

Data Frame

0.1

Generated by Doxygen 1.9.1

1 Namespace Index	1
1.1 Namespace List	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Namespace Documentation	7
4.1 DF Namespace Reference	7
4.2 std Namespace Reference	7
4.3 std::shorts Namespace Reference	7
4.3.1 Detailed Description	8
4.3.2 Typedef Documentation	8
4.3.2.1 Data	8
4.3.2.2 V_any	8
4.3.2.3 V_double	8
4.3.2.4 V_int	8
4.3.2.5 V_pair_ints	8
4.3.2.6 V_string	9
4.3.2.7 VV_any	9
4.3.2.8 VV_string	9
5 Class Documentation	11
5.1 DF::DataFrame Class Reference	11
5.1.1 Detailed Description	12
5.1.2 Member Function Documentation	12
5.1.2.1 copy()	12
5.1.2.2 copy_by_headers()	12
5.1.2.3 fill_data() [1/2]	13
5.1.2.4 fill_data() [2/2]	14
5.1.2.5 get_by_header()	15
5.1.2.6 get_headers()	15
5.1.2.7 get_n_cols()	16
5.1.2.8 get_n_rows()	16
5.1.2.9 head()	16
5.1.2.10 parse_line() [1/2]	17
5.1.2.11 parse_line() [2/2]	17
5.1.2.12 read_files()	18
5.1.2.13 read_lines() [1/2]	18
5.1.2.14 read_lines() [2/2]	19
5.1.2.15 read_text()	19
5.1.2.16 remove_duplications()	19

5.1.2.17 set_headers()	20
5.1.2.18 swap_cols_pos()	20
5.1.3 Member Data Documentation	20
5.1.3.1 data	20
5.1.3.2 headers	21
5.1.3.3 missing_values	21
5.1.3.4 n_cols	21
5.1.3.5 n_rows	21
6 File Documentation	23
6.1 include/ReadFiles.hpp File Reference	23
6.1.1 Detailed Description	24
6.2 src/ReadFiles.cpp File Reference	25
6.2.1 Function Documentation	25
6.2.1.1 main()	25
Index	27

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

DF	7
std	7
std::shorts Namespace for introducing shortnames	7

Chapter 2

Class Index

2.1 Class List

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

[DF::DataFrame](#)

[DataFrame](#) is class for parsing data in a given file with a give delimiter (default is comma ','). all data will be saved as string wich user can later covert to desired type [11](#)

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

include/ ReadFiles.hpp	
A class for reading files with different delimiters	23
src/ ReadFiles.cpp	25

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 delimiter (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 introducing shortnames

4.3 std::shorts Namespace Reference

namespace for introducing 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 introducing 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 delimiter (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 [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

Public Attributes

- unsigned long long [n_rows](#)
number of rows
- unsigned long long [n_cols](#)
number of columns
- `std::unordered_map< int, int >` [missing_values](#)
an unordered 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](#) (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](#) ([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={})

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 delimiter (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 copy()

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

to copy current data into new data frame

Returns

[DataFrame](#) new data frame

5.1.2.2 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

<code>v_hdrs</code>	
---------------------	--

Returns

[DataFrame](#) new data frame

Definition at line 300 of file ReadFiles.cpp.

```

301 {
302     DF::DataFrame new_df;
303     new_df.n_cols = v_hdrs.size();
304     new_df.n_rows = n_rows;
305     new_df.headers = v_hdrs;
306
307     for(auto const& hdr : v_hdrs)
308     {
309         new_df.data[hdr] = data[hdr];
310     }
311
312     return new_df;
313 }
```

5.1.2.3 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 97 of file ReadFiles.cpp.

```

98 {
99     std::shorts::VV_string vv_strs;
100
101     for(auto const& line : lines)
102     {
103         std::shorts::V_string v_str_tmp = parse_line(line, delim);
104         vv_strs.emplace_back(v_str_tmp);
105     }
106
107     // init n_rows and n_cols
108     n_rows = vv_strs.size();
109     n_cols = vv_strs[0].size();
110
111     headers.resize(n_cols);
112
113     // initializing headers
114     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
115     {
116         if(is_first_col_header)
117         {
118             headers[i_col] = vv_strs[0][i_col];
119         }
120         else if(v_hdrs.size() > 0)
121         {
122             if(v_hdrs.size() == n_cols)
123             {
124                 headers[i_col] = v_hdrs[i_col];
125             }
126             else
127             {
128                 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"));
129             }
130         }
131         else
132         {
133             headers[i_col] = std::to_string(i_col + 1);
134         }
135     }
136 }
```

```

135     }
136
137     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
138     {
139         std::shorts::V_string values;
140         for(unsigned long long i_row{is_first_col_header}; i_row < n_rows; ++i_row)
141         {
142             if(vv_strs[i_row].size() != n_cols)
143             {
144                 throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: inconsistent number of
columns, check row {}", i_row + 1));
145             }
146
147             values.emplace_back(vv_strs[i_row][i_col]);
148
149             // check for the missig values
150             // missing values are empty string, NA, and NAN
151             if(vv_strs[i_row][i_col] == "" ||
152                vv_strs[i_row][i_col] == "NA" ||
153                vv_strs[i_row][i_col] == "NAN")
154             {
155                 missing_values.insert({i_row, i_col});
156             }
157         }
158
159         data[headers[i_col]] = values;
160     }
161 }

