CSV Handler

> ***Author : csl***

> ***E-Mail : 3079625093@qq.com***

CSV Handler

1. OverView

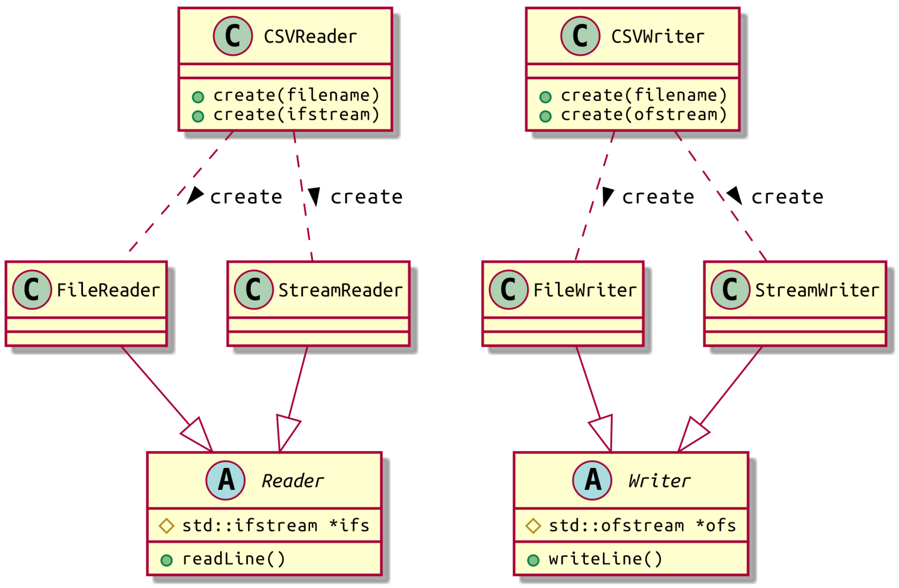
This is a library implemented with cpp macros to read and write CSV files. It is simple and universal.

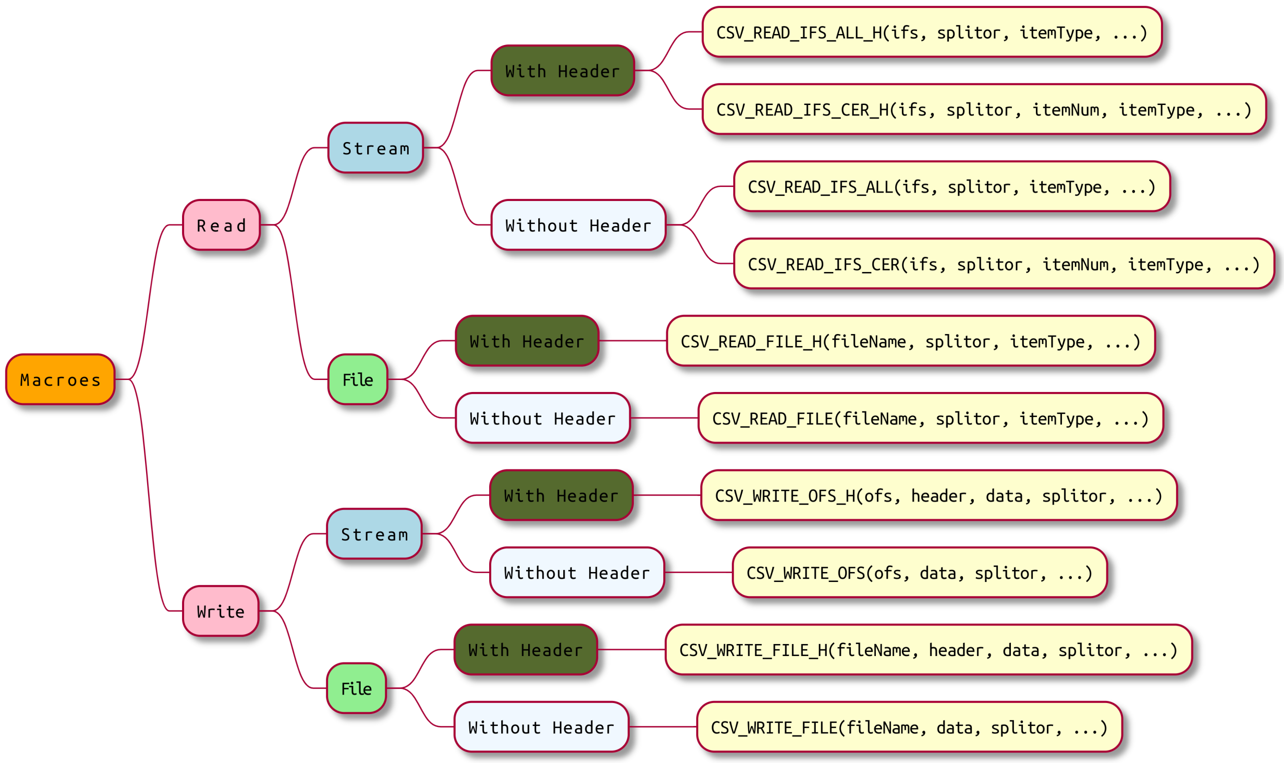
```

  _|_|_|    _|_|_|  _|      _|  _|    _|                            _|  _|
_|        _|        _|      _|  _|    _|    _|_|_|  _|_|_|      _|_|_|  _|    _|_|    _|  _|_|
_|          _|_|    _|      _|  _|_|_|_|  _|    _|  _|    _|  _|    _|  _|  _|_|_|_|  _|_|
_|              _|    _|  _|    _|    _|  _|    _|  _|    _|  _|    _|  _|  _|        _|
  _|_|_|  _|_|_|        _|      _|    _|    _|_|_|  _|    _|    _|_|_|  _|    _|_|_|  _|

