C CSV Parser

Generated by Doxygen 1.9.3

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 include/csvparser.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
```

Functions

- void csv_read (const char *filename, char ****data, char delim, size_t(*data_dims)[2], bool has_headers)
- void csv_read_column_by_index (const char *filename, size_t column_index, char ***data, char delim, size_t *data_rows, bool has_headers)
- void csv_read_column_by_name (const char *filename, const char *column_name, char ***data, char delim, size_t *data_rows)
- void csv_data_to_int (char ***data, size_t(*data_dims)[2], int ***int_data)
- void csv_data_to_float (char ***data, size_t(*data_dims)[2], float ***float_data)
- void csv_column_to_int (char **data, size_t data_rows, int **int_data)
- void csv_column_to_float (char **data, size_t data_rows, float **float_data)
- void csv_free (char ****data, size_t data_dims[2])
- void csv_free_int (int ***data, size_t data_rows)
- void csv_free_float (float ***data, size_t data_rows)
- void csv_free_column (char ***data, size_t data_rows)
- void csv_free_column_int (int **data)
- void csv_free_column_float (float **data)

2.1.1 Function Documentation

2.1.1.1 csv_column_to_float()

```
void csv_column_to_float (
            char ** data,
            size_t data_rows,
            float ** float_data )
```

Parameters

data	A char** pointer to data loaded with csv_read_column_by_name() or csv_read_column_by_index().
data_rows	How many rows are present in the data.
float_data	A float* pointer passed by address to allocate and store the casted floats from data.

Example:

```
#include "csvparser.h"
int main() {
    char** data = NULL;
    float* float_data = NULL;
    size_t data_rows;
    csv_read_column_by_index("../test.csv", 0, &data, ',', &data_rows, true);
    csv_column_to_float(data, data_rows, &float_data);
    printf("String Data:\n");
    for (size_t i = 0; i < data_rows; ++i)
        printf("%s\n", data[i]);
    printf("\nFloat Data:\n");
    for (size_t i = 0; i < data_rows; ++i)
        printf("%f\n", float_data[i] + 1.0);
    // don't forget to free all the memory allocated to store the strings
    csv_free_column(&data, data_rows);
    csv_free_column_float(&float_data);
    return 0;
}
```

Output:

```
String Data:
10
20
30
Float Data:
11.000000
21.000000
31.000000
```

2.1.1.2 csv_column_to_int()

```
void csv_column_to_int (
            char ** data,
            size_t data_rows,
            int ** int_data )
```

Parameters

data	A char** pointer to data loaded with csv_read_column_by_name() or csv_read_column_by_index().
data_rows	How many rows are present in the data.
int_data	An int* pointer passed by address to allocate and store the casted integers from data.

Example:

```
#include "csvparser.h"
int main() {
    char** data = NULL;
    int* int_data = NULL;
    size_t data_rows;
    csv_read_column_by_index("../test.csv", 0, &data, ',', &data_rows, true);
    csv_column_to_int(data, data_rows, &int_data);
    printf("String Data:\n");
    for (size_t i = 0; i < data_rows; ++i)
        printf("%s\n", data[i]);
    printf("\nInt Data:\n");
    for (size_t i = 0; i < data_rows; ++i)
        printf("%d\n", int_data[i] + 1);
    // don't forget to free all the memory allocated to store the strings
```

```
    csv_free_column(&data, data_rows);
    csv_free_column_int(&int_data);
    return 0;
}
```

Output:

```
String Data:
10
20
30
Int Data:
11
21
31
```

2.1.1.3 csv_data_to_float()

```
void csv_data_to_float (
              char *** data,
              size_t(*) data_dims[2],
              float *** float_data )
```

Parameters

data	A char*** pointer to data loaded with csv_read().
data_dims	Array specifying dimensions of the data with the 0th index counting the rows and 1st index counting the columns
float_data	A float** pointer passed by address to allocate and store the casted floats from data.

Example:

```
#include "csvparser.h"
int main() {
    char*** data = NULL;
    float** float_data = NULL;
    size_t data_dims[2];
    csv_read("../test.csv", &data, ',', &data_dims, true);
    csv_data_to_float(data, &data_dims, &float_data);
    printf("String Data:\n");
    for (size_t i = 0; i < data_dims[0]; ++i)
    {
         for (size_t j = 0; j < data_dims[1]; ++j)
            printf("%s ", data[i][j]);
        printf("\n");
    }
    printf("\nFloat Data:\n");
    for (size_t i = 0; i < data_dims[0]; ++i)
    {
        for (size_t j = 0; j < data_dims[1]; ++j)
            printf("%f ", float_data[i][j]);
        printf("\n");
    }
    // don't forget to free all the memory allocated to store the strings
    csv_free(&data, data_dims);
    csv_free_float(&float_data, data_dims[0]);
    return 0;
}
```

Output:

