

## CSV Reader/Writer - Jacopo Vitale MSc

Generated by Doxygen 1.9.1



---

<b>1 A very tiny simple CSV file reader/writer for C++</b>	<b>1</b>
1.1 Usage: . . . . .	1
<b>2 Data Structure Index</b>	<b>3</b>
2.1 Data Structures . . . . .	3
<b>3 Data Structure Documentation</b>	<b>5</b>
3.1 CSVFile< T > Class Template Reference . . . . .	5
3.1.1 Detailed Description . . . . .	5
3.1.2 Constructor & Destructor Documentation . . . . .	6
3.1.2.1 CSVFile() . . . . .	6
3.1.3 Member Function Documentation . . . . .	6
3.1.3.1 head() . . . . .	6
3.2 CSVRW< T > Class Template Reference . . . . .	6
3.2.1 Detailed Description . . . . .	7
3.2.2 Member Function Documentation . . . . .	7
3.2.2.1 instance() . . . . .	7
3.2.2.2 read_file() . . . . .	7
3.2.2.3 write_file() . . . . .	9
<b>Index</b>	<b>11</b>



# Chapter 1

## A very tiny simple CSV file reader/writer for C++

Not for a professional use, but for just visualize, manipulate and print your comma separated data.

### 1.1 Usage:

First you need to instantiate a reader/writer, suppose you have a `float` file to read:

```
CSVWR<float>* rw = CSVRW::instance(dtype::F)
```

`dtype::F` is an enum variable defined in [CSVWR.h](#) and it needs to manual provide the data type for a better data fetching and memory management (no more needed in C++20).

Once instantiated a reader/writer, a `CSVFile` variable needs to be constructed.

Two examples are needed here.

Suppose we have data to read stored in a local csv file:

```
CSVFile<float>* myfile = new CSVFile<float>() --> empty in this case, it will work as a to-fill-container taking data from our file
```

So now we can call our reader/writer ready to read:

```
'rw->read_file("path/to/file", myfile, true, ',') --> the reader/writer will fill our container
```

Now, `myfile` contains your file data.

You can access to your data typing:

```
myfile->getHeader() // Retrieve columns names as string vector  
myfile->getData() //And you can access like a matrix by [][] operator
```

Second Example: you have your data stored in a matrix of float:

```
float my_float_data[ROWS][COLS]; // Suppose this is the filled matrix  
vector<string> col_names = {"col1", "col2", ... "col3"};  
CSVFile<float>* my_csv_file = new CSVFile<float>(&col_names, &my_float_data, ROWS, COLS)
```

Once built the `CSVFile<float>` object you can, for example manipulate your data and then write out to a file:

```
rw->write_file("/path/to/file.csv", my_csv_file, ',');
```

A message will confirm that output is completed.



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">CSVFile&lt; T &gt;</a> . . . . .	5
<a href="#">CSVrw&lt; T &gt;</a>	
This is the <a href="#">CSVrw</a> (CSV Read & Write) This class can help to read and write CSV files just instanciating this object . . . . .	6





## Chapter 3

# Data Structure Documentation

### 3.1 CSVFile< T > Class Template Reference

```
#include <CSVFile.h>
```

#### Public Member Functions

- [CSVFile](#) ()  
*Construct an empty [CSVFile](#) class object.*
- [CSVFile](#) (std::vector< std::string > header, T \*\*data, int rows, int cols)  
*Construct a [CSVFile](#) starting from local variables.*
- void [head](#) (int heads=5)  
*Print elements to stdout (default is 5 elements)*
- std::vector< std::string > [getHeader](#) ()
- std::vector< std::vector< T > > [getData](#) ()
- void [appendToHeader](#) (std::string element)
- void [appendRowToData](#) (std::vector< T > row)
- int [getDataRows](#) ()  
*Safe getter of number of rows.*
- int [getDataCols](#) ()  
*Safe getter of number of cols.*

#### 3.1.1 Detailed Description

```
template<class T>  
class CSVFile< T >
```

This is [CSVFile](#) class, use this class for manage or create your CSV file. This class has two attributes:

- `header (vector<string>)` : Usually a CSV File contains an header for Columns Names
- `data (vector<vector<T>>)`: The data (numerical) contained in the file.

