Testing

BMI Dashboard, Team Dash, December 3, 2015

The BMI Dashboard consists of three modules for the system to function. The three modules are: data acquisition, data storage, and data display. Each one of the team members did six test cases. We also did test cases that satisfy the user stories listed in the sprints(1,2,3).

Data Display Test Case 1 by Venkata Karthik Thota

The function parseTime(d.TimeStep) is called in the loop called dataset.forEach. The function parseTime takes in one parameter which is a timestamp value. This parameter is read from the dataset(which is the csv file). The d3.time.format("%H:%M").parse is a D3 in-built function that checks if the parameter being passed in matches the expression "%H:%M" which denotes for Hour(1-24):Minutes(00-60). On success, the timestamp is parsed without any errors into a 12-hour clock representation. For example, 0:00 is 12:00 AM and 13:00 is 1:00 PM. On failure, the timestamp returns a NaN (error in JavaScript).

```
d.TimeStep = +parseTime(d.TimeStep);
```

```
var parseTime = d3.time.format("%H:%M").parse;
.
.
.
dataset.forEach(function(d) {
      d.TimeStep = +parseTime(d.TimeStep);
      d.Temperature = +d.Temperature;
      d.Vac = +d.Vac;
      d.Pac = +d.Pac;
      d.VarDate = d.VarDate;
});
```

<u>The equivalences classes for parseTime are:</u>

E.C.1 = XX:YY where 0 <= XX <= 24 and 0 <= YY <= 59 (correct)

E.C.2 = XX:YY where 0 > XX and XX > 24 and 0 > YY and YY > 50 (correct)

E.C.3 = XXYY where 0 <= XX <= 24 and 0 <= YY <= 59 (incorrect)

Set of Tests are:
Input = 0:00, 6:00; expected output = 12 AM, 6AM. Satisfies E.C.1.
Input = 13:00; expected output = 1 PM. Satisfies E.C.1.
Input = 24:00; expected output = 12 PM. Satisfies E.C.1.
Input = 25:00; expected output = NaN. Satisfies E.C.2
Input = 0, 4; expected output = NaN. Satisfies E.C.3
Input = 25; expected output = NaN. Satisfies E.C.3

Boundary Values:
E.C.6 = 0:00
E.C.7 = 1:00
E.C.8 = 24:00
E.C.9 = 23:00
E.C.10 = 25:00
EC.11 = -1:00
EC.12 = 23:59

Set of Test for Boundary Values:
Input = 0:00, expected output = 12 AM. Satisfies E.C.6
Input = 1:00, expected output = 1 AM. Satisfies E.C.7
Input = 24:00, expected output = 12 PM. Satisfies E.C.8
Input = 23:00, expected output = 11 PM. Satisfies E.C.9
Input = 25:00, expected output = NaN. Satisfies E.C.10
Input = -1:00, expected output = NaN. Satisfies E.C.11
Input = 23:59, expected output = 11:59 PM. Satisfies E.C.12

Data Acquisition Test Case 2 by Justin Barros

The function createDataDict is found in the Parser.py file. This function parses out the data dictionary from the ESO file that has been cleaned (.csv and ESO format) for reference. "Extracts a data dictionary that maps each of the numerical keys to their string locations." The data dictionary is a set of keys along with the attribute that corresponds to each key. The only parameter for the function is a filepath. The filepath is the directory path to where we store the ESO file.

```
def createDataDict(filepath):
    data = openFile(filepath).splitlines()
    dname = "DATABASE/"
    bname = "Data_Dictionary.csv"
    dict_file = open(dname + bname, "w")
    x = 0
    for d in data:
        if d == "End of Data Dictionary":
            break
        attributes = d.split(",")
        if len(attributes) >= 4 and x >= 6:
            val = attributes[0] + "," + attributes[2] + "," + attributes[3] + "\n"
            dict_file.write(val)
        x += 1
    dict_file.close()
    print("Dictionary has been created in directory CLEAN_ESO as 'Data_Dictionary.csv'\n")
```

<u>The equivalences classes for createDataDict are:</u>
E.C.1.1 = filepath is valid (goes to ESO file in directory)
E.C.1.2 = filepath is invalid

<u>Set of Tests:</u>
Input = valid filepath; expected output = a file with data dictionary in it (Data_Dictionary.csv). Satisfies E.C.1.
Input = invalid filepath; expected output = no such file or directory .Satisfies E.C.2

<u>Boundary Values:</u>
E.C.1.1 = filepath is valid (End of filepath is an actual readable file. Example: "/directory/subdirectory/file")

E.C.1.2 = filepath is invalid (End of filepath is a directory and not a readable file. Example: "/directory/subdirectory/subsubdirectory")

Test Cases with Boundary Values:
Input = file; expected output = A file that is a data dictionary.
Input = directory; expected output = No such file or directory.

