Data Frame

0.1

Generated by Doxygen 1.9.1

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 DF Namespace Reference

Classes

- class DataFrame

  DataFrame is class for parsing data in a given file with a give delimeter (default is comma ','). all data will be saved as string wich user can later covert to desired type.

4.2 std Namespace Reference

Namespaces

- shorts

  namespace for introducting shortnames

4.3 std::shorts Namespace Reference

namespace for introducting shortnames

Typedefs

- using V_string = vector< string >
- using VV_string = vector< V_string >
- using V_any = vector< any >
- using VV_any = vector< vector< any > >
- using Data = unordered_map< string, V_string >
- using V_double = vector< double >
- using V_int = vector< int >
- using V_pair_ints = std::vector< std::pair< int, int > >

4.3.1 Detailed Description

namespace for introducting shortnames

4.3.2 Typedef Documentation

4.3.2.1 Data

```
using std::shorts::Data = typedef unordered_map<string, V_string>
```

Definition at line 39 of file ReadFiles.hpp.

4.3.2.2 V_any

```
using std::shorts::V_any = typedef vector<any>
```

Definition at line 37 of file ReadFiles.hpp.

4.3.2.3 V_double

```
using std::shorts::V_double = typedef vector<double>
```

Definition at line 40 of file ReadFiles.hpp.

4.3.2.4 V_int

```
using std::shorts::V_int = typedef vector<int>
```

Definition at line 41 of file ReadFiles.hpp.

4.3.2.5 V_pair_ints

```
using std::shorts::V_pair_ints = typedef std::vector<std::pair<int, int> >
```

Definition at line 42 of file ReadFiles.hpp.

4.3.2.6 V_string

```
using std::shorts::V_string = typedef vector<string>
```

Definition at line 35 of file ReadFiles.hpp.

4.3.2.7 VV_any

```
using std::shorts::VV_any = typedef vector<vector<any> >
```

Definition at line 38 of file ReadFiles.hpp.

4.3.2.8 VV_string

```
using std::shorts::VV_string = typedef vector<V_string>
```

Definition at line 36 of file ReadFiles.hpp.

Chapter 5

Class Documentation

5.1 DF::DataFrame Class Reference

DataFrame is class for parsing data in a given file with a give delimeter (default is comma ','). all data will be saved as string wich user can later covert to desired type.

`#include <ReadFiles.hpp>`

Public Member Functions

- int get_n_rows () const

  Get the number of rows of a given data.
- int get_n_cols () const

  Get the number cols of a given data.
- void set_headers (std::shorts::V_string const &v_hdrs)

  Set the headers with user provided vector of strings.
- std::shorts::V_string get_headers () const

  Get the headers.
- std::shorts::V_string get_by_header (std::string const &hdr)
- DataFrame copy () const

  to copy current data into new data frame
- DataFrame copy_by_headers (std::shorts::V_string const &v_hdrs)

  copy the requested data by provided headers name as vector of strings into a new data frame
- void remove_duplications (std::string const &hdr)

  get a given header and only keep the first occurance and remove the remaning rows
- void read_files (std::string_view path, char delim=',', bool is_first_col_header=true, std::shorts::V_string v_hdrs={})

  read files
- void read_text (std::string const &text, std::shorts::V_pair_ints const &v_cols_start_length, bool is_first_col_header=true, std::shorts::V_string v_hdrs={})
- void read_text_whitespace (std::string const &text, bool is_first_col_header=true, std::shorts::V_string v_hdrs={})

  read text with whitespaces
- void head (unsigned long long n=5)

  print n first rows off all columns
- void swap_cols_pos (std::string first_hdr, std::string second_hdr)

swap two columns with their header with each others

- void add_col (std::shorts::V_string const &v_values, std::string hdr="new_col")

  add a column to the end of the dataframe based on a provided vector of string
- void add_col_of (std::string const &value, std::string hdr="new_col")

  dd a column to the end of the dataframe based on a provided value for all rows
- void append (std::vector< DataFrame > &&v_dfs)

  check if headers are the same then append the values
- void clear ()

  to clear headers and data
- std::shorts::V_string & operator[ ] (std::string hdr)

  subscript operator
- void write (std::string_view path, char delimiter=',')

  write the dataframe in a file with given path and delimiter
- void save_as_csv (std::string_view path)

  saving dataframe in comma separted format file

Public Attributes

- unsigned long long n_rows

  number of rows
- unsigned long long n_cols

  number of columns
- std::unordered_map< int, int > mising_values

  an unorderd maps for missing values

Private Member Functions

- std::shorts::V_string read_lines (std::string_view path)
- std::shorts::V_string read_lines (std::string const &text)
- std::shorts::V_string parse_line_whitespace (std::string const &line)
- std::shorts::V_string parse_line (std::string const &line, char delim)
- std::shorts::V_string parse_line (std::string const &line, std::shorts::V_pair_ints const &v_cols_start_ends)
- void fill_data_whitespace (std::shorts::V_string const &v_lines, bool is_first_col_header=true, std::shorts::V_string v_hdrs={})
- void fill_data (std::shorts::V_string const &v_lines, char delim=',', bool is_first_col_header=true, std::shorts::V_string v_hdrs={})
- void fill_data (std::shorts::V_string const &v_lines, std::shorts::V_pair_ints const &v_cols_start_length, bool is_first_col_header=true, std::shorts::V_string v_hdrs={})
- void insert_col (std::shorts::V_string const &values, std::string hdr)

Private Attributes

- std::shorts::Data data
- std::shorts::V_string headers

5.1.1 Detailed Description

DataFrame is class for parsing data in a given file with a give delimeter (default is comma ','). all data will be saved as string wich user can later covert to desired type.