```

2. Structure



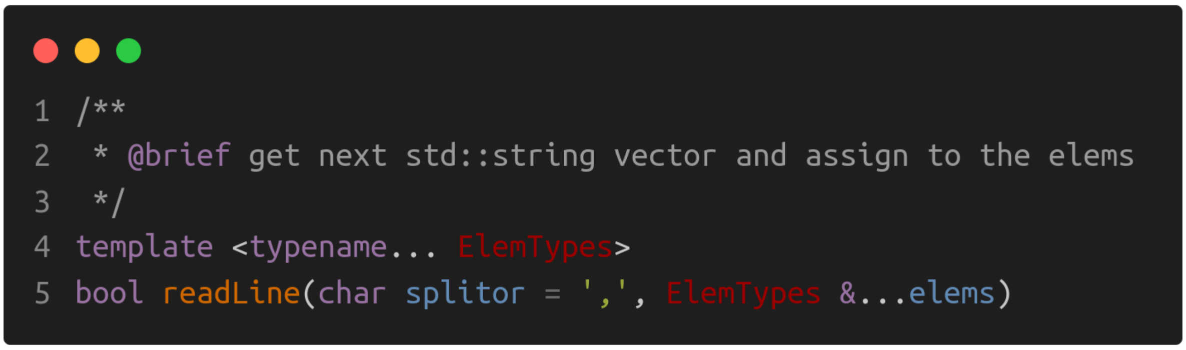


Macroes
Read
Stream
With Header
CSV_READ_IFS_ALL_H(ifs, splitor, itemType, ...)
CSV_READ_IFS_CER_H(ifs, splitor, itemNum, itemType, ...)
Without Header
CSV_READ_IFS_ALL(ifs, splitor, itemType, ...)
CSV_READ_IFS_CER(ifs, splitor, itemNum, itemType, ...)
File
With Header
CSV_READ_FILE_H(fileName, splitor, itemType, ...)
Without Header
CSV_READ_FILE(fileName, splitor, itemType, ...)
Write
Stream
With Header
CSV_WRITE_OFS_H(ofs, header, data, splitor, ...)
Without Header
CSV_WRITE_OFS(ofs, data, splitor, ...)
File
With Header
CSV_WRITE_FILE_H(fileName, header, data, splitor, ...)
Without Header
CSV_WRITE_FILE(fileName, data, splitor, ...)

