



Processing Files (CSV and Excel)

For Wakanda-4D

**Supervised by:**

**Mr. Abdelahad SATOUR**

**Made by :**

**Amal BAHOUS  
Najoua MAHI**

## 1 Table of contents

1. Generality:.....	4
1.1 Purpose of the document :.....	4
1.2 Introduction :.....	4
2 Contents :.....	5
2.1 Class diagramm of JavaCsv :.....	5
2.1.1 JavaCvsReader class:.....	5
2.1.2 JavaCsvWriter class :.....	6
2.2 JavascriptCsv API : .....	6
2.2.1 CsvReader class: .....	6
2.2.2 CsvWriter class : .....	7
2.3 Progression :.....	8

## 1. Generality:

### 1.1 Purpose of the document :

This document represents the progression and the future tasks of our project; processing files (CSV and Excel) for Wakanda.

### 1.2 Introduction :

Wakanda is a new 4D product; it's a multiplatform environment for development and deploying Web, and Mobile application in JavaScript (100%).

Our goal is to develop an API or a module for processing files in Wakanda.

We have started with developing the API JavascriptCsv, drawing the API JavaCsv that contains two important classes: the CsvReader class for reading CSV files and the CsvWriter class for writing.

## 2 Content of the document :

### 2.1 Class diagramm of JavaCsv :

#### 2.1.1 JavaCsvReader class:

CsvReader	
-	<b>ESCAPE_MODE_BACKSLASH</b> : int
-	<b>ESCAPE_MODE_DOUBLED</b> : int
-	<b>inputStream</b> : inputStream
-	<b>delimiter</b> : char
-	<b>charset</b> : charset
-	<b>fileName</b> : String
+	<b>CsvReader ()</b>
+	<b>getCaptureRawRecord ()</b> : boolean
+	<b>setCaptureRawRecord ()</b> : void
+	<b>getRawRecord ()</b> : String
+	<b>getTrimWhitespace ()</b> : boolean
+	<b>setTrimWhitespace ()</b> : void
+	<b>getDelimiter ()</b> : char
+	<b>setDelimiter ()</b> : void
+	<b>getRecordDelimiter ()</b> : char
+	<b>setRecordDelimiter ()</b> : void
+	<b>getTextQualifier ()</b> : char
+	<b>setTextQualifier ()</b> : void
+	<b>getUseTextQualifier ()</b> : boolean
+	<b>setUseTextQualifier ()</b> : void
+	<b>getComment ()</b> : char
+	<b>setComment ()</b> : void
+	<b>getUseComments ()</b> : boolean
+	<b>setUseComments ()</b> : void
+	<b>getEscapeMode ()</b> : int
+	<b>setEscapeMode ()</b> : void
+	<b>getSkipEmptyRecords ()</b> : boolean
+	<b>setSkipEmptyRecords ()</b> : void
+	<b>getSafetySwitch ()</b> : boolean
+	<b>setSafetySwitch ()</b> : void
+	<b>getColumnCount ()</b> : int
+	<b>getCurrentRecord ()</b> : long
+	<b>getHeaderCount ()</b> : int
+	<b>getHeaders ()</b> : String

### 2.1.2 JavaCsvWriter class :

CsvWriter	
- outputStream	: outputStream
- delimiter	: char
- charset	: charset
- fileName	: String
+ ESCAPE_MODE_BACKSLASH	: int
+ ESCAPE_MODE_DOUBLED	: int
<b>+ CsvWriter ()</b>	
+ getDelimiter ()	: char
+ setDelimiter ()	: char
+ getRecordDelimiter ()	: char
+ setRecordDelimiter ()	: char
+ getTextQualifier ()	: char
+ setTextQualifier ()	: void
+ getUseTextQualifier ()	: boolean
+ setUseTextQualifier ()	: void
+ getEscapeMode ()	: int
+ setEscapeMode ()	: void
+ setComment ()	: void
+ getComment ()	: char
+ getForceQualifier ()	: boolean
+ setForceQualifier ()	: void
+ write ()	: void
+ writeComment ()	: void
+ writeRecord ()	: void
+ endRecord ()	: void
+ flush ()	: void
+ close ()	: void
+ replace ()	: String