```
String Data:
10 "bob,joe,kyle" 10.33
20 jim 8.11
30 kyle "13.52,20.111"
Float Data:
10.000000 0.000000 10.330000
20.000000 0.000000 8.110000
30.000000 0.000000 0.000000
```

2.1.1.4 csv_data_to_int()

```
void csv_data_to_int (
            char *** data,
            size_t(*) data_dims[2],
            int *** int_data )
```

Convert CSV data (char∗∗∗) to integers (int∗∗).

Parameters

data	A char∗∗∗ pointer to data loaded with csv_read().
data_dims	Array specifying dimensions of the data with the 0th index counting the rows and 1st index counting the columns
int_data	An int∗∗ pointer passed by address to allocate and store the casted integers from data. Example:

```
#include "csvparser.h"
int main() {
    char*** data = NULL;
    int** int_data = NULL;
    size_t data_dims[2];
    csv_read("../test.csv", &data, ',', &data_dims, true);
    csv_data_to_int(data, &data_dims, &int_data);
    printf("String Data:\n");
    for (size_t i = 0; i < data_dims[0]; ++i)
    {
        for (size_t j = 0; j < data_dims[1]; ++j)
        {
            printf("%s ", data[i][j]);
        }
        printf("\n");
    }
    printf("\nInt Data:\n");
    for (size_t i = 0; i < data_dims[0]; ++i)
    {
        for (size_t j = 0; j < data_dims[1]; ++j)
        {
            printf("%d ", int_data[i][j]);
        }
        printf("\n");
    }
    // don't forget to free all the memory allocated to store the strings
    csv_free(&data, data_dims);
    csv_free_int(&int_data, data_dims[0]);
    return 0;
}
```

Output:

```
String Data:
10 "bob,joe,kyle" 10.33
20 jim 8.11
30 kyle "13.52,20.111"
Int Data:
10 0 10
20 0 8
30 0 0
```

2.1.1.5 csv_free()

```
void csv_free (
            char **** data,
            size_t data_dims[2] )
```

Free the memory allocated to data after reading a CSV file. This MUST be done if you intend on using the same pointer to read a different file.

Parameters

data	The address to a char*** pointer holding the CSV data loaded by csv_read().
data_dims	Array of data dimensions with the 0th index counting the rows and 1st index counting the columns.

2.1.1.6 csv_free_column()

```
void csv_free_column (
            char *** data,
            size_t data_rows )
```

Free the memory allocated to data after reading a CSV file. This MUST be done if you intend on using the same pointer to read a different file.

Parameters

data	The address to a char** pointer holding the CSV data loaded by csv_read_column().
data_rows	How many rows to free from the data.

2.1.1.7 csv_free_column_float()

```
void csv_free_column_float (
            float ** data )
```

Free the memory allocated to data after reading a CSV file. This MUST be done if you intend on using the same pointer to read a different file.

Parameters

data	The address to a float* pointer holding the CSV data loaded by csv_read_column_float().
data_rows	How many rows to free from the data.

2.1.1.8 csv_free_column_int()

```
void csv_free_column_int (
            int ** data )
```

Free the memory allocated to data after reading a CSV file. This MUST be done if you intend on using the same pointer to read a different file.

Parameters

data	The address to a int∗ pointer holding the CSV data loaded by csv_read_column_int().
data_rows	How many rows to free from the data.

2.1.1.9 csv_free_float()

```
void csv_free_float (
            float *** data,
            size_t data_rows )
```

Free the memory allocated to data after reading a CSV file. This MUST be done if you intend on using the same pointer to read a different file.

Parameters

data	The address to a float∗∗ pointer holding the CSV data loaded by csv_data_to_float().
data_rows	How many rows to free from the data.

2.1.1.10 csv_free_int()

```
void csv_free_int (
            int *** data,
            size_t data_rows )
```

Free the memory allocated to data after reading a CSV file. This MUST be done if you intend on using the same pointer to read a different file.

Parameters

data	The address to an int∗∗ pointer holding the CSV data loaded by csv_data_to_int().
data_rows	How many rows to free from the data.

2.1.1.11 csv_read()

```
void csv_read (
            const char * filename,
            char **** data,
            char delim,
            size_t(*) data_dims[2],
            bool has_headers )
```

Read a CSV file and store cells into a char∗∗∗ pointer.

Parameters

filename	Filename to read CSV file from.
data	A char**** pointer (char*** passed by address) that holds the CSV cells (need to pass by address, hence the parameter is char****). It's structured as data[x][y] where x represents the row, y represents the column and the contents is a string (char*).
data_dims	A size_t array with two indices: 0th index stores the row count and 1st index stores the column count.
delim	A single-character delimiter.
has_headers	A boolean indicating if the file has headers or not. If true, the first line will be skipped and not stored into data.

Example:

```
#include "csvparser.h"
int main() {
    char*** data = NULL;
    size_t data_dims[2];
    csv_read("../test.csv", &data, &data_dims, ',', true);
    printf("Rows: %zu\nColumns: %zu\n\n", data_dims[0], data_dims[1]);
    // print all cells of the parsed data
    for (size_t i = 0; i < data_dims[0]; ++i)
    {
        for (size_t j = 0; j < data_dims[1]; ++j)
        {
            printf("%s ", data[i][j]);
        }
        printf("\n");
    }
    // don't forget to free all the memory allocated to store the strings
    csv_free(&data, data_dims);
    return 0;
}
```

Output:

```
Rows: 3
Columns: 3
10 joe 10.33
20 jim 41.55
30 kyle -3.55
```

2.1.1.12 csv_read_column_by_index()

```
void csv_read_column_by_index (
            const char * filename,
            size_t column_index,
            char *** data,
            char delim,
            size_t * data_rows,
            bool has_headers )
```

Read a single column (by index) from CSV file and store cells into a char** pointer.

Parameters

filename	Filename to read CSV file from.
column_index	Index of the column to read.
data	A char*** pointer (char** passed by address) that holds the CSV cells from the specified column
delim	A single-character delimiter.
data_rows	A size_t* pointer (passed by address) to store the number of rows after parsing CSV file.
has_headers	A boolean indicating if the file has headers or not. If true, the first line will be skipped and not stored into data.

Example:
```
#include "csvparser.h"
int main() {
    char** data = NULL;
    size_t rows;
    csv_read_column_by_index("../test.csv", 1, &data, ',', &rows, true);
    printf("Rows: %zu\n", rows);
    // print all cells of the parsed data
    for (size_t i = 0; i < rows; ++i)
        printf("%s\n", data[i]);
    // don't forget to free all the memory allocated to store the strings
    csv_free_column(&data, rows);
    return 0;
}
```

Output: (this is the second column (1st index) of my sample CSV file)
```
Rows: 3
joe
jim
kyle
```

2.1.1.13 csv_read_column_by_name()

```
void csv_read_column_by_name (
            const char * filename,
            const char * column_name,
            char *** data,
            char delim,
            size_t * data_rows )
```

Read a single column (by name) from CSV file and store cells into a char** pointer.

Parameters

filename	Filename to read CSV file from.
column_name	Name of the column to read.
data	A char*** pointer (char** passed by address) that holds the CSV cells from the specified column.
delim	A single-character delimiter.
data_rows	A size_t* pointer (passed by address) to store the number of rows after parsing CSV file.

Example:
```
#include "csvparser.h"
int main() {
    char** data = NULL;
    size_t rows;
    csv_read_column_by_name("../test.csv", "col2", &data, ',', &rows);
    printf("Rows: %zu\n", rows);
    // print all cells of the parsed data
    for (size_t i = 0; i < rows; ++i)
        printf("%s\n", data[i]);
    // don't forget to free all the memory allocated to store the strings
    csv_free_column(&data, rows);
    return 0;
}
```

Output:
```
Rows: 3
joe
jim
kyle
```

2.2 csvparser.h

Go to the documentation of this file.

```
00001 #ifndef CSVPARSER_CSVPARSER_H
00002 #define CSVPARSER_CSVPARSER_H
00003
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #include <string.h>
00007 #include <stdbool.h>
00008
00057 void csv_read(const char* filename, char**** data, char delim, size_t (*data_dims)[2], bool
       has_headers);
00058
00101 void csv_read_column_by_index(const char* filename, size_t column_index, char*** data, char delim,
       size_t* data_rows, bool has_headers);
00102
00144 void csv_read_column_by_name(const char* filename, const char* column_name, char*** data, char delim,
       size_t* data_rows);
00145
00208 void csv_data_to_int(char*** data, size_t (*data_dims)[2], int*** int_data);
00209
00268 void csv_data_to_float(char*** data, size_t (*data_dims)[2], float*** float_data);
00269
00319 void csv_column_to_int(char** data, size_t data_rows, int** int_data);
00320
00370 void csv_column_to_float(char** data, size_t data_rows, float** float_data);
00371
00377 void csv_free(char**** data, size_t data_dims[2]);
00378
00385 void csv_free_int(int*** data, size_t data_rows);
00386
00393 void csv_free_float(float*** data, size_t data_rows);
00394
00400 void csv_free_column(char*** data, size_t data_rows);
00401
00407 void csv_free_column_int(int** data);
00408
00414 void csv_free_column_float(float** data);
00415
00416 #endif //CSVPARSER_CSVPARSER_H
```

Index