Definition at line 54 of file ReadFiles.hpp.

5.1.2 Member Function Documentation

5.1.2.1 add_col()

```
void DF::DataFrame::add_col (
            std::shorts::V_string const & v_values,
            std::string hdr = "new_col" )
```

add a column to the end of the dataframe based on a provided vector of string

Parameters

v_strs	vector of values to add
hdr	given header name (default new: new_col), if the header is ther it would modified the header name

Definition at line 514 of file ReadFiles.cpp.

```
516 {
517     insert_col(values, hdr);
518 }
```

5.1.2.2 add_col_of()

```
void DF::DataFrame::add_col_of (
            std::string const & value,
            std::string hdr = "new_col" )
```

dd a column to the end of the dataframe based on a provided value for all rows

Parameters

value	value to be add to all rows
hdr	given header name (default new: new_col), if the header is ther it would modified the header name

Definition at line 520 of file ReadFiles.cpp.

```
521 {
522     std::shorts::V_string values(data[headers[0]].size(), value);
523     insert_col(values, hdr);
524 }
```

5.1.2.3 append()

```
void DF::DataFrame::append (
            std::vector< DataFrame > && v_dfs )
```

check if headers are the same then append the values

Parameters

v_dfs	

Definition at line 532 of file ReadFiles.cpp.

```
533 {
534     if(v_dfs.empty()) return;
535
536     for(auto& curr_df : v_dfs)
537     {
538         if(headers == curr_df.get_headers())
539         {
540             for(auto curr_hdr : headers)
541             {
542                 data[curr_hdr].insert(data[curr_hdr].end(), curr_df.data.at(curr_hdr).begin(),
      curr_df.data.at(curr_hdr).end());
543             }
544         }
545         curr_df.clear();
546     }
547     n_rows = data[headers[0]].size();
548     n_cols = headers.size();
549 }
```

5.1.2.4 clear()

```
void DF::DataFrame::clear ( )
```

to clear headers and data

Definition at line 526 of file ReadFiles.cpp.

```
527 {
528     headers.clear();
529     data.clear();
530 }
```

5.1.2.5 copy()

```
DataFrame DF::DataFrame::copy ( ) const
```

to copy current data into new data frame

Returns

DataFrame new data frame

5.1.2.6 copy_by_headers()

```
DF::DataFrame DF::DataFrame::copy_by_headers (
            std::shorts::V_string const & v_hdrs )
```

copy the requested data by provided headers name as vector of strings into a new data frame

Parameters

v_hdrs	

Returns

DataFrame new data frame

Definition at line 472 of file ReadFiles.cpp.

```
473 {
474     DF::DataFrame new_df;
475     new_df.n_cols = v_hdrs.size();
476     new_df.n_rows = n_rows;
477     new_df.headers = v_hdrs;
478 
479     for(auto const& hdr : v_hdrs)
480     {
481         new_df.data[hdr] = data[hdr];
482     }
483 
484     return new_df;
485 }
```

5.1.2.7 fill_data() [1/2]

```
void DF::DataFrame::fill_data (
              std::shorts::V_string const & v_lines,
              char delim = ',',
              bool is_first_col_header = true,
              std::shorts::V_string v_hdrs = {} )  [private]
```

Definition at line 190 of file ReadFiles.cpp.

```
191 {
192     std::shorts::VV_string vv_strs;
193 
194     for(auto const& line : lines)
195     {
196         std::shorts::V_string v_str_tmp = parse_line(line, delim);
197         vv_strs.emplace_back(v_str_tmp);
198     }
199 
200     // init n_rows and n_cols
201     n_rows = vv_strs.size();
202     n_cols = vv_strs[0].size();
203 
204     headers.resize(n_cols);
205 
206     // initializing headers
207     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
208     {
209         if(is_first_col_header)
210         {
211             headers[i_col] = vv_strs[0][i_col];
212         }
213         else if(v_hdrs.size() > 0)
214         {
215             if(v_hdrs.size() == n_cols)
216             {
217                 headers[i_col] = v_hdrs[i_col];
218             }
219             else
220             {
221                 throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: number of provided
       headers does not match with the number of columns in the data"));
222             }
223         }
224         else
225         {
226             headers[i_col] = std::to_string(i_col + 1);
227         }
```

```
228     }
229
230     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
231     {
232         std::shorts::V_string values;
233         for(unsigned long long i_row{is_first_col_header}; i_row < n_rows; ++i_row)
234         {
235             if(vv_strs[i_row].size() != n_cols)
236             {
237                 throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: inconsistent number of
    columns, check row {}", i_row + 1));
238             }
239
240             values.emplace_back(vv_strs[i_row][i_col]);
241
242             // check for the missig values
243             // missing values are empty string, NA, and NAN
244             if(vv_strs[i_row][i_col].empty() ||
245                vv_strs[i_row][i_col] == "NA" ||
246                vv_strs[i_row][i_col] == "NAN")
247             {
248                 mising_values.insert({i_row, i_col});
249             }
250         }
251
252         data[headers[i_col]] = values;
253     }
254 }
```

