/26/19 15:40:53.241 DEBUG sending another MakeRecordMessage(71)

10/26/19 15:40:53.244 DEBUG inside incrementCheckAndCount, records made = 72

10/26/19 15:40:53.248 DEBUG sending another MakeRecordMessage(72)

10/26/19 15:40:53.252 DEBUG inside incrementCheckAndCount, records made = 73

10/26/19 15:40:53.252 DEBUG sending another MakeRecordMessage(73)

10/26/19 15:40:53.255 DEBUG inside incrementCheckAndCount, records made = 74

10/26/19 15:40:53.256 DEBUG sending another MakeRecordMessage(74)

10/26/19 15:40:53.262 DEBUG inside incrementCheckAndCount, records made = 75

10/26/19 15:40:53.263 DEBUG sending another MakeRecordMessage(75)

10/26/19 15:40:53.267 DEBUG inside incrementCheckAndCount, records made = 76

10/26/19 15:40:53.268 DEBUG sending another MakeRecordMessage(76)

10/26/19 15:40:53.274 DEBUG inside incrementCheckAndCount, records made = 77

10/26/19 15:40:53.275 DEBUG sending another MakeRecordMessage(77)

10/26/19 15:40:53.278 DEBUG inside incrementCheckAndCount, records made = 78

10/26/19 15:40:53.278 DEBUG sending another MakeRecordMessage(78)

10/26/19 15:40:53.284 DEBUG inside incrementCheckAndCount, records made = 79

10/26/19 15:40:53.285 DEBUG sending another MakeRecordMessage(79)

10/26/19 15:40:53.289 DEBUG inside incrementCheckAndCount, records made = 80

10/26/19 15:40:53.290 DEBUG sending another MakeRecordMessage(80)

10/26/19 15:40:53.295 DEBUG inside incrementCheckAndCount, records made = 81

10/26/19 15:40:53.296 DEBUG sending another MakeRecordMessage(81)

10/26/19 15:40:53.299 DEBUG inside incrementCheckAndCount, records made = 82

10/26/19 15:40:53.300 DEBUG sending another MakeRecordMessage(82)

10/26/19 15:40:53.305 DEBUG inside incrementCheckAndCount, records made = 83

10/26/19 15:40:53.307 DEBUG sending another MakeRecordMessage(83)

10/26/19 15:40:53.311 DEBUG inside incrementCheckAndCount, records made = 84

10/26/19 15:40:53.312 DEBUG sending another MakeRecordMessage(84)

10/26/19 15:40:53.318 DEBUG inside incrementCheckAndCount, records made = 85

10/26/19 15:40:53.318 DEBUG sending another MakeRecordMessage(85)

10/26/19 15:40:53.322 DEBUG inside incrementCheckAndCount, records made = 86

10/26/19 15:40:53.323 DEBUG sending another MakeRecordMessage(86)

10/26/19 15:40:53.328 DEBUG inside incrementCheckAndCount, records made = 87

10/26/19 15:40:53.329 DEBUG sending another MakeRecordMessage(87)

10/26/19 15:40:53.332 DEBUG inside incrementCheckAndCount, records made = 88

10/26/19 15:40:53.333 DEBUG sending another MakeRecordMessage(88)

10/26/19 15:40:53.337 DEBUG inside incrementCheckAndCount, records made = 89

10/26/19 15:40:53.340 DEBUG sending another MakeRecordMessage(89)

10/26/19 15:40:53.344 DEBUG inside incrementCheckAndCount, records made = 90

10/26/19 15:40:53.345 DEBUG sending another MakeRecordMessage(90)

10/26/19 15:40:53.348 DEBUG inside incrementCheckAndCount, records made = 91

10/26/19 15:40:53.348 DEBUG sending another MakeRecordMessage(91)

10/26/19 15:40:53.355 DEBUG inside incrementCheckAndCount, records made = 92

10/26/19 15:40:53.356 DEBUG sending another MakeRecordMessage(92)

10/26/19 15:40:53.358 DEBUG inside incrementCheckAndCount, records made = 93

10/26/19 15:40:53.359 DEBUG sending another MakeRecordMessage(93)

10/26/19 15:40:53.362 DEBUG inside incrementCheckAndCount, records made = 94

10/26/19 15:40:53.367 DEBUG sending another MakeRecordMessage(94)

10/26/19 15:40:53.370 DEBUG inside incrementCheckAndCount, records made = 95

10/26/19 15:40:53.371 DEBUG sending another MakeRecordMessage(95)

