

# Future Transaction Writer

---

## Index

---

1. [Requirement.](#)
2. [Introduction.](#)
3. [Assumptions.](#)
4. [How to Configure?](#)
5. [How to use the Application?](#)
6. [Future Possible Improvements .](#)
7. [How to Build and Run.](#)
8. [Testing.](#)
9. [Troubleshooting.](#)

# 1. Requirement

---

- The Business user would like to see the Total Transaction Amount of each unique product they have done for the day
- The Business user would like a program that can read in the Input file and generate a daily summary report
- The Daily summary report should be in CSV format (called Output.csv) with the following specifications

The CSV has the following Headers

- Client\_Information: A combination of the CLIENT TYPE, CLIENT NUMBER, ACCOUNT NUMBER, SUBACCOUNT NUMBER fields from the Input file
- Product\_Information: A combination of the EXCHANGE CODE, PRODUCT GROUP CODE, SYMBOL, EXPIRATION DATE
- Total\_Transaction\_Amount: A Net Total of the (QUANTITY LONG - QUANTITY SHORT) values for each client per product

**Notes:** Each Record in the input file represents ONE Transaction from the client for a particular product.

## 2. Introduction

---

This application is written in Java 8. It performs below functions.

- This application creates a new CSV file using the provided configuration and metadata file.
- It streams each line, parse it and does the computation to create List of Transaction Amounts for each client per product.
- Once the record is created, the application writes the data to the output file.

## 3. Assumptions

---

- A unique record for each client per product is taken as a combination of CLIENT TYPE, CLIENT NUMBER, ACCOUNT NUMBER, SUBACCOUNT NUMBER, EXCHANGE CODE, PRODUCT GROUP CODE, SYMBOL, and EXPIRATION DATE. (Assumption is that TRANSACTION DATE is not required for the identification of Unique record. Unique Record Field in config can be updated to include TRANSACTION DATE.)
- There is no field separator in the Output apart from ",". For example, Client\_Information field will be a combination of CLIENT TYPE, CLIENT NUMBER, ACCOUNT NUMBER, SUBACCOUNT NUMBER without any field separator.
- Extra space at the end of each field will not be removed by default while parsing the record values (This can be changed by modifying the config value of Trim\_Spaces to true.). The output.csv will have the same effect.

## 4. How to Configure

---

### 4.1 config.properties

- The application needs a config file (*config.properties*) to perform the required action.
- Below are the config values.
- Config file should either be provided in the command line with the argument -Dconfig=<> or should be placed in the directory where the Jar is copied.

Config Value	Explanation
Column_Metadata	Path of the Metadata file
Fixed_Length	Fixed Line length
Trim_Spaces	true/false - weather to trim the Spaces at the end of each record or not.
Output_Structure	Structure of the output example-CLIENT TYPE CLIENT NUMBER ACCOUNT NUMBER SUBACCOUNT NUMBER,EXCHANGE CODE PRODUCT GROUP CODE SYMBOL EXPIRATION DATE
Header	Output Header - Client_Information,Product_Information,Total_Transaction_Amount
Short_Key	QUANTITY SHORT
Long_Key	QUANTITY LONG
Unique_Row	to identify a unique row, example -CLIENT TYPE CLIENT NUMBER ACCOUNT NUMBER SUBACCOUNT NUMBER EXCHANGE CODE PRODUCT GROUP CODE SYMBOL EXPIRATION DATE

## 4.2 Metadata File

- This file contains the Columns to extract from the input file.
- It is a comma-separated file.
- The column name specified in the metadata file is the default definition and the same should be used in all other places/configuration.
- This file defines the start and end index of each field, type of the field and the format. (Type and Field format have not been used in this version.)

Column Name,Start Index,End Index,Type,Format
CLIENT TYPE,4,7,STRING,NA
CLIENT NUMBER,8,11,STRING,NA
ACCOUNT NUMBER,12,15,STRING,NA
SUBACCOUNT NUMBER,16,19,STRING,NA
EXCHANGE CODE,28,31,STRING,NA
PRODUCT GROUP CODE,26,27,STRING,NA
SYMBOL,32,37,STRING,NA
EXPIRATION DATE,38,45,DATE,CCYYMMDD
QUANTITY LONG,53,62,LONG,NA
QUANTITY SHORT,64,72,LONG,NA
TRANSACTION DATE,122,129,DATE,CCYYMMDD

## 5. How to use the Application

---

This application takes 2 arguments as below.

- Argument 1 - Mandatory - Complete Path of the Input File in Fixed File Format.
- Argument 2 - Mandatory - Complete Path for the output file.

## 6. Future Possible Improvements

---

- Program to take input directory from the config file and perform read regularly for a new file.
- Program to have defined output directory in the config file and it writes a new output file for each date.
- Program currently validates the Line Length but it can be enhanced to track the error records and write to a different file.

## 7. How to Build and Run.

---

To Build the package

```
mvn clean compile assembly:single
```

To Run the application:

```
java -Dconfig="" -Dlog4j.configuration="< log4j properti path>" -jar {target_dir}/Future-Transaction-Writer-1.0.jar
```

Example:

```
java -Dconfig="C:\Users\tripaank\Downloads\config.properties" -  
Dlog4j.configuration=file:"C:\Users\tripaank\log4j.properties" -jar Future-Transaction-Writer-1.0.jar  
"C:\Users\tripaank\input" "C:\Users\tripaank\Output.csv"
```

## 8. Testing

---

Most of the common scenarios have been validated with JUnit and manual testing. Test src is included in the repository.

## 9. Troubleshooting

---

- **FileNotFoundException:** Validate the location of the Metadata file, Config File, Input file, Log File and Output.
- **No Record in Output:** Validate if the Fixed\_Length in the config is correct and matching with the input file line width.
- **NotEnoughArgumentException:** Validate the number of Arguments passed to the Application.
- Follow the log files to find out about the issues.