

Integration REPORT



TEAM: ZERO DEFECTS (Group 30)

|  |  |
| --- | --- |
| Name | Student Number |
| Muhammad Usman Majeed | 10086980 |
| Jessica Nahulan | 10029341 |
| Johan Cornelissen | 10098176 |

Contents

[Source Listing 3](#_Toc499294151)

[Scripts Printout 3](#_Toc499294152)

[Integration Defect Table 3](#_Toc499294153)

# Script Execution Instructions

## Daily Script

The daily script (dailyScript.sh) is used to run the FrontEnd application and BackOffice application to simulate a day’s worth of transaction. Currently the script is set to run the FrontEnd (ATM type) application 3 times before running the BackOffice application once at the end of the day. The 3 FrontEnd’s transaction summary files are merged by the script before being provided to the BackOffice. The script itself has 3 inputs and creates 4 types of outputs as seen below. Example usage for the script can be found below along with the error message provided if the user does not provide the correct number of input parameters.

**Daily Script Inputs:**

* Valid Accounts File: Path to the valid accounts file to be used for all the FrontEnd sessions **during** the day. This file is provided by the previous day’s BackOffice application. This path is also used to output the new valid accounts file at the **end** of the day.
* Master Accounts File: Path to the master accounts file to be updated by the BackOffice at the **end** of the day.
* Directory of transaction session inputs: Specify the directory where the transaction session inputs for the day can be found. This directory **must** contain as many files as necessary for the amount of FrontEnd calls (in this case 3). Each filename **must** be in the format of “input\_daily\_transaction\_<run #>.txt”.

**Daily Script Outputs:**

* Valid Accounts File: The new valid accounts file created by the BackOffice at the end of the day will be saved in the path provided using the script parameters (see inputs above).
* Master Accounts File: The new master accounts file created by the BackOffice at the end of the day will be saved in the path provided using the script parameters (see inputs above).
* Merged Transaction Summary File: The merged transaction summary file created by concatenating all the FrontEnd’s transaction summary files can be found in "Daily\_Transaction\_File/mergedTransactionSummaryFile.txt". For simplicity and space consumption, this file will be overwritten on subsequent daily script calls.
* Logs: Logs are created for each FrontEnd and BackOffice session’s std output. Each log is saved in Logs/ in the format of "Logs/frontEnd\_run\_${time}.txt" and "Logs/backend\_run\_${time}.txt".

**Example Usage:**

./dailyScript.sh validaccounts.txt masteraccounts.txt Input\_Files/Day3

**Error message displayed if invalid number of parameters passed to daily script:**

Johans-MacBook-Pro:DriverProgram johancornelissen$ ./dailyScript.sh validaccounts.txt masteraccounts.txt

Error, 3 input arguements must be supplied to dailyScript.sh

Example usage: ./dailyScript.sh validaccounts.txt masteraccounts.txt Input\_Files/Day1

Where:

 validaccounts.txt is the valid accounts file used for the day

 masteraccounts.txt is the master accounts file to use for backend processing at the end of the day

 Input\_Files/Day1 is the path to the directory containing the input transaction files

## Weekly Script

The weekly script (weeklyScript.sh) is used to run the daily script a total of 5 times. This corresponds to the number of business days in a week as it is assumed that the banking system is not in use during the weekend. This action simulates the banking system being used for a total of 5 days, where each day consists of 3 FrontEnd runs and one BackOffice run. The weekly script does not directly take any input parameters but does use multiple pre-defined inputs as described below. The inputs used by the script are assumed to be of proper format and in the pre-defined locations (example, the transaction session inputs must be found in “Input\_Files/Day[X]”).

**Weekly Script Inputs:**

The weekly script does not directly accept any inputs; however, the daily script is called for each processing day using a set of inputs. See the description of dailyScript.sh inputs above. The weekly script itself uses “Output\_Files/masteraccounts.txt” and “Output\_Files/validaccounts.txt” as the **starting** of the week account files (which are cleared at the start of the week) as well as the location for saving the **final** master and valid accounts files created for the week’s transactions. The transaction session inputs used for each day should be located in “Input\_Files/Day[X]” **before** starting the weekly script.

**Weekly Script Outputs:**

* Valid Accounts File: The new valid accounts file created by the BackOffice at the end of the **last day in the week** will be saved at “Output\_Files/validaccounts.txt”.
* Master Accounts File: The new master accounts file created by the BackOffice at the end of the **last day in the week** will be saved at “Output\_Files/masteraccounts.txt”.
* Logs: The daily script which is invoked through the weekly script creates logs for each FrontEnd and BackOffice session’s std output. Each log is saved in Logs/ in the format of "Logs/frontEnd\_run\_${time}.txt" and "Logs/backend\_run\_${time}.txt". The /Logs directory is cleared at the beginning of **each** **week**.