10/26/19 15:40:53.377 DEBUG inside incrementCheckAndCount, records made = 96

10/26/19 15:40:53.377 DEBUG sending another MakeRecordMessage(96)

10/26/19 15:40:53.381 DEBUG inside incrementCheckAndCount, records made = 97

10/26/19 15:40:53.382 DEBUG sending another MakeRecordMessage(97)

10/26/19 15:40:53.384 DEBUG inside incrementCheckAndCount, records made = 98

10/26/19 15:40:53.389 DEBUG sending another MakeRecordMessage(98)

10/26/19 15:40:53.392 DEBUG inside incrementCheckAndCount, records made = 99

10/26/19 15:40:53.392 DEBUG sending another MakeRecordMessage(99)

10/26/19 15:40:53.395 DEBUG inside incrementCheckAndCount, records made = 100

10/26/19 15:40:53.396 DEBUG initiating record copying

10/26/19 15:40:53.400 DEBUG

10/26/19 15:40:53.401 DEBUG inside initiateRecordCopying

10/26/19 15:40:53.406 DEBUG copied 1 records

10/26/19 15:40:53.407 DEBUG

10/26/19 15:40:53.409 DEBUG turning off record making phase

10/26/19 15:40:53.413 DEBUG inside incrementCopyAndCount, records copied = 1

10/26/19 15:40:53.414 DEBUG sending another CopyRecordMessage(1,0)

10/26/19 15:40:53.418 DEBUG inside incrementCopyAndCount, records copied = 2

10/26/19 15:40:53.418 DEBUG sending another CopyRecordMessage(2,0)

10/26/19 15:40:53.420 DEBUG inside incrementCopyAndCount, records copied = 3

10/26/19 15:40:53.421 DEBUG sending another CopyRecordMessage(3,0)

10/26/19 15:40:53.427 DEBUG inside incrementCopyAndCount, records copied = 4

10/26/19 15:40:53.428 DEBUG sending another CopyRecordMessage(4,0)

10/26/19 15:40:53.430 DEBUG inside incrementCopyAndCount, records copied = 5

10/26/19 15:40:53.431 DEBUG sending another CopyRecordMessage(5,0)

10/26/19 15:40:53.433 DEBUG inside incrementCopyAndCount, records copied = 6

10/26/19 15:40:53.434 DEBUG sending another CopyRecordMessage(6,0)

10/26/19 15:40:53.439 DEBUG inside incrementCopyAndCount, records copied = 7

10/26/19 15:40:53.440 DEBUG sending another CopyRecordMessage(7,0)

10/26/19 15:40:53.442 DEBUG inside incrementCopyAndCount, records copied = 8

10/26/19 15:40:53.443 DEBUG sending another CopyRecordMessage(8,0)

10/26/19 15:40:53.451 DEBUG inside incrementCopyAndCount, records copied = 9

10/26/19 15:40:53.451 DEBUG sending another CopyRecordMessage(9,0)

10/26/19 15:40:53.455 DEBUG inside incrementCopyAndCount, records copied = 10

10/26/19 15:40:53.456 DEBUG sending another CopyRecordMessage(10,0)

10/26/19 15:40:53.463 DEBUG inside incrementCopyAndCount, records copied = 11

10/26/19 15:40:53.463 DEBUG sending another CopyRecordMessage(11,0)

10/26/19 15:40:53.466 DEBUG inside incrementCopyAndCount, records copied = 12

10/26/19 15:40:53.469 DEBUG sending another CopyRecordMessage(12,0)

10/26/19 15:40:53.481 DEBUG inside incrementCopyAndCount, records copied = 13

10/26/19 15:40:53.484 DEBUG sending another CopyRecordMessage(13,0)

10/26/19 15:40:53.487 DEBUG inside incrementCopyAndCount, records copied = 14

10/26/19 15:40:53.489 DEBUG sending another CopyRecordMessage(14,0)

10/26/19 15:40:53.495 DEBUG inside incrementCopyAndCount, records copied = 15

10/26/19 15:40:53.496 DEBUG sending another CopyRecordMessage(15,0)

10/26/19 15:40:53.507 DEBUG inside incrementCopyAndCount, records copied = 16

10/26/19 15:40:53.508 DEBUG sending another CopyRecordMessage(16,0)

10/26/19 15:40:53.512 DEBUG inside incrementCopyAndCount, records copied = 17

10/26/19 15:40:53.513 DEBUG sending another CopyRecordMessage(17,0)