```

5.1.2.4 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 163 of file ReadFiles.cpp.

```

164 {
165     std::shorts::VV_string vv_strs;
166     for(auto const& line : lines)
167     {
168         auto v_str_tmp = parse_line(line, v_cols_start_length);
169         vv_strs.emplace_back(v_str_tmp);
170     }
171
172     // init n_rows and n_cols
173     n_rows = vv_strs.size();
174     n_cols = vv_strs[0].size();
175
176     headers.resize(n_cols);
177
178     // initializing headers
179     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
180     {
181         if(is_first_col_header)
182         {
183             headers[i_col] = vv_strs[0][i_col];
184         }
185         else if(v_hdrs.size() > 0)
186         {
187             if(v_hdrs.size() == n_cols)
188             {
189                 headers[i_col] = v_hdrs[i_col];
190             }
191             else
192             {
193                 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"));
194             }
195         }
196         else
197         {
198             headers[i_col] = std::to_string(i_col + 1);
199         }
200     }
201 }

```

```

202     for(unsigned long long i_col{0}; i_col < n_cols; ++i_col)
203     {
204         std::shorts::V_string values;
205         for(unsigned long long i_row{is_first_col_header}; i_row < n_rows; ++i_row)
206         {
207             if(vv_strs[i_row].size() != n_cols)
208             {
209                 throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: inconsistent number of
columns, check row {}", i_row + 1));
210             }
211
212             values.emplace_back(vv_strs[i_row][i_col]);
213
214             // check for the missig values
215             // missing values are empty string, NA, and NAN
216             if(vv_strs[i_row][i_col].empty() ||
217                vv_strs[i_row][i_col] == " " ||
218                vv_strs[i_row][i_col] == "NA" ||
219                vv_strs[i_row][i_col] == "NAN")
220             {
221                 missing_values.insert({i_row, i_col});
222             }
223         }
224
225         data[headers[i_col]] = values;
226     }
227 }

```

5.1.2.5 get_by_header()

```

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

```

Definition at line 295 of file ReadFiles.cpp.

```

296 {
297     return data[hdr];
298 }

```

5.1.2.6 get_headers()

```

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

```

Get the headers.

Returns

`std::shorts::V_string` headers

Definition at line 290 of file ReadFiles.cpp.

```

291 {
292     return headers;
293 }

```

5.1.2.7 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 12 of file ReadFiles.cpp.

```
13 {
14     return n_cols;
15 }
```

5.1.2.8 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 17 of file ReadFiles.cpp.

```
18 {
19     return n_rows;
20 }
```

5.1.2.9 head()

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

print n first rows off all columns

Parameters

<i>n</i>	
----------	--

Definition at line 241 of file ReadFiles.cpp.

```
242 {
243     if(n > n_rows)
244     {
245         fmt::print(fg(fmt::color::yellow),"Warning: number of row requested ({} to be printed is bigger
than number of available rows in data ({})\n",
246             n, data[headers[0]].size());
247         if(n_rows > 5) n = 5;
248         else n = n_rows;
249     }
```

```

250
251     for(auto const& curr_hdr : headers)
252     {
253         fmt::print(fmt::emphasis::bold | fg(fmt::color::green), "{:^15}|" , curr_hdr);
254     }
255
256     std::string sub_separator = std::string(15, '-') + '+';
257     std::string separator = "";
258
259     for(size_t i{0}; i < n_cols; ++i )
260     {
261         separator += sub_separator;
262     }
263     // fmt::println("\n{}", std::string(headers.size() * 16, '-'));
264     fmt::println("\n{}", std::string(separator));
265
266     // std::cout << '\n';
267
268     for(unsigned long long i{0}; i < n; ++i)
269     {
270         for(auto const& curr_hdr : headers)
271         {
272             if(data[curr_hdr][i] == "" ||
273                data[curr_hdr][i] == "NA" ||
274                data[curr_hdr][i] == "NAN")
275             {
276                 fmt::print(bg(fmt::color::red), "{:^15}", data[curr_hdr][i] /*data[curr_hdr][i]*/);
277                 std::cout << "|";
278             }
279             else
280             {
281                 fmt::print("{:^15}|" , data[curr_hdr][i] /*data[curr_hdr][i]*/);
282             }
283         }
284
285         std::cout << "\n";
286     }
287 }
288 }

