npm Enterprise Features Pricing Docs Support

log in or sign up

Search packages

Share your code. npm Orgs help your team discover, share, and reuse code.
Create a free org »

csvtojson

2.0.8 • Public • Published 18 days ago

Readme

3 Dependencies

304 Dependents

81 Versions

install

```
> npm i csvtojson
```

weekly downloads

53,952

version	license
2.0.8	MIT

open issues	pull requests
7	1

homepage
github.com

repository
github

last publish
18 days ago

collaborators

Test With RunKit | Report A Vulnerability

CSVTOJSON

`csvtojson` module is a comprehensive nodejs csv parser to convert csv to json or column arrays. It can be used as node.js library / command line tool / or in browser. Below are some features:

- Strictly follow CSV definition RF4180
- Work with millions of lines of CSV data
- Provide comprehensive parsing parameters
- Provide out of box CSV parsing tool for Command Line
- Blazing fast -- Focus on performance
- Give flexibility to developer with 'pre-defined' helpers
- Allow async / streaming parsing
- Provide a csv parser for both Node.JS and browsers
- Easy to use API

csvtojson online

Here is a free online csv to json convert service utilizing latest `csvtojson` module.

Upgrade to V2

`csvtojson` has released version `2.0.0`.

- To upgrade to v2, please follow upgrading guide
- If you are looking for documentation for `v1`, open this page.

It is still able to use v1 with `csvtojson@2.0.0`

```
// v1
const csvtojsonV1=require("csvtojson/v1");
// v2
const csvtojsonV2=require("csvtojson");
const csvtojsonV2=require("csvtojson/v2");
```

Menu

Quick Start

Library

Installation

```
npm i --save csvtojson
```

From CSV File to JSON Array

```
/** csv file
a,b,c
1,2,3
4,5,6
*/
const csvFilePath='<path to csv file>'
const csv=require('csvtojson')
csv()
.fromFile(csvFilePath)
.then((jsonObj)=>{
    console.log(jsonObj);
    /**
     * [
     *  {a:"1", b:"2", c:"3"},
     *  {a:"4", b:"5". c:"6"}
     * ]
     */
})

// Async / await usage
const jsonArray=await csv().fromFile(csvFilePath);
```

From CSV String to CSV Row

```
/**
csvStr:
1,2,3
4,5,6
7,8,9
*/
const csv=require('csvtojson')
csv({
    noheader:true,
    output: "csv"
})
```

```
.fromString(csvStr)
.then((csvRow)=>{
    console.log(csvRow) // => [["1","2","3"], ["4","5","6"], ["7","
})
```

Asynchronously process each line from csv url

```
const request=require('request')
const csv=require('csvtojson')

csv()
.fromStream(request.get('http://mywebsite.com/mycsvfile.csv'))
.subscribe((json)=>{
    return new Promise((resolve,reject)=>{
        // long operation for each json e.g. transform / write into
    })
},onError,onComplete);
```

Convert to CSV lines

```
/**
csvStr:
a,b,c
1,2,3
4,5,6
*/

const csv=require('csvtojson')
csv({output:"line"})
.fromString(csvStr)
.subscribe((csvLine)=>{
    // csvLine =>  "1,2,3" and "4,5,6"
})
```

Use Stream

```
const csv=require('csvtojson');

const readStream=require('fs').createReadStream(csvFilePath);

const writeStream=request.put('http://mysite.com/obj.json');

readStream.pipe(csv()).pipe(writeStream);
```

To find more detailed usage, please see API section

Command Line Usage

Installation

```
$ npm i -g csvtojson
```

Usage

```
$ csvtojson [options] <csv file path>
```

Example

Convert csv file and save result to json file:

```
$ csvtojson source.csv > converted.json
```

Pipe in csv data:

```
$ cat ./source.csv | csvtojson > converted.json
```

Print Help:

```
$ csvtojson
```

API

Parameters

`require('csvtojson')` returns a constructor function which takes 2 arguments:

1. parser parameters
2. Stream options

```
const csv=require('csvtojson')
const converter=csv(parserParameters, streamOptions)
```

Both arguments are optional.