Constructor:

- No params: Creates empty [CSVFile](#) Object

Constructor with Parameters:

- `vector<string> header`: if you want to build [CSVFile](#) object starting from already existing data
- `T** data` : if you have a matrix of numerical data

Methods:

- `appendToHeader`: Append element to header (kept private)

- `appendRowToData`: Append a vector of values to data matrix (kept private)
- getters

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 CSVFile()

```
template<class T >
CSVFile< T >::CSVFile (
    std::vector< std::string > header,
    T ** data,
    int rows,
    int cols )
```

Construct a `CSVFile` starting from local variables.

##### Parameters

<code>(vector&lt;string&gt;*)</code>	header: string vector containing all columns names
<code>(T**)</code>	data: variable containing numerical data e.g. a float matrix
<code>(int)</code>	rows: data matrix total number of rows
<code>(int)</code>	cols: data matrix total number of cols

### 3.1.3 Member Function Documentation

#### 3.1.3.1 head()

```
template<class T >
void CSVFile< T >::head (
    int heads = 5 )
```

Print elements to stdout (default is 5 elements)

##### Parameters

<code>(int)</code>	heads: number of elements to print
--------------------	------------------------------------

The documentation for this class was generated from the following file:

- `CSVFile.h`

## 3.2 CSVRW< T > Class Template Reference

This is the `CSVRW` (CSV Read & Write) This class can help to read and write CSV files just instantiating this object.  
`#include <CSVRW.h>`

### Public Member Functions

- `CSVRW` (`CSVRW` &other)=delete
  - void `operator=` (const `CSVRW` &)=delete
  - void `read_file` (std::string filepath, `CSVFile`< T > \*file, bool header=true, char delim=',')
- Read a csv file.*

- void `write_file` (std::string filename, `CSVFile< T > *file`, char delim=',')  
*Write a csv file.*

## Static Public Member Functions

- static `CSVRW *instance` (dtypes dt)  
*Instantiate a `CSVRW` class object.*

### 3.2.1 Detailed Description

```
template<class T>
class CSVRW< T >
```

This is the `CSVRW` (CSV Read & Write) This class can help to read and write CSV files just instantiating this object. So this class needs to be instantiated by using this dtype flags. Manual data type control can improve memory space allocation.

#### Attention

This class can only read a file like:

```
|col1 |col2 |col3 |...|col_m | -->header
|val11|val12|val13|...|val_1m| -->data (numeric)
[...]
```

#### Remarks

Columns having categorical data types are not supported

## 3.2.2 Member Function Documentation

### 3.2.2.1 instance()

```
template<class T >
CSVRW< T > * CSVRW< T >::instance (
    dtypes dt ) [static]
```

Instantiate a `CSVRW` class object.

#### Parameters

<i>dt</i>	dtypes::F, dtypes::D, dtypes::I
-----------	------------------------------------

#### Returns

`CSVRW` instance if is not instantiated.

### 3.2.2.2 read\_file()

```
template<class T >
void CSVRW< T >::read_file (
    std::string filepath,
    CSVFile< T > * file,
    bool header = true,
    char delim = ',' )
```

Read a csv file.

## Parameters

<i>(string)</i>	filepath: path/to/file
<i>(CSVFile*)</i>	file: variable containing csv fetched data
<i>(bool)</i>	header: if the file contains an header (columns names)

**3.2.2.3 write\_file()**

```
template<class T >
void CSVRW< T >::write_file (
    std::string filename,
    CSVFile< T > * file,
    char delim = ', ' )
```

Write a csv file.

## Parameters

<i>(string)</i>	filepath: path/to/write (including .csv)
<i>(CSVFile*)</i>	file: variable containing csv data to write
<i>(char)</i>	delim: custom delimiter default is comma

The documentation for this class was generated from the following file:

- CSVRW.h



# Index

- CSVFile
  - CSVFile< T >, [6](#)
- CSVFile< T >, [5](#)
  - CSVFile, [6](#)
  - head, [6](#)
- CSV RW< T >, [6](#)
  - instance, [7](#)
  - read\_file, [7](#)
  - write\_file, [9](#)
- head
  - CSVFile< T >, [6](#)
- instance
  - CSV RW< T >, [7](#)
- read\_file
  - CSV RW< T >, [7](#)
- write\_file
  - CSV RW< T >, [9](#)