```

5.1.2.10 parse_line() [1/2]

```

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

```

Definition at line 65 of file ReadFiles.cpp.

```

66 {
67     std::shorts::V_string v_strs;
68
69     std::istringstream iss(line);
70     std::string cell;
71
72     while(std::getline(iss, cell, delim))
73     {
74         cell.erase(std::remove(cell.begin(), cell.end(), '\\'), cell.end());
75         cell.erase(std::remove(cell.begin(), cell.end(), ' '), cell.end());
76         v_strs.emplace_back(cell);
77     }
78
79     return v_strs;
80 }

```

5.1.2.11 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 82 of file ReadFiles.cpp.

```

83 {
84     std::shorts::V_string v_strs;
85
86     for(auto const& [start, end] : v_cols_start_ends)
87     {
88         auto cell = line.substr(start, end);
89         cell.erase(std::remove(cell.begin(), cell.end(), '\\'), cell.end());
90         cell.erase(std::remove(cell.begin(), cell.end(), ' '), cell.end());
91         v_strs.push_back(cell);
92     }
93
94     return v_strs;
95 }
```

5.1.2.12 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

<i>path</i>	path to input file
<i>delim</i>	delimiter for parsing the input file
<i>is_first_col_header</i>	boolean

Definition at line 229 of file ReadFiles.cpp.

```

230 {
231     auto lines = read_lines(path);
232     fill_data(lines, delim, is_first_col_header, v_hdrs);
233 }
```

5.1.2.13 read_lines() [1/2]

```

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

Definition at line 46 of file ReadFiles.cpp.

```

47 {
48     std::istringstream iss(text);
49
50     std::string line;
51     std::shorts::V_string v_strs;
52
53     while(std::getline(iss, line))
54     {
55         if(line.size() == 0) continue;
56         line.erase(std::remove(line.begin(), line.end(), '\\r'), line.end());
57         v_strs.emplace_back(line);
58     }
59
60     return v_strs;
61 }
62 }
```

5.1.2.14 read_lines() [2/2]

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

Definition at line 22 of file ReadFiles.cpp.

```
23 {
24     std::ifstream ifs(path.data());
25
26     if(ifs.fail())
27     {
28         throw std::runtime_error(fmt::format(fg(fmt::color::red), "Error: unable to read file {}.\\nPlease
check your input.", path));
29     }
30
31
32     std::string line;
33     std::shorts::V_string v_strs;
34
35
36     while(std::getline(ifs, line))
37     {
38         if(line.size() == 0) continue;
39         line.erase(std::remove(line.begin(), line.end(), '\\r'), line.end());
40         v_strs.emplace_back(line);
41     }
42
43     return v_strs;
44 }
```

5.1.2.15 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

<i>path</i>	std::string_view
<i>v_cols_length</i>	
<i>is_first_col_header</i>	

Definition at line 235 of file ReadFiles.cpp.

```
236 {
237     auto lines = read_lines(text);
238     fill_data(lines, v_cols_start_length, is_first_col_header, v_hdrs);
239 }
```

5.1.2.16 remove_duplications()

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

get a given header and only keep the first occurrence and remove the remaining rows

Parameters

<i>hdr</i>	header to check the duplication
------------	---------------------------------

5.1.2.17 set_headers()

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

Set the headers with user provided vector of strings.

Parameters

<i>v_hdrs</i>	provided headers from users
---------------	-----------------------------

5.1.2.18 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

<i>first_hdr</i>	std::string
<i>second_hdr</i>	std::string

Definition at line 315 of file ReadFiles.cpp.

```
316 {
317     std::string tmp_hdr;
318     std::shorts::V_string tmp_vals;
319     auto first_it = std::find(headers.begin(), headers.end(), first_hdr);
320     auto second_it = std::find(headers.begin(), headers.end(), second_hdr);
321     swap(headers[first_it - headers.begin()], headers[second_it - headers.begin()]);
322
323     std::swap(data[first_hdr], data[second_hdr]);
324 }
```

5.1.3 Member Data Documentation**5.1.3.1 data**

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

Definition at line 163 of file ReadFiles.hpp.

5.1.3.2 headers

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

Definition at line 164 of file ReadFiles.hpp.

5.1.3.3 missing_values

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

an unordered 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

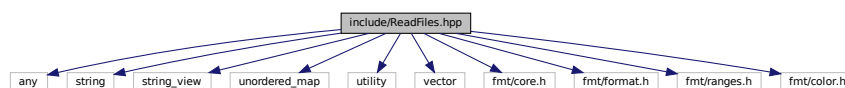
File Documentation

6.1 include/ReadFiles.hpp File Reference

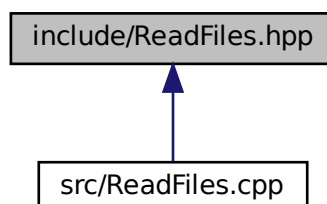
A class for reading files with different delimiters.

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

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 delimiter (default is comma ','). all data will be saved as string wich user can later covert to desired type.

Namespaces

- [std](#)
- [std::shorts](#)
namespace for introducing 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

Copyright (c) 2024