5.1.2.8 fill_data() [2/2]

```
void DF::DataFrame::fill_data (
            std::shorts::V_string const & v_lines,
            std::shorts::V_pair_ints const & v_cols_start_length,
            bool is_first_col_header = true,
            std::shorts::V_string v_hdrs = {} )  [private]
```

Definition at line 256 of file ReadFiles.cpp.

```
257 {
258     std::shorts::VV_string vv_strs;
259     for(auto const& line : lines)
260     {
261         auto v_str_tmp = parse_line(line, v_cols_start_length);
262         vv_strs.emplace_back(v_str_tmp);
263     }
264
265     // init n_rows and n_cols
266     n_rows = vv_strs.size();
267     n_cols = vv_strs[0].size();
268
269     headers.resize(n_cols);
270
271     // initializing headers
272     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
273     {
274         if(is_first_col_header)
275         {
276             headers[i_col] = vv_strs[0][i_col];
277         }
278         else if(v_hdrs.size() > 0)
279         {
280             if(v_hdrs.size() == n_cols)
281             {
282                 headers[i_col] = v_hdrs[i_col];
283             }
284             else
285             {
286                 throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: number of provided
    headers does not match with the number of columns in the data"));
287             }
288         }
289         else
290         {
291             headers[i_col] = std::to_string(i_col + 1);
292         }
293     }
294
```

```
295     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
296     {
297         std::shorts::V_string values;
298         for(unsigned long long i_row{is_first_col_header}; i_row < n_rows; ++i_row)
299         {
300             if(vv_strs[i_row].size() != n_cols)
301             {
302                 throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: inconsistent number of
    columns, check row {}", i_row + 1));
303             }
304 
305             values.emplace_back(vv_strs[i_row][i_col]);
306 
307             // check for the missig values
308             // missing values are empty string, NA, and NAN
309             if(vv_strs[i_row][i_col].empty() ||
310                vv_strs[i_row][i_col] == "NA" ||
311                vv_strs[i_row][i_col] == "NAN")
312             {
313                 mising_values.insert({i_row, i_col});
314             }
315         }
316 
317         data[headers[i_col]] = values;
318     }
319 }
```

5.1.2.9 fill_data_whitespace()

```
void DF::DataFrame::fill_data_whitespace (
              std::shorts::V_string const & v_lines,
              bool is_first_col_header = true,
              std::shorts::V_string v_hdrs = {} )  [private]
```

Definition at line 124 of file ReadFiles.cpp.

```
125 {
126     std::shorts::VV_string vv_strs;
127 
128     for(auto const& line : v_lines)
129     {
130         std::shorts::V_string v_str_tmp = parse_line_whitespace(line);
131         vv_strs.emplace_back(v_str_tmp);
132     }
133 
134     //initialize n_rows and n_cols
135     n_rows = vv_strs.size();
136     n_cols = vv_strs[0].size();
137 
138     headers.resize(n_cols);
139 
140     // initializing headers
141     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
142     {
143         if(is_first_col_header)
144         {
145             headers[i_col] = vv_strs[0][i_col];
146         }
147         else if(v_hdrs.size() > 0)
148         {
149             if(v_hdrs.size() == n_cols)
150             {
151                 headers[i_col] = v_hdrs[i_col];
152             }
153             else
154             {
155                 throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: number of provided
    headers does not match with the number of columns in the data"));
156             }
157         }
158         else
159         {
160             headers[i_col] = std::to_string(i_col + 1);
161         }
162     }
163 
164     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
165     {
```

```
166         std::shorts::V_string values;
167         for(unsigned long long i_row{is_first_col_header}; i_row < n_rows; ++i_row)
168         {
169             if(vv_strs[i_row].size() != n_cols)
170             {
171                 throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: inconsistent number of
    columns, check row {}", i_row + 1));
172             }
173 
174             values.emplace_back(vv_strs[i_row][i_col]);
175 
176             // check for the missig values
177             // missing values are empty string, NA, and NAN
178             if(vv_strs[i_row][i_col] == ""   ||
179                vv_strs[i_row][i_col] == "NA" ||
180                vv_strs[i_row][i_col] == "NAN")
181             {
182                 mising_values.insert({i_row, i_col});
183             }
184         }
185 
186         data[headers[i_col]] = values;
187     }
188 }
```

5.1.2.10 get_by_header()

```
std::shorts::V_string DF::DataFrame::get_by_header (
            std::string const & hdr )
```

Definition at line 467 of file ReadFiles.cpp.

```
468 {
469     return data[hdr];
470 }
```

5.1.2.11 get_headers()

```
std::shorts::V_string DF::DataFrame::get_headers ( ) const
```

Get the headers.