Data Acquisition Test Case 3 by Sterling Salvaterra

The function parseESOKeys is found in the Parser.py file. This function runs through the clean ESO file and takes each key and makes a file of each key, the name of the file being the key. The function also takes the data for each key and places it in the corresponding files. It then takes the timestamps and prepends them in front of that data. The filepath is the directory path to where we store the ESO file. The tsc parameter is a time stamp counter. This counter is used to keep track of the most recent timestamp. It is an index to the timestamp file so that we know which timestamp to attach to the current data stream.

```
def parseESOKeys(filepath, tsc):
    data = openFile(filepath).splitlines()
    dname = "CLEAN_ESO\\"
    bname = os.path.basename(filepath)

    ts_filepath = "2" + bname[:len(bname)-4] + ".csv"
    ts_file = open(dname + ts_filepath, "a")


    for d in data:
        if d.startswith("2,", 0, 2):
            ts_file.write(d + "\n")
    ts_file.close()
    timestamps = openFile(dname + ts_filepath).splitlines()

      for d1 in data:
        if d1 == "End of Data Dictionary":
            break
        attributes = d1.split(",")
        key = attributes[0]
        t = key + "_" + bname[:len(bname)-4] + ".csv"
        temp_file = open(dname + t, "a")
        for d2 in data:
            if d2.startswith(key + ",", 0, len(key)+1) and not d2.startswith("2,", 0, 2):
                temp_file.write(timestamps[tsc] + "," + d2 + "\n")
```

The equivalences classes for parseESOKeys are:
E.C.1.1 = filepath is valid (goes to ESO file in directory)

E.C.1.2 = filepath is invalid
E.C.2.1 = 0 <= tsc < 289
E.C.2.2 = tsc < 0
E.C.2.3 = tsc > 289

<u>Set of Tests:</u>
Input = valid filepath x 35; expected output = File for each key with corresponding data (Data_Dictionary.csv). Satisfies E.C.1.1 , E.C.2.1

Input = valid filepath x -2; expected output = index out of bounds exception
Satisfies E.C.1.1, E.C.2.2

Input = valid filepath x 350; expected output = index out of bounds exception
Satisfies E.C.1.1, E.C.2.3

Input = invalid filepath x 35; expected output = no such file or directory.
Satisfies E.C.1.2, E.C.2.1
Input = invalid filepath x-2; expected output = no such file or directory.
Satisfies E.C.1.2, E.C.2.2

Input = invalid filepath x350; expected output = no such file or directory.
Satisfies E.C.1.2, E.C.2.3

<u>Boundary Values:</u>
E.C.2.4: -1
E.C.2.5: 0
E.C.2.6: 1
E.C.2.7: 288
E.C.2.8: 289
E.C.2.9: 290

<u>Test Cases with Boundary Values:</u>
Input = valid filepath x -1; expected output = index out of bounds (Data_Dictionary.csv). Satisfies E.C.1.1 , E.C.2.4

Input = valid filepath x 0; expected output = File for each key with corresponding data
Satisfies E.C.1.1, E.C.2.5

Input = valid filepath x 1; expected output = File for each key with corresponding data
Satisfies E.C.1.1, E.C.2.6

Input = valid filepath x 288; expected output = File for each key with corresponding data.
Satisfies E.C.1.1, E.C.2.7

Input = valid filepath x 289; expected output = the program halts.
Satisfies E.C.1.1, E.C.2.8

Input = valid filepath x 290; expected output = index out of bounds
Satisfies E.C.1.1, E.C.2.9

Input = invalid filepath x -1; expected output = no such file or directory(Data_Dictionary.csv). Satisfies E.C.1.2 , E.C.2.4

Input = invalid filepath x 0; expected output = no such file or directory
Satisfies E.C.1.2, E.C.2.5

Input = invalid filepath x 1; expected output =no such file or directory
Satisfies E.C.1.2, E.C.2.6

Input = invalid filepath x 288; expected output = no such file or directory
Satisfies E.C.1.2, E.C.2.7

Input = invalid filepath x 289; expected output = no such file or directory
Satisfies E.C.1.2, E.C.2.8

Input = invalid filepath x 290; expected output = no such file or directory
Satisfies E.C.1.2, E.C.2.9

Data Storage Test Case 4 by Oscar Pinedo

The function insertToDB_PV takes in a clean PV file and reads inserts the values into the database. The parameters to this function are the connection to the database server containing the pre-constructed PV database and a Scanner object for the clean PV file that contains the data to be inserted into the database. The last parameter in this function is the filename of the PV file being read through the