```

357     std::cout << '\n';
358
359     df.head(600);
360     std::cout << '\n';
361
362     DF::DataFrame df3;
363     std::vector<std::string> v_hdrs {"group_PDB", "id", "label_atom_id", "label_comp_id",
    "label_asym_id",
364                                     "label_seq_id", "Cartn_x", "Cartn_y", "Cartn_z", "occupancy",
365                                     "B_iso_or_equiv", "type_symbol", "charge"};
366
367     std::shorts::V_pair_ints fields_intervals = {{0,6}, {6,5}, {12,4}, {17,3}, {20,2}, {22,4}, {30,8},
    {38,8}, {46,8}, {54,6}, {60,6}, {76,2}, {78,2}};
368
369     // 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"};
370     std::string text = R"(HETATM    1  PG  GTP  A    1          24.181  32.064  27.670  0.10 24.73          P
371 HETATM    2  O1G  GTP  A    1          24.342  33.433  27.064  0.10 37.56          O
372 HETATM    3  O2G  GTP  A    1          24.519  32.013  29.136  0.10 32.46          O
373 HETATM    4  O3G  GTP  A    1          25.048  31.062  26.907  1.00 47.91          O
374 HETATM    5  O3B  GTP  A    1          22.644  31.665  27.526  1.00 33.11          O )";
375
376     df3.read_text(text, fields_intervals, false, v_hdrs);
377     df3.head();
378
379
380     return EXIT_SUCCESS;
381 }

```

Index

copy
 DF::DataFrame, [12](#)
copy_by_headers
 DF::DataFrame, [12](#)

Data
 std::shorts, [8](#)

data
 DF::DataFrame, [20](#)

DF, [7](#)
DF::DataFrame, [11](#)
 copy, [12](#)
 copy_by_headers, [12](#)
 data, [20](#)
 fill_data, [13](#), [14](#)
 get_by_header, [15](#)
 get_headers, [15](#)
 get_n_cols, [15](#)
 get_n_rows, [16](#)
 head, [16](#)
 headers, [20](#)
 missing_values, [21](#)
 n_cols, [21](#)
 n_rows, [21](#)
 parse_line, [17](#)
 read_files, [18](#)
 read_lines, [18](#)
 read_text, [19](#)
 remove_duplications, [19](#)
 set_headers, [20](#)
 swap_cols_pos, [20](#)

fill_data
 DF::DataFrame, [13](#), [14](#)

get_by_header
 DF::DataFrame, [15](#)

get_headers
 DF::DataFrame, [15](#)

get_n_cols
 DF::DataFrame, [15](#)

get_n_rows
 DF::DataFrame, [16](#)

head
 DF::DataFrame, [16](#)

headers
 DF::DataFrame, [20](#)

include/ReadFiles.hpp, [23](#)

main
 ReadFiles.cpp, [25](#)

missing_values
 DF::DataFrame, [21](#)

n_cols
 DF::DataFrame, [21](#)

n_rows
 DF::DataFrame, [21](#)

parse_line
 DF::DataFrame, [17](#)

read_files
 DF::DataFrame, [18](#)

read_lines
 DF::DataFrame, [18](#)

read_text
 DF::DataFrame, [19](#)

ReadFiles.cpp
 main, [25](#)

remove_duplications
 DF::DataFrame, [19](#)

set_headers
 DF::DataFrame, [20](#)

src/ReadFiles.cpp, [25](#)

std, [7](#)
std::shorts, [7](#)
 Data, [8](#)
 V_any, [8](#)
 V_double, [8](#)
 V_int, [8](#)
 V_pair_ints, [8](#)
 V_string, [8](#)
 VV_any, [9](#)
 VV_string, [9](#)

swap_cols_pos
 DF::DataFrame, [20](#)

V_any
 std::shorts, [8](#)

V_double
 std::shorts, [8](#)

V_int
 std::shorts, [8](#)

V_pair_ints
 std::shorts, [8](#)

V_string
 std::shorts, [8](#)

VV_any

std::shorts, [9](#)
VV_string
std::shorts, [9](#)