Returns

std::shorts::V_string headers

Definition at line 457 of file ReadFiles.cpp.

```
458 {
459     return headers;
460 }
```

5.1.2.12 get_n_cols()

```
int DF::DataFrame::get_n_cols ( ) const
```

Get the number cols of a given data.

Returns

int number of columns

Definition at line 13 of file ReadFiles.cpp.

```
14 {
15     return n_cols;
16 }
```

5.1.2.13 get_n_rows()

```
int DF::DataFrame::get_n_rows ( ) const
```

Get the number of rows of a given data.

Returns

int number of rows

Definition at line 18 of file ReadFiles.cpp.

```
19 {
20     return n_rows;
21 }
```

5.1.2.14 head()

```
void DF::DataFrame::head (
            unsigned long long n = 5 )
```

print n first rows off all columns

Parameters

n	

Definition at line 339 of file ReadFiles.cpp.

```
340 {
341     if(n > n_rows)
342     {
343         fmt::print(fg(fmt::color::yellow),"Warning: number of row requested ({}) to be printed is bigger
    than number of available rows in data ({})\n",
344         n, data[headers[0]].size());
345         if(n_rows > 5) n = 5;
346         else n = n_rows;
347     }
348
349     struct winsize w;
350
351     // Get the terminal size
352     if (ioctl(STDOUT_FILENO, TIOCGWINSZ, &w) == -1) {
353         std::cerr « "Error getting terminal size\n";
354         exit(EXIT_FAILURE);
355     }
356
357     int col_width = (w.ws_col - 30)/(headers.size() + 1);
358     if(col_width < 5)
359     {
360         fmt::print(fg(fmt::color::yellow), "to many data to show. the printed data may be not be
    informative.\n");
361     }
362
363
364     fmt::print("{:{}}|", " ", col_width);
365     for(auto const& curr_hdr : headers)
366     {
367         fmt::print(fmt::emphasis::bold | fg(fmt::color::green), "{:^{}}|", curr_hdr.substr(0,col_width),
    col_width);
368     }
369
370     std::string sub_separator(col_width, '-');
371     sub_separator += '+';
372
373     std::string separator;
374     separator.reserve(n_cols * sub_separator.size()+1);
```

```
375
376     for (size_t i = 0; i < n_cols+1; ++i)
377     {
378         separator += sub_separator;
379     }
380
381     fmt::println("\n{}",std::string(separator));
382
383     // std::cout « '\n';
384
385     if(n <= 10)
386     {
387         for(unsigned long long i{0}; i < n; ++i)
388         {
389             fmt::print(fg(fmt::color::green),"|{:^{}}|", std::to_string(i+1).substr(0,col_width),
    col_width - 1);
390             for(auto const& curr_hdr : headers)
391             {
392                 const auto& value = data[curr_hdr][i];
393                 if (value.empty() || value == "NA" || value == "NAN")
394                 {
395                     fmt::print(bg(fmt::color::red),"{:^{}}", data[curr_hdr][i].substr(0,col_width),
    col_width);
396                     std::cout « "|";
397                 }
398                 else
399                 {
400                     fmt::print("{:^{}}|", data[curr_hdr][i].substr(0,col_width), col_width);
401                 }
402
403             }
404
405             std::cout « "\n";
406         }
407     }
408     else
409     {
410         for(unsigned long long i{0}; i < 5; ++i)
411         {
412             fmt::print(fg(fmt::color::green),"|{:^{}}|", std::to_string(i+1).substr(0,col_width),
    col_width - 1);
413             for(auto const& curr_hdr : headers)
414             {
415                 const auto& value = data[curr_hdr][i];
416                 if (value.empty() || value == "NA" || value == "NAN")
417                 {
418                     fmt::print(bg(fmt::color::red),"{:^{}}", data[curr_hdr][i].substr(0,col_width),
    col_width);
419                     std::cout « "|";
420                 }
421                 else
422                 {
423                     fmt::print("{:^10}|", data[curr_hdr][i].substr(0,col_width), col_width );
424                 }
425
426             }
427
428             std::cout « "\n";
429         }
430
431         std::cout « "\n\t.\n\t.\n\t.\n\n";
432
433         for(unsigned long long i{n-5}; i < n; ++i)
434         {
435             fmt::print(fg(fmt::color::green),"|{:^{}}|", std::to_string(i+1).substr(0,col_width),
    col_width - 1);
436             for(auto const& curr_hdr : headers)
437             {
438                 const auto& value = data[curr_hdr][i];
439                 if (value.empty() || value == "NA" || value == "NAN")
440                 {
441                     fmt::print(bg(fmt::color::red),"{:^{}}", data[curr_hdr][i].substr(0,col_width),
    col_width);
442                     std::cout « "|";
443                 }
444                 else
445                 {
446                     fmt::print("{:^{}}|", data[curr_hdr][i].substr(0,col_width), col_width);
447                 }
448
449             }
450
451             std::cout « "\n";
452         }
453     }
454     fmt::println("{}",std::string(separator));
455 }
```

