C CSV Parser Documentation

# Overview

The included files provide functions to parse CSV (or other character separated) files and/or strings into an accessible structure in the C programming language. The underlying parsed structures make use of doubly linked lists and are allocated on the heap of the running program.

The reason I wrote this CSV parser for C is its ease of access. While there are several other CSV parsers available, I could not find anyone that was as convenient as calling a function on a string or filename parameter and the parsed CSV structure being returned. Most of the parsers I found were more complicated than what I needed.

This documentation is intended to provide information on how to use the provided CSV parser.

# Basic Tutorial

Once the header file and the implementation file are included in your project, CSV files and strings can be parsed using the following functions:

|  |
| --- |
| struct csv\_table \* parse\_string\_to\_csv\_table(char \* string, char delim, char quot\_char, int strip\_spaces, int discard\_empty\_cells);  struct csv\_table \* open\_and\_parse\_file\_to\_csv\_table(char \* filename, char delim, char quot\_char, int strip\_spaces, int discard\_empty\_cells); |

The functions take the string to parse (string) or the address of the file to read and parse (filename). The other function parameters control the parser behaviour:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| delim | This is the character used to separate values. For CSVs it is , but it can be set to any symbol. |
| quot\_char | Character used for making quotes, e.g. " or ' |
| strip\_spaces | Boolean flag. If true (1), parsed CSV values will be stripped of leading and trailing spaces. |
| discard\_empty\_cells | Boolean flag. If true (1), any value that is an empty string ("") will not be added to the parsed structure. |

*Note: If the delimiter is a space character* (' ')*,* strip\_spaces *and* discard\_empty\_cells *will be overridden to* 0 *(false) and* 1 *(true) respectively.*

The return value is a pointer to the structure used to store the parsed CSV values, a struct csv\_table.

struct csv\_table is the highest level of the parsed CSV structure. It is made up of a doubly linked list of struct csv\_row, which are in turn made up of a doubly linked list of struct csv\_cell. The struct csv\_cell contains the appropriate parsed string value in its str field.

Diagram

Description automatically generated

Figure 1 - Parsed CSV Structure Layout

The structure is designed to mirror the table layout in the unparsed CSV, and provides functions to get a row/cell at a specific row and column index (see <some link here>).

The parsed CSV structure is allocated on the heap and therefore must be freed manually. The header files include free functions for each of the CSV structures to make sure all allocated memory is freed (see <some link here>).

# In-depth Tutorial

## Creating CSV Structures

CSV structures can be created in code using the *new\_csv* functions.

### Create CSV Cells

A new CSV cell can be created using the following:

|  |
| --- |
| // Create CSV cell with no string value  struct csv\_cell \*c1 = new\_csv\_cell();  // Create a cell with a string value  struct csv\_cell \*c2 = new\_csv\_cell\_from\_str("ACell2"); |

CSV cells can have their str field populated/overwritten with a string value using populate\_csv\_cell\_str. If the str field already has a value, the function will free the current value and write in the string passed in.

|  |
| --- |
| // Populate c1 with string  populate\_csv\_cell\_str(c1, "Cell1");  // Overwrite value in c2  populate\_csv\_cell\_str(c2, "Cell2"); |

### Create CSV Rows

New CSV rows can be created with the new\_csv\_row.

|  |
| --- |
| struct csv\_row \*r1 = new\_csv\_row(); |

### Create CSV Tables

New CSV tables can be created with new\_csv\_table.

|  |
| --- |
| struct csv\_table \*t1 = new\_csv\_table(); |

## Freeing CSV Structures

CSV Structures are allocated on the heap and therefore must be freed appropriately.

### Free CSV Cells

An allocated CSV cell can be de-allocated using free\_csv\_cell.

|  |
| --- |
| free\_csv\_cell(c1); |

### Free CSV Rows

An allocated CSV row can be de-allocated using free\_csv\_row.

|  |
| --- |
| free\_csv\_row(r1); |

When a CSV row is freed, all cells in its cell list are also freed (using free\_csv\_cell). **Therefore, if you are working with a cell mapped to a row, be sure to clone the cell contents before freeing the row.**

### Free CSV Tables

An allocated CSV table can be de-allocated using free\_csv\_table.

|  |
| --- |
| free\_csv\_table(t1); |

When a CSV table is freed, all rows in its row list are also freed (using free\_csv\_row).

## Printing CSV Cell Structures

## Getting Nested CSV Structures

## Combining CSV Structures

## Separating and Deleting CSV Structures

## Comparing CSV Structures

## Checking for Strings in CSV Structures

## Iterating Through Rows/Cells