For `Stream Options` please read Stream Option from Node.JS

`parserParameters` is a JSON object like:

```
const converter=csv({
    noheader:true,
    trim:true,
})
```

Following parameters are supported:

- **output**: The format to be converted to. "json" (default) -- convert csv to json. "csv" -- convert csv to csv row array. "line" -- convert csv to csv line string
- **delimiter**: delimiter used for seperating columns. Use "auto" if delimiter is unknown in advance, in this case, delimiter will be auto-detected (by best attempt). Use an array to give a list of potential delimiters e.g. [",","|","$"]. default: ","
- **quote**: If a column contains delimiter, it is able to use quote character to surround the column content. e.g. "hello, world" wont be split into two columns while parsing. Set to "off" will ignore all quotes. default: " (double quote)
- **trim**: Indicate if parser trim off spaces surrounding column content. e.g. " content " will be trimmed to "content". Default: true
- **checkType**: This parameter turns on and off whether check field type. Default is false. (The default is `true` if version < 1.1.4)
- **ignoreEmpty**: Ignore the empty value in CSV columns. If a column value is not given, set this to true to skip them. Default: false.
- **fork (experimental)**: Fork another process to parse the CSV stream. It is effective if many concurrent parsing sessions for large csv files. Default: false
- **noheader**:Indicating csv data has no header row and first row is data row. Default is false. See header row
- **headers**: An array to specify the headers of CSV data. If --noheader is false, this value will override CSV header row. Default: null. Example: ["my field","name"]. See header row
- **flatKeys**: Don't interpret dots (.) and square brackets in header fields as nested object or array identifiers at all (treat them like regular characters for JSON field identifiers). Default: false.
- **maxRowLength**: the max character a csv row could have. 0 means infinite. If max number exceeded, parser will emit "error" of "row_exceed". if a possibly corrupted csv data provided, give it a number like 65535 so the parser wont consume memory. default: 0
- **checkColumn**: whether check column number of a row is the same as headers. If column number mismatched headers number, an error of "mismatched_column" will be emitted.. default: false
- **eol**: End of line character. If omitted, parser will attempt to retrieve it from the first chunks of CSV data.
- **escape**: escape character used in quoted column. Default is double quote (") according to RFC4108. Change to back slash (\) or other chars for your own case.
- **includeColumns**: This parameter instructs the parser to include only those columns as specified by the regular expression. Example: /(name|age)/ will parse and include columns whose header contains "name" or "age"
- **ignoreColumns**: This parameter instructs the parser to ignore columns as specified by the regular expression. Example: /(name|age)/ will ignore columns whose header contains "name" or "age"

- **colParser**: Allows override parsing logic for a specific column. It accepts a JSON object with fields like: `headName: <String | Function | ColParser>`. e.g. {field1:'number'} will use built-in number parser to convert value of the `field1` column to number. For more information See details below
- **alwaysSplitAtEOL**: Always interpret each line (as defined by `eol`) as a row. This will prevent `eol` characters from being used within a row (even inside a quoted field). This ensures that misplaced quotes only break on row, and not all ensuing rows.

All parameters can be used in Command Line tool.

Asynchronouse Result Process

Since `v2.0.0`, asynchronouse processing has been fully supported.

e.g. Process each JSON result asynchronousely.

```
csv().fromFile(csvFile)
.subscribe((json)=>{
    return new Promise((resolve,reject)=>{
        // Async operation on the json
        // dont forget to call resolve and reject
    })
})
```

For more details please read:

- Add Promise and Async / Await support
- Add asynchronous line by line processing support
- Async Hooks Support

Events

`Converter` class defined a series of events.

header

`header` event is emitted for each CSV file once. It passes an array object which contains the names of the header row.

```
const csv=require('csvtojson')
csv()
.on('header',(header)=>{
    //header=> [header1, header2, header3]
})
```

`header` is always an array of strings without types.

data

`data` event is emitted for each parsed CSV line. It passes buffer of strigified JSON in ndjson format unless `objectMode` is set true in stream option.

```
const csv=require('csvtojson')
csv()
.on('data',(data)=>{
    //data is a buffer object
    const jsonStr= data.toString('utf8')
})
```

error

`error` event is emitted if there is any errors happened during parsing.

```
const csv=require('csvtojson')
csv()
.on('error',(err)=>{
    console.log(err)
})
```

Note that if `error` being emitted, the process will stop as node.js will automatically `unpipe()` upper-stream and chained down-stream[1]. This will cause `end` event never being emitted because `end` event is only emitted when all data being consumed [2]. If need to know when parsing finished, use `done` event instead of `end`.