3. Methods

1. Read CSV

1). class object



```cpp
/**
 * @brief get next std::string vector and assign to the elems
 */
template <typename... ElemTypes>
bool readLine(char splitor = ',', ElemTypes &...elems)
```

- *CSVReader[IFS]*

```cpp
void test_CSVReader_IFS() {
  ns_log::info("test the ns_csv::CSVReader[IFS], file '../data/info.csv'");

  std::ifstream ifs("../data/info.csv");
  ns_csv::CSVReader::Ptr readerIFS = ns_csv::CSVReader::create(ifs);
  int id;
  std::string name;
  float score;
  while (readerIFS->readLine(',', id, name, score)) {
    std::cout << Info(id, name, score) << std::endl;
  }
  ifs.close();
}
```

- *CSVReader[FILE]*

```cpp
void test_CSVReader_FILE() {
  ns_log::info("test the ns_csv::CSVReader[FILE], file '../data/info.csv'");

  ns_csv::CSVReader::Ptr reader = ns_csv::CSVReader::create("../data/info.csv");
  int id;
  std::string name;
  float score;
  while (reader->readLine(',', id, name, score)) {
    std::cout << Info(id, name, score) << std::endl;
  }
}
```

2). macroes

- *CSV_READ_FILE(fileName, splitor, itemType, ...)*

param	describe
fileName	the file name
splitor	the splitor
itemType	the type of the item in the csv file

param	describe
...	the types of the members, it's order is same as the declaration sequence of member variables.
return	std::vector data

```cpp
void test_CSV_READ_FILE()
{
    INFO("test the macro 'CSV_READ_FILE', file '../data/info.csv'");
    auto data = CSV_READ_FILE("../data/info.csv", ',', Info, int, std::string, float);
    vecOutput(data);
}
```

- ***CSV_READ_FILE_H(fileName, splitor, itemType, ...)***

param	describe
fileName	the file name
splitor	the splitor
itemType	the type of the item in the csv file
...	the types of the members, it's order is same as the declaration sequence of member variables.
return	std::pair(std::array<std::string, LabNum>, std::vector) {header, data}

```cpp
void test_CSV_READ_FILE_H()
{
    INFO("test the macro 'CSV_READ_FILE_H', file '../data/refpoint3f.csv'");
    auto data = CSV_READ_FILE_H("../data/refpoint3f.csv", ',', ns_geo::RefPoint3f, uint, float, float, float);
    INFO("header: ", data.first.at(0), ',', data.first.at(1), ',', data.first.at(2), ',', data.first.at(3));
    vecOutput(data.second);
}
```

- ***CSV_READ_IFS_ALL(ifs, splitor, itemType, ...)***

param	describe
ifs	the input fstream
splitor	the splitor
itemType	the type of the item in the csv file
...	the types of the members, it's order is same as the declaration sequence of member variables.
return	std::vector data

```cpp
void test_CSV_READ_IFS_ALL()
{
    INFO("test the macro 'CSV_READ_IFS_ALL', file '../data/info.csv'");
    std::ifstream ifs("../data/info.csv");
    auto data = CSV_READ_IFS_ALL(ifs, ',', Info, int, std::string, float);
    vecOutput(data);
    ifs.close();
}
```

- ***CSV_READ_IFS_ALL_H(ifs, splitor, itemType, ...)***

param	describe
ifs	the input fstream
splitor	the splitor
itemType	the type of the item in the csv file
...	the types of the members, it's order is same as the declaration sequence of member variables.
return	std::pair(std::array<std::string, LabNum>, std::vector) {header, data}

```cpp
void test_CSV_READ_IFS_ALL_H()
{
    INFO("test the macro 'CSV_READ_IFS_ALL_H', file '../data/refpoint3f.csv'");
    std::ifstream ifs("../data/refpoint3f.csv");
    auto data = CSV_READ_IFS_ALL_H(ifs, ',', ns_geo::RefPoint3f, uint, float, float, float);
    INFO("header: ", data.first.at(0), ',', data.first.at(1), ',', data.first.at(2), ',',
data.first.at(3));
    vecOutput(data.second);
    ifs.close();
}
```