5.1.2.15 insert_col()

```
void DF::DataFrame::insert_col (
            std::shorts::V_string const & values,
            std::string hdr )  [private]
```

Definition at line 498 of file ReadFiles.cpp.

```
499 {
500     auto [_it, inserted] = data.insert({hdr, values});
501
502     int n = 1;
503     std::string original_hdr = hdr;
504
505     while (!inserted)
506     {
507         hdr = fmt::format("{}_{}", original_hdr, n++);
508         std::tie(_it, inserted) = data.insert({hdr, values});
509     }
510     headers.push_back(hdr);
511     n_cols++;
512 }
```

5.1.2.16 operator[]()

```
std::shorts::V_string & DF::DataFrame::operator[] (
            std::string hdr )
```

subscript operator

Parameters

hdr	

Returns

std::shorts::V_string

Definition at line 551 of file ReadFiles.cpp.

```
552 {
553     return data[hdr];
554 }
```

5.1.2.17 parse_line() [1/2]

```
std::shorts::V_string DF::DataFrame::parse_line (
            std::string const & line,
            char delim )  [private]
```

Definition at line 82 of file ReadFiles.cpp.

```
83 {
84     std::shorts::V_string v_strs;
```

```
85
86      std::istringstream iss(line);
87      std::string cell;
88
89      while(std::getline(iss, cell, delim))
90      {
91          cell.erase(std::remove(cell.begin(), cell.end(), '\"'), cell.end());
92          cell.erase(std::remove(cell.begin(), cell.end(), ' '), cell.end());
93          v_strs.emplace_back(cell);
94      }
95
96      return v_strs;
97  }
```

5.1.2.18 parse_line() [2/2]

```
std::shorts::V_string DF::DataFrame::parse_line (
            std::string const & line,
            std::shorts::V_pair_ints const & v_cols_start_ends )  [private]
```

Definition at line 99 of file ReadFiles.cpp.

```
100 {
101     std::shorts::V_string v_strs;
102
103     for(auto const& [start, end] : v_cols_start_ends)
104     {
105         std::string cell;
106         try
107         {
108             cell = line.substr(start, end);
109         }
110         catch(...)
111         {
112             cell = "NA";
113         }
114
115         // auto cell = line.substr(start, end);
116         cell.erase(std::remove(cell.begin(), cell.end(), '\"'), cell.end());
117         cell.erase(std::remove(cell.begin(), cell.end(), ' '), cell.end());
118         v_strs.push_back(cell);
119     }
120
121     return v_strs;
122 }
```

5.1.2.19 parse_line_whitespace()

```
std::shorts::V_string DF::DataFrame::parse_line_whitespace (
            std::string const & line )  [private]
```

Definition at line 65 of file ReadFiles.cpp.

```
66 {
67     std::shorts::V_string v_strs;
68     std::istringstream iss(line);
69     std::string cell;
70
71     while(iss » cell)
72     {
73         cell.erase(std::remove(cell.begin(), cell.end(), '\"'), cell.end());
74         v_strs.emplace_back(cell);
75     }
76
77     return v_strs;
78 }
```

5.1.2.20 read_files()

```
void DF::DataFrame::read_files (
            std::string_view path,
            char delim = ',',
            bool is_first_col_header = true,
            std::shorts::V_string v_hdrs = {} )
```

read files

Parameters

path	path to input file
delim	delimiter for parsing the input file
is_first_col_header	boolean

Definition at line 321 of file ReadFiles.cpp.

```
322 {
323     auto lines = read_lines(path);
324     fill_data(lines, delim, is_first_col_header, v_hdrs);
325 }
```

5.1.2.21 read_lines() [1/2]

```
std::shorts::V_string DF::DataFrame::read_lines (
            std::string const & text )  [private]
```

Definition at line 47 of file ReadFiles.cpp.

```
48 {
49     std::istringstream iss(text);
50 
51     std::string line;
52     std::shorts::V_string v_strs;
53 
54 
55     while(std::getline(iss, line))
56     {
57         if(line.size() == 0) continue;
58         line.erase(std::remove(line.begin(), line.end(), '\r'),line.end());
59         v_strs.emplace_back(line);
60     }
61 
62     return v_strs;
63 }
```

5.1.2.22 read_lines() [2/2]

```
std::shorts::V_string DF::DataFrame::read_lines (
            std::string_view path )  [private]
```