10/26/19 15:40:53.530 DEBUG inside incrementCopyAndCount, records copied = 18

10/26/19 15:40:53.531 DEBUG sending another CopyRecordMessage(18,0)

10/26/19 15:40:53.541 DEBUG inside incrementCopyAndCount, records copied = 19

10/26/19 15:40:53.544 DEBUG sending another CopyRecordMessage(19,0)

10/26/19 15:40:53.553 DEBUG inside incrementCopyAndCount, records copied = 20

10/26/19 15:40:53.554 DEBUG sending another CopyRecordMessage(20,0)

10/26/19 15:40:53.560 DEBUG inside incrementCopyAndCount, records copied = 21

10/26/19 15:40:53.561 DEBUG sending another CopyRecordMessage(21,0)

10/26/19 15:40:53.567 DEBUG inside incrementCopyAndCount, records copied = 22

10/26/19 15:40:53.570 DEBUG sending another CopyRecordMessage(22,0)

10/26/19 15:40:53.573 DEBUG inside incrementCopyAndCount, records copied = 23

10/26/19 15:40:53.574 DEBUG sending another CopyRecordMessage(23,0)

10/26/19 15:40:53.580 DEBUG inside incrementCopyAndCount, records copied = 24

10/26/19 15:40:53.581 DEBUG sending another CopyRecordMessage(24,0)

10/26/19 15:40:53.594 DEBUG inside incrementCopyAndCount, records copied = 25

10/26/19 15:40:53.595 DEBUG sending another CopyRecordMessage(25,0)

10/26/19 15:40:53.602 DEBUG inside incrementCopyAndCount, records copied = 26

10/26/19 15:40:53.603 DEBUG sending another CopyRecordMessage(26,0)

10/26/19 15:40:53.621 DEBUG inside incrementCopyAndCount, records copied = 27

10/26/19 15:40:53.622 DEBUG sending another CopyRecordMessage(27,0)

10/26/19 15:40:53.629 DEBUG inside incrementCopyAndCount, records copied = 28

10/26/19 15:40:53.630 DEBUG sending another CopyRecordMessage(28,0)

10/26/19 15:40:53.638 DEBUG inside incrementCopyAndCount, records copied = 29

10/26/19 15:40:53.640 DEBUG sending another CopyRecordMessage(29,0)

10/26/19 15:40:53.651 DEBUG inside incrementCopyAndCount, records copied = 30

10/26/19 15:40:53.656 DEBUG sending another CopyRecordMessage(30,0)

10/26/19 15:40:53.665 DEBUG inside incrementCopyAndCount, records copied = 31

10/26/19 15:40:53.669 DEBUG sending another CopyRecordMessage(31,0)

10/26/19 15:40:53.673 DEBUG inside incrementCopyAndCount, records copied = 32

10/26/19 15:40:53.679 DEBUG sending another CopyRecordMessage(32,0)

10/26/19 15:40:53.686 DEBUG inside incrementCopyAndCount, records copied = 33

10/26/19 15:40:53.687 DEBUG sending another CopyRecordMessage(33,0)

10/26/19 15:40:53.692 DEBUG inside incrementCopyAndCount, records copied = 34

10/26/19 15:40:53.693 DEBUG sending another CopyRecordMessage(34,0)

10/26/19 15:40:53.701 DEBUG inside incrementCopyAndCount, records copied = 35

10/26/19 15:40:53.703 DEBUG sending another CopyRecordMessage(35,0)

10/26/19 15:40:53.706 DEBUG inside incrementCopyAndCount, records copied = 36

10/26/19 15:40:53.706 DEBUG sending another CopyRecordMessage(36,0)

10/26/19 15:40:53.715 DEBUG inside incrementCopyAndCount, records copied = 37

10/26/19 15:40:53.718 DEBUG sending another CopyRecordMessage(37,0)

10/26/19 15:40:53.721 DEBUG inside incrementCopyAndCount, records copied = 38

10/26/19 15:40:53.725 DEBUG sending another CopyRecordMessage(38,0)

10/26/19 15:40:53.728 DEBUG inside incrementCopyAndCount, records copied = 39

10/26/19 15:40:53.729 DEBUG sending another CopyRecordMessage(39,0)

10/26/19 15:40:53.732 DEBUG inside incrementCopyAndCount, records copied = 40

10/26/19 15:40:53.734 DEBUG sending another CopyRecordMessage(40,0)