Scanner and this it is simply used to extract the date from the filename. The filename extracted is then reformatted by date_parse() and inserted as one of the values in the database.

```
static void insertToDB_PV(Connection myConn, Scanner in1, String filename){
       try{
              String line,value1,value2,date,timeStamp;
              String insertDB = "INSERT INTO PV"
                            +"(TimeStamp, Date, PAC, Temperature) VALUES"
                            + "(?,?,?,?)";
              PreparedStatement myStmt = myConn.prepareStatement(insertDB);
           date = date_parse(filename);
           while(in1.hasNextLine()){
              line = in1.nextLine();
              String[] results = line.split(",");
              timeStamp = results[0];
              value1 = results[3];
              value2 = results[2];
              myStmt.setString(1, timeStamp);
              myStmt.setString(2, date);
              myStmt.setString(3, value1);
              myStmt.setString(4, value2);
              myStmt.executeUpdate();
           }

       }catch(Exception x){
              x.printStackTrace();
       }
}
```

```
//uses the filename to parse out the date
static String date_parse(String date){
       String[] something = date.split("\\.");
       String[] something1 = something[0].split("_");
       String[] something2 = something1[1].split("-");
       String something3 = something2[1] + "-" + something2[2] + "-" + something2[0];
       return something3;
}
```

The equivalences classes for insertToDB_PV() are:

EC1.1 = myConn is a valid connection
EC1.2 = myConn is not a valid connection
EC2.1 = in1 is a Scanner object to file formatted correctly (clean pv file)
EC2.2 = in1 is a Scanner object to file not in correct format (any other file)
EC3.1 = the pv filename is formatted correctly YYYY-MM-DD.csv
EC3.2 = the pv filename is in an incorrect format but valid format
EC3.3 = the pv filename is in an invalid format

Set of Tests:

Input = valid Connection object x valid Scanner object x filename in correct format;
expected output = data is inserted into database
Satisfies E.C.1.1, E.C.2.1, E.C.3.1

Input = invalid Connection object x valid Scanner object x filename in correct format;
expected output = throws an exception and no data is inserted
Satisfies E.C.1.2, E.C.2.1, E.C.3.1

Input = valid Connection object x invalid Scanner object x filename in correct format;
expected output = no data is inserted and program throws exception
Satisfies E.C.1.1, E.C.2.2, E.C.3.1

Input = invalid Connection object x invalid Scanner object x filename in correct format;
expected output = throws an exception and no data is inserted
Satisfies E.C.1.2, E.C.2.2, E.C.3.1

Input = valid Connection object x valid Scanner object x filename is incorrect but can still be parsed;
expected output = data will be inserted into the database but will have junk value for the date field
Satisfies E.C.1.1, E.C.2.1, E.C.3.2

Input = valid Connection object x valid Scanner object x filename format is invalid;
expected output = will throw an ArrayIndexOutOfBoundsException and no data will be inserted
Satisfies E.C.1.1, E.C.2.1, E.C.3.3

Boundary Values:
There are no boundary values for this function since each of the parameters will either result in the program running successfully or having it crash/throw an exception.

Data Storage Test Case 5 by Octavio Rodriguez

The function printFromDB_PV uses the connection to the database server to query the PV table and print the file takes as input the connection to the database url which was created before the function was called. We use the connection object to send statements/queries to the database. In this function we only use one query which selects all the data from the PV table and it is then written to a file. We print the data in CSV format so that the data visualization tool can utilize the data for the graph display. This function should always run successfully provided we have a valid connection because the PV table has already been created.

```
static void printFromDB_PV(Connection myConn){
       //create a statement.
       try{
       Statement pvStmt = myConn.createStatement();
       BufferedWriter out = new BufferedWriter (new FileWriter("dataC.csv"));

       //make and SQL query for the PV table in the database.
       ResultSet pvResults = pvStmt.executeQuery("select * from PV");

       //write the columns of the database before the data is written out
       out.write("TimeStamp"
                     + "," + "Date"
                     + "," + "Pac"
                     + "," + "Temperature"
                     + "\n");
       //process the results from the pv table in the database
       while(pvResults.next()){
              out.write(pvResults.getString("TimeStamp")
                            + "," + pvResults.getString("Date")
                            + "," + pvResults.getString("Pac")
                            + "," + pvResults.getString("Temperature")
                            + "\n");

       }
       out.flush();
       out.close();
       }catch(Exception x){
              x.printStackTrace();
       }
}
```

The equivalences classes for printFromDB_PV are:

EC1.1 = myConn is a valid connection
EC1.2 = myConn is not a valid connection

Set of Tests:
Input = valid connection
expected output = file with contents of PV table is written
Satisfies EC1.1

Input = invalid connection
expected output = exception is thrown and no file is written
Satisfies E.C.1.2

Boundary Values:
No boundary values since all that is needed is a valid connection to the database server and if it is valid the PV file is written otherwise an exception is thrown.