Definition at line 23 of file ReadFiles.cpp.

```
24 {
25     std::ifstream ifs(path.data());
26 
27     if(ifs.fail())
28     {
29         throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: unable to read file {}.\nPlease
       check your input.", path));
```

```
30     }
31
32
33     std::string line;
34     std::shorts::V_string v_strs;
35
36
37     while(std::getline(ifs, line))
38     {
39         if(line.size() == 0) continue;
40         line.erase(std::remove(line.begin(), line.end(), '\r'),line.end());
41         v_strs.emplace_back(line);
42     }
43
44     return v_strs;
45 }
```

5.1.2.23 read_text()

```
void DF::DataFrame::read_text (
             std::string const & text,
             std::shorts::V_pair_ints const & v_cols_start_length,
             bool is_first_col_header = true,
             std::shorts::V_string v_hdrs = {} )
```

Parameters

path	std::string_view
v_cols_length	
is_first_col_header	

Definition at line 327 of file ReadFiles.cpp.

```
328 {
329     auto lines = read_lines(text);
330     fill_data(lines, v_cols_start_length, is_first_col_header, v_hdrs);
331 }
```

5.1.2.24 read_text_whitespace()

```
void DF::DataFrame::read_text_whitespace (
             std::string const & text,
             bool is_first_col_header = true,
             std::shorts::V_string v_hdrs = {} )
```

read text with whitespaces

Parameters

text	the whole text we need to parse
is_first_col_header	check if the first column should be used as column or not
v_hdrs	provided headers

Definition at line 333 of file ReadFiles.cpp.

```
334 {
335     auto lines = read_lines(text);
336     fill_data_whitespace(lines, is_first_col_header, v_hdrs);
337 }
```

5.1.2.25 remove_duplications()

```
void DF::DataFrame::remove_duplications (
            std::string const & hdr )
```

get a given header and only keep the first occurance and remove the remaning rows

Parameters

hdr	header to check the duplication

5.1.2.26 save_as_csv()

```
void DF::DataFrame::save_as_csv (
            std::string_view path )
```

saving dataframe in comma separted format file

Parameters

path	

Definition at line 579 of file ReadFiles.cpp.

```
580 {
581     write(path /*,delimiter=','*/);
582 }
```

5.1.2.27 set_headers()

```
void DF::DataFrame::set_headers (
            std::shorts::V_string const & v_hdrs )
```

Set the headers with user provided vector of strings.

Parameters

v_hdrs	provided headers from users

Definition at line 462 of file ReadFiles.cpp.

```
463 {
464     headers = v_hdrs;
465 }
```

5.1.2.28 swap_cols_pos()

```
void DF::DataFrame::swap_cols_pos (
            std::string first_hdr,
            std::string second_hdr )
```

swap two columns with their header with each others

Parameters

first_hdr	std::string
second_hdr	std::string

Definition at line 487 of file ReadFiles.cpp.

```
488 {
489     std::string tmp_hdr;
490     std::shorts::V_string tmp_vals;
491     auto first_it = std::find(headers.begin(), headers.end(), first_hdr);
492     auto second_it = std::find(headers.begin(), headers.end(), second_hdr);
493     swap(headers[first_it - headers.begin()], headers[second_it - headers.begin()]);
494 
495     std::swap(data[first_hdr], data[second_hdr]);
496 }
```

5.1.2.29 write()

```
void DF::DataFrame::write (
            std::string_view path,
            char delimiter = ',' )
```

write the dataframe in a file with given path and delimiter

Parameters

path	
delimiter	

Definition at line 556 of file ReadFiles.cpp.

```
557 {
558     fmt::ostream out = fmt::output_file(path.data());
559 
560     for(auto const& curr_hdr : headers)
561     {
562         out.print("{}{}", curr_hdr.substr(0,10), delimiter);
563     }
564     out.print("\n");
565 
566     for(unsigned long long i{0}; i < n_rows; ++i)
567     {
568         for(auto const& curr_hdr : headers)
```

```
569         {
570             const auto& value = data[curr_hdr][i];
571             out.print("{}{}", value, delimiter);
572 
573         }
574 
575         out.print("\n");
576     }
577 }
```

5.1.3 Member Data Documentation

5.1.3.1 data

`std::shorts::Data DF::DataFrame::data [private]`

Definition at line 224 of file ReadFiles.hpp.

5.1.3.2 headers

`std::shorts::V_string DF::DataFrame::headers [private]`

Definition at line 225 of file ReadFiles.hpp.

5.1.3.3 mising_values

`std::unordered_map<int, int> DF::DataFrame::mising_values`

an unorderd maps for missing values

Definition at line 74 of file ReadFiles.hpp.

5.1.3.4 n_cols

`unsigned long long DF::DataFrame::n_cols`

number of columns

Definition at line 68 of file ReadFiles.hpp.

5.1.3.5 n_rows

`unsigned long long DF::DataFrame::n_rows`

number of rows

Definition at line 62 of file ReadFiles.hpp.