10/26/19 15:40:53.742 DEBUG inside incrementCopyAndCount, records copied = 41

10/26/19 15:40:53.742 DEBUG sending another CopyRecordMessage(41,0)

10/26/19 15:40:53.748 DEBUG inside incrementCopyAndCount, records copied = 42

10/26/19 15:40:53.751 DEBUG sending another CopyRecordMessage(42,0)

10/26/19 15:40:53.755 DEBUG inside incrementCopyAndCount, records copied = 43

10/26/19 15:40:53.756 DEBUG sending another CopyRecordMessage(43,0)

10/26/19 15:40:53.773 DEBUG inside incrementCopyAndCount, records copied = 44

10/26/19 15:40:53.775 DEBUG sending another CopyRecordMessage(44,0)

10/26/19 15:40:53.779 DEBUG inside incrementCopyAndCount, records copied = 45

10/26/19 15:40:53.783 DEBUG sending another CopyRecordMessage(45,0)

10/26/19 15:40:53.788 DEBUG inside incrementCopyAndCount, records copied = 46

10/26/19 15:40:53.788 DEBUG sending another CopyRecordMessage(46,0)

10/26/19 15:40:53.791 DEBUG inside incrementCopyAndCount, records copied = 47

10/26/19 15:40:53.792 DEBUG sending another CopyRecordMessage(47,0)

10/26/19 15:40:53.801 DEBUG inside incrementCopyAndCount, records copied = 48

10/26/19 15:40:53.804 DEBUG sending another CopyRecordMessage(48,0)

10/26/19 15:40:53.814 DEBUG inside incrementCopyAndCount, records copied = 49

10/26/19 15:40:53.822 DEBUG sending another CopyRecordMessage(49,0)

10/26/19 15:40:53.828 DEBUG inside incrementCopyAndCount, records copied = 50

10/26/19 15:40:53.830 DEBUG sending another CopyRecordMessage(50,0)

10/26/19 15:40:53.848 DEBUG inside incrementCopyAndCount, records copied = 51

10/26/19 15:40:53.853 DEBUG sending another CopyRecordMessage(51,0)

10/26/19 15:40:53.858 DEBUG inside incrementCopyAndCount, records copied = 52

10/26/19 15:40:53.859 DEBUG sending another CopyRecordMessage(52,0)

10/26/19 15:40:53.867 DEBUG inside incrementCopyAndCount, records copied = 53

10/26/19 15:40:53.867 DEBUG sending another CopyRecordMessage(53,0)

10/26/19 15:40:53.876 DEBUG inside incrementCopyAndCount, records copied = 54

10/26/19 15:40:53.878 DEBUG sending another CopyRecordMessage(54,0)

10/26/19 15:40:53.881 DEBUG inside incrementCopyAndCount, records copied = 55

10/26/19 15:40:53.884 DEBUG sending another CopyRecordMessage(55,0)

10/26/19 15:40:53.891 DEBUG inside incrementCopyAndCount, records copied = 56

10/26/19 15:40:53.892 DEBUG sending another CopyRecordMessage(56,0)

10/26/19 15:40:53.906 DEBUG inside incrementCopyAndCount, records copied = 57

10/26/19 15:40:53.906 DEBUG sending another CopyRecordMessage(57,0)

10/26/19 15:40:53.908 DEBUG inside incrementCopyAndCount, records copied = 58

10/26/19 15:40:53.914 DEBUG sending another CopyRecordMessage(58,0)

10/26/19 15:40:53.925 DEBUG inside incrementCopyAndCount, records copied = 59

10/26/19 15:40:53.929 DEBUG sending another CopyRecordMessage(59,0)

10/26/19 15:40:53.939 DEBUG inside incrementCopyAndCount, records copied = 60

10/26/19 15:40:53.940 DEBUG sending another CopyRecordMessage(60,0)

10/26/19 15:40:53.948 DEBUG inside incrementCopyAndCount, records copied = 61

10/26/19 15:40:53.950 DEBUG sending another CopyRecordMessage(61,0)

10/26/19 15:40:53.956 DEBUG inside incrementCopyAndCount, records copied = 62

10/26/19 15:40:53.957 DEBUG sending another CopyRecordMessage(62,0)

10/26/19 15:40:53.962 DEBUG inside incrementCopyAndCount, records copied = 63

10/26/19 15:40:53.964 DEBUG sending another CopyRecordMessage(63,0)

10/26/19 15:40:53.971 DEBUG inside incrementCopyAndCount, records copied = 64

10/26/19 15:40:53.976 DEBUG sending another CopyRecordMessage(64,0)