**Example Usage:**

./weeklyScript.sh

# Source Listing

## Daily Script Source (BASH):

#!/bin/bash

# STDOUT Colour Definitions (NC stands for No Colour)

RED='\033[0;31m';

BLUE='\033[0;34m';

GREEN='\033[0;32m';

NC='\033[0m';

#Check number of input arguements

if [ $# -ne 3 ]

  then

errorMessage="

${RED}Error, 3 input arguements must be supplied to dailyScript.sh

\nExample usage: ./dailyScript.sh validaccounts.txt masteraccounts.txt Input\_Files/Day1${NC}

\nWhere:

\n validaccounts.txt is the valid accounts file used for the day

\n masteraccounts.txt is the master accounts file to use for backend processing at the end of the day

\n Input\_Files/Day1 is the path to the directory containing the input transaction files

"

echo -e $errorMessage

exit

fi

#Create a temporary directory to store transaction summary files for each front end session

mkdir Temp\_Output\_Files/

numFrontEndRuns=3

#Run front end application X number of times. Where stdout is stored in Logs/, and transaction summary file in stored in Temp\_Output\_Files

echo -e "${BLUE}Running Front End $numFrontEndRuns times. View logs at \"Logs/frontEnd\_run\_<run\_time>.txt\"  ${NC}";

for i in $(seq 1 $numFrontEndRuns)

do

time=$(date +"%FT%H%M%S%3N")

../FrontEnd/QBasic.py "${1}" "Temp\_Output\_Files/transactionSummaryFile\_${i}.txt" < "${3}/input\_daily\_transaction\_${i}.txt" > "Logs/frontEnd\_run\_${time}.txt"

done

echo -e "Completed running Front End $numFrontEndRuns times.";

#Create the merged transaction summary file from all transaction summary files created in Temp\_Output\_Files directory.

#Save the merged transaction file in Daily\_Transactio\_File/mergedTransactionSummaryFile.txt (overwrite if necessary)

echo -e "\n${BLUE}Creating merged transaction summary file.  ${NC}";

FILES="Temp\_Output\_Files/\*"

firstFile=true

for file in $FILES

do

if [ "$firstFile" = true ] ; then

mergedTransactionString="$(cat ${file})"''

firstFile=false

else

mergedTransactionString="${mergedTransactionString}\n$(cat ${file})"''

fi

done

echo -e "${mergedTransactionString}" > "Daily\_Transaction\_File/mergedTransactionSummaryFile.txt";

echo -e "Successfully created merged transaction summary file.";

#Run the BackOffice application using the merged transaction summary file found in Daily\_Transaction\_File/mergedTransactionSummaryFile.txt

time=$(date +"%FT%H%M%S%3N")

echo -e "\n${BLUE}Running BackOffice with merged transaction summary file. View logs at \"Logs/backend\_run\_${time}.txt\"  ${NC}";

../BackOffice/BackOffice.py "${2}" "Daily\_Transaction\_File/mergedTransactionSummaryFile.txt" "${2}" "${1}" > "Logs/backend\_run\_${time}.txt"

echo -e "Completed running BackOffice.";

#Delete temporary directory used to store transaction summary files for each front end session

rm -rf Temp\_Output\_Files

## Weekly Script Source (BASH):

#!/bin/bash

# STDOUT Colour Definitions (NC stands for No Colour)

RED='\033[0;31m';

BLUE='\033[0;34m';

GREEN='\033[0;32m';

NC='\033[0m';

#Clear out the log file directory at the start of each week (logs created by frontEnd and backOffice applications)

rm -rf Logs/\*

#Empty the masteraccounts.txt and validaccounts.txt file from the previous week.

#As stated in Assignment 6 description, each week should start with empty masteraccounts.txt and validaccounts.txt files.

> Output\_Files/masteraccounts.txt

> Output\_Files/validaccounts.txt

#Set the number of days in a week

numDays=5

#Run the daily script for every day in the week

for i in $(seq 1 $numDays)

do

echo -e "\n${RED}Running Daily Script for Day ${i}.";

./dailyScript.sh "Output\_Files/validaccounts.txt" "Output\_Files/masteraccounts.txt" "Input\_Files/Day${i}"

done

echo -e "\n\n${GREEN}Weekly QBasic run completed successfully. Daily script (dailyScript.sh) was executed total of $numDays times.${NC}"

# Scripts Printout

## Transaction Session Inputs for Daily Script

## Merged Transaction Summary File after Daily Script execution

## Master Accounts File after each day created by Weekly Script

# Integration Defect Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Defect# | Defect Area | Defect description | Error in code, and fix applied | Re-Test details |
| 1 | Login | When users log in but do not successfully complete the login process - that is, they do not specify a valid Machine/Agent type. The system currently treats this as a valid login and default the type to Machine and allows subsequent transactions.  Expected behavior is that no transaction is allowed until the user has the completed the login process and specified the valid session type. | The system login status was being set to true at the beginning of the login process before even obtaining the session type from the user. So, the invalid session type still left the system logged in.  Fixed so that the system’s login status is not set to true until the login process has been complete, and the session type has been successfully obtained from the user. | Performed manual unit test on several different transactions at the point where it asks for session type. Checked if the transactions were able to bypass the session type command.  Created additional tests in Front End testing suite to ensure defect can be detected if it re-occurs in the future. |

# 

# Team Contribution:

|  |  |  |
| --- | --- | --- |
| Name | Hours Spent | Tasks |
| Muhammad Usman Majeed | 2.5 hours | * Listed source for WITHDRAW transaction testing. * Defined decision partitions for WITHDRAW transaction testing. * Defined test cases for WITHDRAW transaction testing. * Corrected any failures encountered during WITHDRAW tests. |
| Jessica Nahulan | 2.5 hours | * Participated in meeting to decide on testing methods and splitting up tasks. * Analyzed test cases for WITHDRAW transaction testing. * Ran and documented WITHDRAW tests. * Created overall documentation and reporting skeleton |
| Johan Cornelissen | 2.5 hours | * Participated in meeting to decide on testing methods and splitting up tasks. * Listed source for CREATEACCT transaction testing. * Defined input partitions for CREATEACCT transaction testing. * Defined and analyzed test cases for CREATEACCT transaction testing. * Ran CREATEACCT tests and corrected any failures encountered. |