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

- include/ReadFiles.hpp
- src/ReadFiles.cpp

Chapter 6

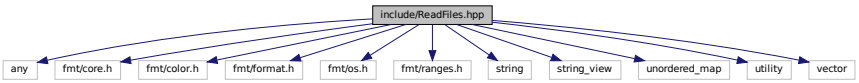
File Documentation

6.1 include/ReadFiles.hpp File Reference

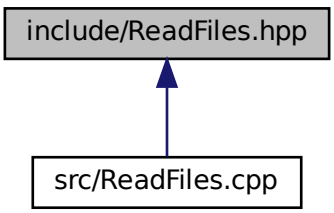
A class for reading files with different delimiters.

```
#include <any>
#include "fmt/core.h"
#include "fmt/color.h"
#include "fmt/format.h"
#include "fmt/os.h"
#include "fmt/ranges.h"
#include <string>
#include <string_view>
#include <unordered_map>
#include <utility>
#include <vector>
```

Include dependency graph for ReadFiles.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class DF::DataFrame

  DataFrame is class for parsing data in a given file with a give delimeter (default is comma ','). all data will be saved as string wich user can later covert to desired type.

Namespaces

- std
- std::shorts

  namespace for introducting shortnames
- DF

Typedefs

- using std::shorts::V_string = vector< string >
- using std::shorts::VV_string = vector< V_string >
- using std::shorts::V_any = vector< any >
- using std::shorts::VV_any = vector< vector< any > >
- using std::shorts::Data = unordered_map< string, V_string >
- using std::shorts::V_double = vector< double >
- using std::shorts::V_int = vector< int >
- using std::shorts::V_pair_ints = std::vector< std::pair< int, int > >

6.1.1 Detailed Description

A class for reading files with different delimiters.

Author

Naeim Moafinejad (snmoafinejad@iimcb.gov.pl, s.naeim.moafi.n@gmail.com)

Version

0.1

Date

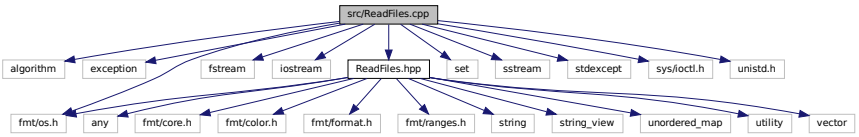
2024-10-29

Copyright

6.2 src/ReadFiles.cpp File Reference

```
#include <algorithm>
#include <exception>
#include <fmt/os.h>
#include <fstream>
#include <iostream>
#include "ReadFiles.hpp"
#include <set>
#include <sstream>
#include <stdexcept>
#include <sys/ioctl.h>
#include <unistd.h>
```

Include dependency graph for ReadFiles.cpp:



Functions

- int main ()