1. Node.JS Readable Stream
2. Writable end Event

done

`done` event is emitted either after parsing successfully finished or any error happens. This indicates the processor has stopped.

```
const csv=require('csvtojson')
csv()
.on('done',(error)=>{
    //do some stuff
})
```

if any error during parsing, it will be passed in callback.

Hook & Transform

Raw CSV Data Hook

the hook -- `preRawData` will be called with csv string passed to parser.

```
const csv=require('csvtojson')
// synchronouse
csv()
.preRawData((csvRawData)=>{
    var newData=csvRawData.replace('some value','another value');
    return newData;
})

// asynchronouse
csv()
.preRawData((csvRawData)=>{
    return new Promise((resolve,reject)=>{
        var newData=csvRawData.replace('some value','another value'
        resolve(newData);
    })

})
```

CSV File Line Hook

the function is called each time a file line has been parsed in csv stream. the `lineIdx` is the file line number in the file starting with 0.

```
const csv=require('csvtojson')
// synchronouse
csv()
.preFileLine((fileLineString, lineIdx)=>{
    if (lineIdx === 2){
        return fileLineString.replace('some value','another value')
    }
    return fileLineString
})

// asynchronouse
csv()
.preFileLine((fileLineString, lineIdx)=>{
    return new Promise((resolve,reject)=>{
            // async function processing the data.
    })


})
```

Result transform

To transform result that is sent to downstream, use `.subscribe` method for each json populated.

```
const csv=require('csvtojson')
csv()
.subscribe((jsonObj,index)=>{
    jsonObj.myNewKey='some value'
    // OR asynchronousely
    return new Promise((resolve,reject)=>{
        jsonObj.myNewKey='some value';
        resolve();
```

```
    })
})
.on('data',(jsonObj)=>{
    console.log(jsonObj.myNewKey) // some value
});
```

Nested JSON Structure

`csvtojson` is able to convert csv line to a nested JSON by correctly defining its csv header row. This is default out-of-box feature.

Here is an example. Original CSV:

```
fieldA.title, fieldA.children.0.name, fieldA.children.0.id,fieldA.cl
Food Factory, Oscar, 0023, Tikka, Tim, Joe, 3 Lame Road, Grantstown
Kindom Garden, Ceil, 54, Pillow, Amst, Tom, 24 Shaker Street, Hello
```

The data above contains nested JSON including nested array of JSON objects and plain texts.

Using csvtojson to convert, the result would be like:

```
[{
    "fieldA": {
        "title": "Food Factory",
        "children": [{
            "name": "Oscar",
            "id": "0023"
        }, {
            "name": "Tikka",
            "employee": [{
                "name": "Tim"
            }, {
                "name": "Joe"
            }]
```

```
        }],
        "address": ["3 Lame Road", "Grantstown"]
    },
    "description": "A fresh new food factory"
}, {
    "fieldA": {
        "title": "Kindom Garden",
        "children": [{
            "name": "Ceil",
            "id": "54"
        }, {
            "name": "Pillow",
            "employee": [{
                "name": "Amst"
            }, {
                "name": "Tom"
            }]
        }],
        "address": ["24 Shaker Street", "HelloTown"]
    },
    "description": "Awesome castle"
}]
```

Flat Keys

In order to not produce nested JSON, simply set `flatKeys:true` in parameters.

```
/**
csvStr:
a.b,a.c
1,2
*/
csv({flatKeys:true})
.fromString(csvStr)
.subscribe((jsonObj)=>{
    //{"a.b":1,"a.c":2}  rather than  {"a":{"b":1,"c":2}}
});
```