10/26/19 15:40:53.988 DEBUG inside incrementCopyAndCount, records copied = 65

10/26/19 15:40:53.989 DEBUG sending another CopyRecordMessage(65,0)

10/26/19 15:40:53.993 DEBUG inside incrementCopyAndCount, records copied = 66

10/26/19 15:40:53.994 DEBUG sending another CopyRecordMessage(66,0)

10/26/19 15:40:53.996 DEBUG inside incrementCopyAndCount, records copied = 67

10/26/19 15:40:54.001 DEBUG sending another CopyRecordMessage(67,0)

10/26/19 15:40:54.006 DEBUG inside incrementCopyAndCount, records copied = 68

10/26/19 15:40:54.012 DEBUG sending another CopyRecordMessage(68,0)

10/26/19 15:40:54.019 DEBUG inside incrementCopyAndCount, records copied = 69

10/26/19 15:40:54.022 DEBUG sending another CopyRecordMessage(69,0)

10/26/19 15:40:54.025 DEBUG inside incrementCopyAndCount, records copied = 70

10/26/19 15:40:54.026 DEBUG sending another CopyRecordMessage(70,0)

10/26/19 15:40:54.028 DEBUG inside incrementCopyAndCount, records copied = 71

10/26/19 15:40:54.029 DEBUG sending another CopyRecordMessage(71,0)

10/26/19 15:40:54.047 DEBUG inside incrementCopyAndCount, records copied = 72

10/26/19 15:40:54.047 DEBUG sending another CopyRecordMessage(72,0)

10/26/19 15:40:54.061 DEBUG inside incrementCopyAndCount, records copied = 73

10/26/19 15:40:54.062 DEBUG sending another CopyRecordMessage(73,0)

10/26/19 15:40:54.073 DEBUG inside incrementCopyAndCount, records copied = 74

10/26/19 15:40:54.075 DEBUG sending another CopyRecordMessage(74,0)

10/26/19 15:40:54.082 DEBUG inside incrementCopyAndCount, records copied = 75

10/26/19 15:40:54.088 DEBUG sending another CopyRecordMessage(75,0)

10/26/19 15:40:54.099 DEBUG inside incrementCopyAndCount, records copied = 76

10/26/19 15:40:54.102 DEBUG sending another CopyRecordMessage(76,0)

10/26/19 15:40:54.106 DEBUG inside incrementCopyAndCount, records copied = 77

10/26/19 15:40:54.115 DEBUG sending another CopyRecordMessage(77,0)

10/26/19 15:40:54.121 DEBUG inside incrementCopyAndCount, records copied = 78

10/26/19 15:40:54.122 DEBUG sending another CopyRecordMessage(78,0)

10/26/19 15:40:54.139 DEBUG inside incrementCopyAndCount, records copied = 79

10/26/19 15:40:54.139 DEBUG sending another CopyRecordMessage(79,0)

10/26/19 15:40:54.148 DEBUG inside incrementCopyAndCount, records copied = 80

10/26/19 15:40:54.149 DEBUG sending another CopyRecordMessage(80,0)

10/26/19 15:40:54.162 DEBUG inside incrementCopyAndCount, records copied = 81

10/26/19 15:40:54.162 DEBUG sending another CopyRecordMessage(81,0)

10/26/19 15:40:54.169 DEBUG inside incrementCopyAndCount, records copied = 82

10/26/19 15:40:54.178 DEBUG sending another CopyRecordMessage(82,0)

10/26/19 15:40:54.194 DEBUG inside incrementCopyAndCount, records copied = 83

10/26/19 15:40:54.201 DEBUG sending another CopyRecordMessage(83,0)

10/26/19 15:40:54.236 DEBUG inside incrementCopyAndCount, records copied = 84

10/26/19 15:40:54.236 DEBUG sending another CopyRecordMessage(84,0)

10/26/19 15:40:54.247 DEBUG inside incrementCopyAndCount, records copied = 85

10/26/19 15:40:54.251 DEBUG sending another CopyRecordMessage(85,0)

10/26/19 15:40:54.256 DEBUG inside incrementCopyAndCount, records copied = 86

10/26/19 15:40:54.257 DEBUG sending another CopyRecordMessage(86,0)

10/26/19 15:40:54.260 DEBUG inside incrementCopyAndCount, records copied = 87

10/26/19 15:40:54.269 DEBUG sending another CopyRecordMessage(87,0)