6.2.1 Function Documentation

6.2.1.1 main()

```
int main ( )
```

Definition at line 585 of file ReadFiles.cpp.

```
586 {
587     std::string software_name = R"( ____        _        _____
588 |  _ \  __ _| |_ __ _|  ___| __ __ _ _ __ ___   ___
589 | | | |/ _` | __/ _` | |_ | '__/ _` | '_ ` _ \ / _ \
590 | |_| | (_| | || (_| |  _|| | | (_| | | | | | |  __/
591 |____/ \__,_|\__\__,_|_|  |_|  \__,_|_| |_| |_|\___|)";
592
593         std::cout << software_name << std::endl << std::endl;
594     // DF::DataFrame df;
595
596
597     // df.read_files("test.txt", '\t');
598     // df.head();
599     // std::cout << '\n';
600
601     // auto df2 = df.copy_by_headers({"id", "age", "disease"});
602     // df2.head();
603     // std::cout << '\n';
604
605     // df2.swap_cols_pos("age", "disease");
606     // df2.head();
607     // std::cout << '\n';
608
609     // df.read_files("test.txt", '\t', false);
```

```
610     // df.head();
611     // std::cout « '\n';
612
613     // df.read_files("test.txt", '\t', false, {"1.0000", "2.0000", "3.0000", "4.0000", "5.0000"});
614     // df.head();
615     // std::cout « '\n';
616
617     // df.head(600);
618     // std::cout « '\n';
619
620     DF::DataFrame df3;
621     std::vector<std::string> v_hdrs {"group_PDB", "id", "label_atom_id", "alt_id", "label_comp_id",
      "label_asym_id",
622                                      "label_seq_id", "Cartn_x", "Cartn_y", "Cartn_z", "occupancy",
623                                      "B_iso_or_equiv", "type_symbol", "charge"};
624
625     std::shorts::V_pair_ints fields_intervals = {{0,6}, {6,5}, {12,4}, {16,1}, {17,3}, {20,2}, {22,4},
      {30,8}, {38,8}, {46,8}, {54,6}, {60,6}, {76,2}, {78,2}};
626
627     // std::string text{"HETATM    1  PG  GTP A   1      24.181  32.064  27.670  0.10 24.73           P
      \nHETATM    2  O1G GTP A   1      24.342  33.433  27.064  0.10 37.56           O  \nHETATM    3  O2G
      GTP A   1      24.519  32.013  29.136  0.10 32.46           O  \n"};
628     std::string text = R"(HETATM    1  PG  GTP A   1      24.181  32.064  27.670  0.10 24.73           P

629 HETATM    2  O1G GTP A   1      24.342  33.433  27.064  0.10 37.56           O
630 HETATM    3  O2G GTP A   1      24.519  32.013  29.136  0.10 32.46           O
631 HETATM    4  O3G GTP A   1      25.048  31.062  26.907  1.00 47.91           O
632 HETATM    5  O3B GTP A   1      22.644  31.665  27.526  1.00 33.11           O )";
633
634     df3.read_text(text, fields_intervals, false, v_hdrs);
635     df3.head();
636
637     std::string text2 = R"(HETATM 1   P  PG    . GTP A 1 1  ? 24.181  32.064 27.670 0.10 24.73 ? 1   GTP
      A PG    1
638 HETATM 2   O  O1G   . GTP A 1 1  ? 24.342  33.433 27.064 0.10 37.56 ? 1   GTP A O1G   1
639 HETATM 3   O  O2G   . GTP A 1 1  ? 24.519  32.013 29.136 0.10 32.46 ? 1   GTP A O2G   1
640 HETATM 4   O  O3G   . GTP A 1 1  ? 25.048  31.062 26.907 1.00 47.91 ? 1   GTP A O3G   1
641 HETATM 5   O  O3B   . GTP A 1 1  ? 22.644  31.665 27.526 1.00 33.11 ? 1   GTP A O3B   1
642 HETATM 6   P  PB    . GTP A 1 1  ? 21.944  30.914 26.290 1.00 30.44 ? 1   GTP A PB    1
643 HETATM 7   O  O1B   . GTP A 1 1  ? 20.481  30.888 26.504 1.00 30.75 ? 1   GTP A O1B   1
644 HETATM 8   O  O2B   . GTP A 1 1  ? 22.544  29.597 26.037 1.00 31.25 ? 1   GTP A O2B   1
645 HETATM 9   O  O3A   . GTP A 1 1  ? 22.230  31.881 25.061 1.00 27.41 ? 1   GTP A O3A   1
646 HETATM 10  P  PA    . GTP A 1 1  ? 21.357  32.886 24.136 1.00 28.14 ? 1   GTP A PA    1
647 HETATM 11  O  O1A   . GTP A 1 1  ? 22.257  33.379 23.051 1.00 29.88 ? 1   GTP A O1A   1
648 HETATM 12  O  O2A   . GTP A 1 1  ? 20.694  33.906 24.985 1.00 28.72 ? 1   GTP A O2A   1
649 HETATM 13  O  "O5'" A GTP A 1 1  ? 20.284  31.879 23.543 0.50 23.11 ? 1   GTP A "O5'" 1
650 HETATM 14  O  "O5'" B GTP A 1 1  ? 20.244  31.934 23.525 0.50 23.80 ? 1   GTP A "O5'" 1
651 HETATM 15  C  "C5'" A GTP A 1 1  ? 19.131  32.316 22.825 0.50 21.95 ? 1   GTP A "C5'" 1
652 HETATM 16  C  "C5'" B GTP A 1 1  ? 19.049  32.533 22.993 0.50 20.46 ? 1   GTP A "C5'" 1
653 HETATM 17  C  "C4'" A GTP A 1 1  ? 18.305  31.114 22.431 0.50 18.93 ? 1   GTP A "C4'" 1
654 HETATM 18  C  "C4'" B GTP A 1 1  ? 18.162  31.423 22.482 0.50 19.32 ? 1   GTP A "C4'" 1
655 HETATM 19  O  "O4'" A GTP A 1 1  ? 19.032  30.287 21.495 0.50 16.95 ? 1   GTP A "O4'" 1
656 HETATM 20  O  "O4'" B GTP A 1 1  ? 18.804  30.735 21.386 0.50 17.14 ? 1   GTP A "O4'" 1
657 HETATM 21  C  "C3'" A GTP A 1 1  ? 17.949  30.186 23.587 0.50 16.54 ? 1   GTP A "C3'" 1
658 HETATM 22  C  "C3'" B GTP A 1 1  ? 17.857  30.361 23.529 0.50 18.33 ? 1   GTP A "C3'" 1
659 HETATM 23  O  "O3'" A GTP A 1 1  ? 16.747  30.614 24.223 0.50 17.14 ? 1   GTP A "O3'" 1
660 HETATM 24  O  "O3'" B GTP A 1 1  ? 16.664  30.669 24.225 0.50 18.17 ? 1   GTP A "O3'" 1
661 HETATM 25  C  "C2'" A GTP A 1 1  ? 17.816  28.834 22.929 0.50 16.16 ? 1   GTP A "C2'" 1 )";
662
663     DF::DataFrame df4;
664
665     df4.read_text_whitespace(text2, /*is_first_cols_header=*/false /*headers=default*/);
666     df4.head();
667
668
669
670
671     return EXIT_SUCCESS;
672 }
```

Index