- ***CSV_READ_IFS_CER(ifs, splitor, itemNum, itemType, ...)***

param	describe
ifs	the input fstream
splitor	the splitor
itemNum	the number of the items to read
itemType	the type of the item in the csv file
...	the types of the members, it's order is same as the declaration sequence of member variables.
return	std::vector data

```cpp
void test_CSV_READ_IFS_CER()
{
    INFO("test the macro 'CSV_READ_IFS_CER', file '../data/info.csv'");
    std::ifstream ifs("../data/info.csv");
    auto data = CSV_READ_IFS_CER(ifs, ',', 4, Info, int, std::string, float);
    vecOutput(data);
    ifs.close();
}
```

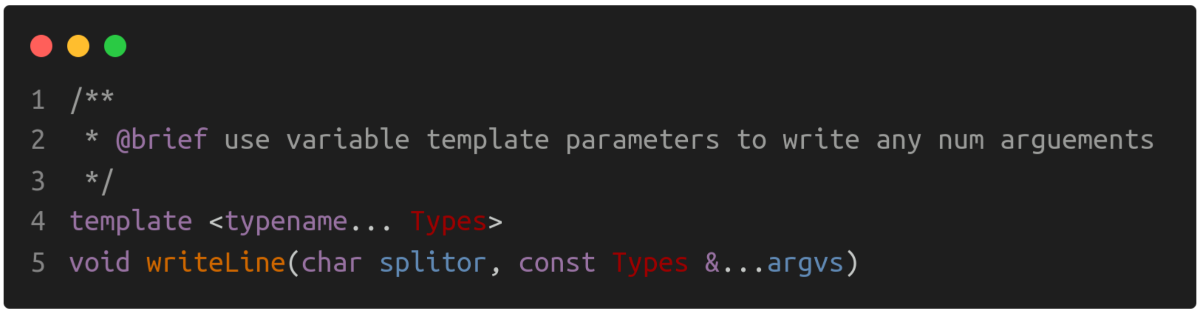
- ***CSV_READ_IFS_CER_H(ifs, splitor, itemNum, itemType, ...)***

param	describe
ifs	the input fstream
splitor	the splitor
itemNum	the number of the items to read
itemType	the type of the item in the csv file
...	the types of the members, it's order is same as the declaration sequence of member variables.
return	std::pair(std::array<std::string, LabNum>, std::vector) {header, data}

```cpp
void test_CSV_READ_IFS_CER_H()
{
    INFO("test the macro 'CSV_READ_IFS_CER_H', file '../data/refpoint3f.csv'");
    std::ifstream ifs("../data/refpoint3f.csv");
    auto data = CSV_READ_IFS_CER_H(ifs, ',', 4, ns_geo::RefPoint3f, uint, float, float, float);
    INFO("header: ", data.first.at(0), ',', data.first.at(1), ',', data.first.at(2), ',', data.first.at(3));
    vecOutput(data.second);
    ifs.close();
}
```

2. *Write CSV*

1). class object



```cpp
/**
 * @brief use variable template parameters to write any num arguements
 */
template <typename... Types>
void writeLine(char splitor, const Types &...argvs)
```

- *CSVWriter[OFS]*

```cpp
void test_CSVWriter_OFS() {
  ns_log::info("test the ns_csv::CSVWriter[OFS], file '../data/info.csv'");

  auto ps = ns_geo::PointSet3f::randomGenerator(10, 0.0f, 1.0f, 0.0f, 1.0f,
                                                 0.0f, 1.0f);
  std::ofstream ofs("../data/point3f.csv");
  ns_csv::CSVWriter::Ptr writer = ns_csv::CSVWriter::create(ofs);
  writer->writeLine(',', "x+z", "x+y", "y-z", "z-y");
  for (const auto &p : ps)
    writer->writeLine(',', p.x(), p.y(), p.z());
  ofs.close();
}
```

- *CSVWriter[FILE]*

```cpp
void test_CSVWriter_FILE() {
  ns_log::info("test the ns_csv::CSVWriter[FILE], file '../data/info.csv'");

  auto ps = ns_geo::PointSet3f::randomGenerator(10, 0.0f, 1.0f, 0.0f, 1.0f,
                                                 0.0f, 1.0f);
  ns_csv::CSVWriter::Ptr writer = ns_csv::CSVWriter::create("../data/point3f.csv");
  writer->writeLine(',', "x+z", "x+y", "y-z", "z-y");
  for (const auto &p : ps)
    writer->writeLine(',', p.x(), p.y(), p.z());
}
```