10/26/19 15:40:54.281 DEBUG inside incrementCopyAndCount, records copied = 88

10/26/19 15:40:54.285 DEBUG sending another CopyRecordMessage(88,0)

10/26/19 15:40:54.299 DEBUG inside incrementCopyAndCount, records copied = 89

10/26/19 15:40:54.300 DEBUG sending another CopyRecordMessage(89,0)

10/26/19 15:40:54.319 DEBUG inside incrementCopyAndCount, records copied = 90

10/26/19 15:40:54.329 DEBUG sending another CopyRecordMessage(90,0)

10/26/19 15:40:54.343 DEBUG inside incrementCopyAndCount, records copied = 91

10/26/19 15:40:54.348 DEBUG sending another CopyRecordMessage(91,0)

10/26/19 15:40:54.362 DEBUG inside incrementCopyAndCount, records copied = 92

10/26/19 15:40:54.363 DEBUG sending another CopyRecordMessage(92,0)

10/26/19 15:40:54.366 DEBUG inside incrementCopyAndCount, records copied = 93

10/26/19 15:40:54.380 DEBUG sending another CopyRecordMessage(93,0)

10/26/19 15:40:54.410 DEBUG inside incrementCopyAndCount, records copied = 94

10/26/19 15:40:54.411 DEBUG sending another CopyRecordMessage(94,0)

10/26/19 15:40:54.413 DEBUG inside incrementCopyAndCount, records copied = 95

10/26/19 15:40:54.414 DEBUG sending another CopyRecordMessage(95,0)

10/26/19 15:40:54.427 DEBUG inside incrementCopyAndCount, records copied = 96

10/26/19 15:40:54.431 DEBUG sending another CopyRecordMessage(96,0)

10/26/19 15:40:54.438 DEBUG inside incrementCopyAndCount, records copied = 97

10/26/19 15:40:54.439 DEBUG sending another CopyRecordMessage(97,0)

10/26/19 15:40:54.445 DEBUG inside incrementCopyAndCount, records copied = 98

10/26/19 15:40:54.446 DEBUG sending another CopyRecordMessage(98,0)

10/26/19 15:40:54.457 DEBUG inside incrementCopyAndCount, records copied = 99

10/26/19 15:40:54.468 DEBUG sending another CopyRecordMessage(99,0)

10/26/19 15:40:54.473 DEBUG inside incrementCopyAndCount, records copied = 100

10/26/19 15:40:54.479 DEBUG initiating file creation for file #0

10/26/19 15:40:54.482 DEBUG

10/26/19 15:40:54.484 DEBUG inside initiateFileCreation

10/26/19 15:40:54.492 DEBUG calling WriteToFileMessage(C:\home\schnarbies\output\example\EXAMPLE\_154054.491)

10/26/19 15:40:54.503 DEBUG inside FileWriterActor.writeToFile method, writing file for Array index = 0

10/26/19 15:40:54.504 DEBUG

10/26/19 15:40:54.507 DEBUG writing generic records to a file

10/26/19 15:40:54.511 DEBUG turning off record copying phase

10/26/19 15:40:54.511 DEBUG file path : C:\home\schnarbies\output\example\EXAMPLE\_154054.491

10/26/19 15:40:54.513 DEBUG file path : json

10/26/19 15:40:54.514 DEBUG opening BufferedWriter

10/26/19 15:40:54.524 DEBUG BufferedWriter open

10/26/19 15:40:54.525 DEBUG records size is : 100

10/26/19 15:40:54.538 DEBUG closing our generic records file

10/26/19 15:40:54.547 DEBUG successfully created our file

10/26/19 15:40:54.552 DEBUG files Finished Count = 1 , numFiles = 1

10/26/19 15:40:54.554 DEBUG inside KafkaProducerActor.receive method

10/26/19 15:40:54.562 DEBUG reading in file : C:\\home\\schnarbies\\output\\example\\EXAMPLE\_154054.491.json

10/26/19 15:40:54.564 DEBUG entering readInPrimaryKeyFile() method

10/26/19 15:40:54.589 DEBUG pushing records to KafkaProducer

10/26/19 15:49:15.589 DEBUG finished pushing records to KafkaProducer

10/26/19 15:49:15.591 DEBUG closing the KafkaProducer

10/26/19 15:49:15.606 DEBUG pushing DONE into queue

10/26/19 15:49:15.608 DEBUG shutting down the actor system

10/26/19 15:49:15.719 DEBUG DONE - generating streaming message information to file

10/26/19 15:49:15.805 DEBUG END - mssgen program