Header Row

`csvtojson` uses csv header row as generator of JSON keys. However, it does not require the csv source containing a header row. There are 4 ways to define header rows:

1. First row of csv source. Use first row of csv source as header row. This is default.
2. If first row of csv source is header row but it is incorrect and need to be replaced. Use `headers:[]` and `noheader:false` parameters.
3. If original csv source has no header row but the header definition can be defined. Use `headers:[]` and `noheader:true` parameters.
4. If original csv source has no header row and the header definition is unknow. Use `noheader:true`. This will automatically add `fieldN` header to csv cells

Example

```
// replace header row (first row) from original source with 'header1
csv({
    noheader: false,
    headers: ['header1','header2']
})

// original source has no header row. add 'field1' 'field2' ... 'fie
csv({
    noheader: true
})

// original source has no header row. use 'header1' 'header2' as its
csv({
    noheader: true,
    headers: ['header1','header2']
})
```

Column Parser

`Column Parser` allows writing a custom parser for a column in CSV data.

What is Column Parser

When `csvtojson` walks through csv data, it converts value in a cell to something else. For example, if `checkType` is `true`, `csvtojson` will attempt to find a proper type parser according to the cell value. That is, if cell value is "5", a `numberParser` will be used and all value under that column will use the `numberParser` to transform data.

Built-in parsers

There are currently following built-in parser:

- string: Convert value to string
- number: Convert value to number
- omit: omit the whole column

This will override types infered from `checkType:true` parameter. More built-in parsers will be added as requested in issues page.

Example:

```
/*csv string
column1,column2
hello,1234
*/
csv({
    colParser:{
        "column1":"omit",
        "column2":"string",
    },
    checkType:true
})
.fromString(csvString)
.subscribe((jsonObj)=>{
    //jsonObj: {column2:"1234"}
})
```

Custom parsers function

Sometimes, developers want to define custom parser. It is able to pass a function to specific column in `colParser`.

Example:

```
/*csv data
name, birthday
Joe, 1970-01-01
*/
csv({
    colParser:{
        "birthday":function(item, head, resultRow, row , colIdx){
            /*
                item - "1970-01-01"
                head - "birthday"
                resultRow - {name:"Joe"}
                row - ["Joe","1970-01-01"]
                colIdx - 1
            */
            return new Date(item);
        }
    }
})
```

Above example will convert `birthday` column into a js `Date` object.

the returned value will be used in result JSON object. returning `undefined` will not change result JSON object.

Flat key column

It is also able to mark a column as `flat`:

```
/*csv string
person.comment,person.number
```

```
hello,1234
*/
csv({
    colParser:{
        "person.number":{
            flat:true,
            cellParser: "number" // string or a function
        }
    }
})
.fromString(csvString)
.subscribe((jsonObj)=>{
    //jsonObj: {"person.number":1234,"person":{"comment":"hello"}}
})
```

Contribution

Very much appreciate any types of donation and support.

Code

`csvtojson` follows github convention for contributions. Here are some steps:

1. Fork the repo to your github account
2. Checkout code from your github repo to your local machine.
3. Make code changes and dont forget add related tests.
4. Run `npm test` locally before pushing code back.
5. Create a Pull Request on github.
6. Code review and merge
7. Changes will be published to NPM within next version.

Thanks all the contributors

Backers

Thank you to all our backers! [Become a backer]

Sponsors

Thank you to all our sponsors! (please ask your company to also support this open source project by becoming a sponsor)

Paypal

Keywords

csv csv parser parse csv csvtojson json csv to json csv convert tojson convert csv to json csv-json

You Need Help

Documentation
Support / Contact Us
Registry Status
Report Issues
npm Community Site
Security

About npm

About npm, Inc
Jobs
npm Weekly
Blog
Twitter
GitHub

Terms & Policies

Terms of Use
Code of Conduct
Package Name Disputes
Privacy Policy
Reporting Abuse
Other policies

npm loves you