# **PseudoCode for CSV Parser by Victor Aluko**

## **Pseudo Code for importing from a CSV file**

**Input:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S/N** | **Param name** | **Data Type** | **Description** | **Required** | **Possible Values** |
| 1 | InputFile | String | The full path to the CSV file to be read | Yes | /home/Ubuntu/report.csv |
| 2 | OutputType | Int | The output type of the data read from the CSV file | Yes | 1 => Collection of Objects  2 => Collection List |
| 3 | AbortOnError | Bool | Indicates if the flow execution should abort when a row fails validation | Yes | True or False  Default => false |
| 4 | UseHeaders | Bool | Indicates if the first row should be used as the attributes of the collection to be built | Yes | True or False  Default => false |
| 5 | ColumnMapping | List | Contains the mapping of the column number to the attribute name. If this is used and UseHeaders is true, the values in this mapping will override the headers read from the first row.  If a column is skipped and Useheaders is false, the attribute name will be used as “colX” where X is the column number | No | [0 => sn, 2 => last\_name, 5 => mobile\_phone] |
| 6 | ValidationRules | List | The validation rules to validate each column from the rows | No | [0 => [type => “int”], 2 => [type => “string”], 5 => [type => “digits”, maxlength => 15], 6 => [type => “string”, allowed\_values => [“victor”,”leyian”] |
|  |  |  |  |  |  |

### **Flow Execution:**

* Validate that InputFile is readable. If not readable, throw exception with message “InputFile not readable” and terminate
* Validate the OutputType provided. If not valid, throw exception with message “Unknown Output Type” and terminate
* If useHeaders is false, generate the attribute names for the desired type
  + If ColumnMapping is provided, use the ColumnMapping to generate the attribute names
  + For missing columns, generate the column names for them using the format “colX”
* Open file for reading
* If useHeaders is true
  + Read the first row from the file
  + Convert the column values to attribute name.
    - Convert all multiple spaces to single space i.e. “ “ => “ “
    - Convert all single space to underscore i.e. “First name” => “First\_name”
    - Remove all special characters i.e. “Student’s Address” => “Students\_Address”
    - Convert the string to lower case i.e. “First\_name” => “first\_name”
  + If ColumnMapping is provided, use the values in ColumnMapping to override the headers read
* Read all subsequent rows from the file
* If ValidationRules is provided, run each row through the validation rules
* If row fails the validation rules
  + If AbortOnError is true,
    - Close file
    - Discard data already fetched from file
    - Throw exception
  + If AbortOnError is false
    - Increase counter of failed rows
    - Log the row as a failed read
    - Append the error message generated and continue
* If row passes validation rules
  + Convert the row data to the appropriate Output type i.e. List or Object. This would involve setting the attributes values to the column values of that row
  + Append the List/Object to the collection
  + Increment the successful reads count
* Continue reading and processing the rows in the CSV until all rows are read and processed
* After reading all rows and data, return the number of successful rows read
* Calling method can access the collection of successful records from the class properties

## **Pseudo Code for Exporting to a CSV File**

**Input**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S/N** | **Param Name** | **Type** | **Description** | **Required?** | **Possible Values** |
| 1 | Objects or List Collection | Collection | The data collection to be exported to the CSV file | Yes | Collection of objects OR  Collection of List data |
| 2 | OutputFile | String | The full path to the output CSV file | Yes | /home/Ubuntu/output.csv |
| 3 | AttributeMapping | List | The mapping to be used for assigning the attributes to the columns in the CSV file | Yes | [0 => id, 1 => last\_name, 2 => first\_name] |
| 4 | AddHeaders | Bool | Indicates if a row containing the headers should be added | Yes | True or False  Default => true |
| 5 | ColumnHeaders | List | Contains the desired column header names. If this is empty of a column is missing here, the attribute name from the AttributeMapping will be used | No | [0 => “Student ID”, 2 => “First Name”] |
| 6 | AppendToFile | Bool | Indicates if the data to be written should be appended to the file or the file truncated | Yes | True or False  Default => false |
| 7 | AbortOnError | Bool | Indicates if the program should abort when a failed write occurs | No | True or False  Default => false |

### **Flow Execution**

The following flow indicates the case when AppendToFile is false i.e. if the file should be created

* Check that the output file exists
* If it does not exist, attempt to create it with the appropriate permissions
  + If file could not be created, throw exception and terminate
* If file exists but is not writable, throw exception and terminate
* Truncate file and delete all its contents
* Open the output file for writing
* If AddHeaders is true, Generate the column headers for the file
  + If columnHeaders is provided, use it to generate the column headers for the CSV file
  + Add the column headers not provided in ColumnHeaders
* Write the generated column Headers to the output file
* Loop over the collection data
* Generate the row of data to be written to the file using the AttributeMapping
  + If data could not be written to row in the file
    - If AbortOnError is true, Truncate the file and throw exception
    - If AbortOnError is false, Log the role as a failed write with reason
    - Increment the failed writes count and continue
  + If data successfully written to row in the file
    - Increment the successful writes count and continue
* When all the records in the Collection are handled, close the file from writing
* Return the number of successful writes

The flow execution for AppendToFile is true is similar to the above with the only different that the file is not created and the data is appended to the file without the column headers.