2). macroes

- *CSV_HEADER(...)*

```cpp
#define CSV_HEADER(...) \
    ARRAY(__VA_ARGS__) { __VA_ARGS__ }
```

- *CSV_ELEM(methods)*

```cpp
#define CSV_ELEM(method) elem.method
```

- *CSV_WRITE_OFS(ofs, data, splitor, itemType, ...)*

param	describe
ofs	the output fstream
data	the data vector
splitor	the splitor
...	the [methods
return	void

```cpp
void test_CSV_WRITE_OFS()
{
    INFO("test the macro 'CSV_WRITE_OFS', file '../data/point3f.csv'");
    auto ps = ns_geo::PointSet3f::randomGenerator(10, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f);
    std::ofstream ofs("../data/point3f.csv");
    CSV_WRITE_OFS(ofs, ps, ',',
                  CSV_ELEM(x()) * CSV_ELEM(z()),
                  CSV_ELEM(x()) + CSV_ELEM(y()),
                  CSV_ELEM(y()) - CSV_ELEM(z()),
                  CSV_ELEM(z()) * CSV_ELEM(y()));
    ofs.close();
}
```

- ***CSV_WRITE_OFS_H(ofs, header, data, splitor, itemType, ...)***

param	describe
ofs	the output fstream
header	the header labels
data	the data vector
splitor	the splitor
...	the [methods
return	void

```cpp
void test_CSV_WRITE_OFS_H()
{
    INFO("test the macro 'CSV_WRITE_OFS_H', file '../data/point3f.csv'");
    auto ps = ns_geo::PointSet3f::randomGenerator(10, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f);
    std::ofstream ofs("../data/point3f.csv");
    CSV_WRITE_OFS_H(ofs, CSV_HEADER("x+z", "x+y", "y-z", "z-y"), ps, ',',
                    CSV_ELEM(x()) * CSV_ELEM(z()),
                    CSV_ELEM(x()) + CSV_ELEM(y()),
                    CSV_ELEM(y()) - CSV_ELEM(z()),
                    CSV_ELEM(z()) * CSV_ELEM(y()));
    auto header = CSV_HEADER("x+z", "x+y", "y-z", "z-y");
    ofs.close();
}
```

- ***CSV_WRITE_FILE(fileName, data, splitor, itemType, ...)***

param	describe
fileName	the file name
data	the data vector
splitor	the splitor
...	the [methods
return	void

```cpp
void test_CSV_WRITE_FILE()
{
    INFO("test the macro 'CSV_WRITE_FILE', file '../data/point3f.csv'");
    auto ps = ns_geo::PointSet3f::randomGenerator(10, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f);
    CSV_WRITE_FILE("../data/point3f.csv", ps, ',',
                   CSV_ELEM(x()) * CSV_ELEM(z()),
                   CSV_ELEM(x()) + CSV_ELEM(y()),
                   CSV_ELEM(y()) - CSV_ELEM(z()),
                   CSV_ELEM(z()) * CSV_ELEM(y()));
}
```

- ***CSV_WRITE_FILE_H(fileName, header, data, splitor, itemType, ...)***

param	describe
fileName	the file name
header	the header labels
data	the data vector
splitor	the splitor
...	the [methods
return	void

```cpp
void test_CSV_WRITE_FILE_H()
{
    INFO("test the macro 'CSV_WRITE_FILE_H', file '../data/point3f.csv'");
    auto ps = ns_geo::PointSet3f::randomGenerator(10, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f);
    CSV_WRITE_FILE_H("../data/point3f.csv", CSV_HEADER("x+z", "x+y", "y-z", "z-y"), ps, ','
                     CSV_ELEM(x()) * CSV_ELEM(z()),
                     CSV_ELEM(x()) + CSV_ELEM(y()),
                     CSV_ELEM(y()) - CSV_ELEM(z()),
                     CSV_ELEM(z()) * CSV_ELEM(y()));
}
```

4. Files

info.csv

point3f.csv

refpoint3f.csv