Data Storage Test Case 6 by Allen Liou

The function printFromDB_BMS_Combined prints the contents of one of the two BMS tables depending on the room number that is specified. We only have the data for two of the rooms which are the only room numbers that count as valid input into our function.

```
	static void printFromDB_BMS_Combined(Connection myConn, int roomNum){

		try{
			Statement pvStmt = myConn.createStatement();
			ResultSet pvResults = null;

			BufferedWriter out = null;
			switch (roomNum){
			case 1:
				out = new BufferedWriter (new FileWriter("BMSOne.csv"));
				pvResults = pvStmt.executeQuery("select * from BMSR1");
				break;
			case 2:
				out = new BufferedWriter (new FileWriter("BMSTwo.csv"));
				pvResults = pvStmt.executeQuery("select * from BMSR2");
				break;
			default:
				throw new Exception(“not a valid room number”);
			}

			//write the column names
			out.write("TimeStamp"
					+ "," + "Date"
					+ "," + "Temperature"
					+ "," + "RelativeHumidity"
					+ "," + "COtwo"
					+ "," + "SensibleHeat"
					+ "\n");
			//process the data from database in BMS
			while(pvResults.next()){
				out.write(pvResults.getString("TimeStamp")
						+ "," + pvResults.getString("Date")
						+ "," + pvResults.getString("Temperature")
						+ "," + pvResults.getString("RelativeHumidity")
						+ "," + pvResults.getString("CO_2")
```

```
                                + "," + pvResults.getString("SensibleHeat")
                                + "\n");
                }
                        out.flush();
                        out.close();


                }catch(Exception x){
                        x.printStackTrace();
                }
        }
```

The equivalences classes for printFromDB_BMS_Combined() are:

E.C.1.1 : valid connection

E.C.1.2 : invalid connection

E.C.2.1 : $\{roomNum :\ roomNum < 1 \cup roomNum > 2\}$ (Incorrect) not a valid room

E.C.2.2 : $\{roomNum :\ roomNum = 1\}$ (Correct)

E.C.2.3 : $\{roomNum :\ roomNum = 2\}$ (Correct)

Set of Tests:

{ (Connection valid, -5)	(Incorrect)
(Connection valid, 1)	(Correct)
(Connection valid, 2)	(Correct)
(Connection valid, 7)	(Incorrect)
(Connection invalid, 1)	(Incorrect) }

Input = valid Connection object x -5;
Expected output = throws exception and file is never written
Satisfies E.C.1.1, E.C.2.1

Input = valid connection x 1;
Expected output = queries all columns from BMSOne table and writes output to a file in csv format
Satisfies E.C.1.1, E.C.2.2

Input = valid connection x 2;
Expected output = queries all columns from BMSTwo table and writes output to a file in csv format
Satisfies E.C.1.1, E.C.2.3

Input = valid connection x 7;
Expected output = throws exception and file is never written
Satisfies E.C.1.1, E.C.2.1

Input = invalid connection x 1;
Expected output = throws exception and file is never written
Satisfies E.C.1.2, E.C.2.1

Boundary Values:
EC2.4 = {0}
EC2.5 = {3}

Test Cases with Boundary Values:

{ (Connection valid, -5)	(Incorrect)
(Connection valid, 1)	(Correct)
(Connection valid, 2)	(Correct)
(Connection valid, 7)	(Incorrect)
(Connection invalid, 1)	(Incorrect)
(Connection valid, 0)	(Incorrect)
(Connection valid, 3)	(Incorrect) }

Input = valid Connection object x -5;
Expected output = throws exception and file is never written
Satisfies E.C.1.1, E.C.2.1

Input = valid connection x 1;
Expected output = queries all columns from BMSOne table and writes output to a file in csv format
Satisfies E.C.1.1, E.C.2.2

Input = valid connection x 2;
Expected output = queries all columns from BMSTwo table and writes output to a file in csv format
Satisfies E.C.1.1, E.C.2.3

Input = valid connection x 7;
Expected output = throws exception and file is never written
Satisfies E.C.1.1, E.C.2.1

Input = invalid connection x 1;
Expected output = throws exception and file is never written
Satisfies E.C.1.2, E.C.2.1

Input = valid connection x 2;
Expected output = queries all columns from BMSTwo table and writes output to a file in csv format
Satisfies E.C.1.1, E.C.2.3

Input = valid connection x 0;
Expected output = throws exception and file is never written
Satisfies E.C.1.1, E.C.2.4

Input = valid connection x 3;
Expected output = throws exception and file is never written
Satisfies E.C.1.1, E.C.2.5