## 2.2 JavascriptCsv API :

### 2.2.1 CsvReader class:

During two weeks, we have developed the following methods:

1. CsvReader(fileName, delimiter, charset): Creates a CsvReader object using a file as the data source
2. CsvReader(fileName, delimiter): Creates a CsvReader object using a file as the data source. Uses UTF-8 as the Charset.
3. CsvReader(fileName): Creates a CsvReader object using a file as the data source. Uses a comma as the column delimiter and UTF-8 as the Charset.
4. CsvReader(textStream, delimiter, charset): Constructs a CsvReader object using a TextStream object as the data source.

5. `CsvReader(textStream, charset)`: Constructs a `CsvReader` object using a `TextStream` object as the data source. Uses a comma as the Column delimiter.
6. `char getDelimiter()`: Gets the character being used as the column delimiter. Default is comma, ','.
7. `char getRecordDelimiter()`: gets the character to use as the record delimiter.
8. `char getTextQualifier()`: Gets the character to use as a text qualifier in the data.
9. `boolean getUseTextQualifier()`: returns Whether text qualifiers will be used while parsing or not.
10. `char getComment()`: Gets the character being used as a comment signal.
11. `boolean getUseComments()`: Gets whether comments are being looked for while parsing or not.
12. `int getEscapeMode()`: Gets the current way to escape an occurrence of the text qualifier inside qualified data.
13. `int getColumnCount()`: Gets the count of columns found in this record.
14. `long getCurrentRecord()`: Gets the index of the current record.
15. `String[] getHeaders()`: Returns the header values as a string array.
16. `String get(int columnIndex)`: Returns the current column value for a given column index.
17. `String get(String headerName)`: Returns the current column value for a given column header name.
18. `String getHeader(int columnIndex)`: Returns the column header value for a given column index.
19. `int getIndex(String headerName)`: Gets the corresponding column index for a given column header name.
20. `void close()`: Closes and releases all related resources.

### 2.2.2 **CsvWriter class :**

1. `CsvWriter(fileName, delimiter, charset)`: Creates a `CsvWriter` object using a file as the data destination
2. `CsvWriter(fileName)`: Creates a `CsvWriter` object using a file as the data destination. Uses a comma as the column delimiter and UTF-8 as the Charset.
3. `CsvWriter(textStream, delimiter, charset)`: Creates a `CsvWriter` object using a `TextStream` to write data to.
4. `CsvWriter(textStream, delimiter)`: Creates a `CsvWriter` object using a `TextStream` to write data to. Uses UTF-8 as the Charset.
5. `void setDelimiter(char delimiter)`: Sets the character to use as the column delimiter
6. `void setTextQualifier(char textQualifier)`: Sets the character to use as a text qualifier in the data.
7. `void setComment(char comment)`
8. `void writeConetnt(content, preserveSpaces, header, wstream)`: Writes another column of data to this record.
9. `void writeCont(header, content, wstream)`: Writes another column of data to this record. Does not preserve leading and trailing whitespace in this column of data.
10. `void writeComment(String commentText)`

11. void writeRec(String[] values, boolean preserveSpaces): Writes a new record using the passed in array of values.
12. void writeRecord(String[] values): Writes a new record using the passed in array of values.
13. ChangeDelimiter(newDelim, stream): changes the char delimiter used in the file.
14. void close(): Closes and releases all related resources.

## 2.3 Progression :

The first step of our project -which is still ongoing- is to develop a JavascriptCsv API for the platform Wakanda.

In 15 days the API will be ready, and then we can move to the second step of development which is API JavaScriptExcel for processing Excel files.