

# csv Manual

Naoki Kaneko

---

---

## 目次

---

---

1. 概要 .....	1
2. CSV データのパーズ .....	2
2.1. 基本的な使い方など .....	2
2.2. パース出来るデータ形式について .....	2
3. CSV データのプリント .....	4
4. 必要なバージョンや依存など .....	5
5. バグ報告・修正や機能追加の提案 .....	6
6. ライセンスとコピーライト .....	6

---

---

## 1. 概要

---

---

このライブラリは、CSV 形式のデータのパーサー関数、及びプリンター関数を提供します。

提供するパッケージファイルは `csv.satyg` ファイルのみですので、読み込みは

```
@require: csv/csv
```

とするだけで完了します。`csv.satyg` ファイルが提供するモジュールの名前は `CSV` です。

## 2. CSV データのパーズ

### 2.1. 基本的な使い方など

CSV 形式のデータをパーズするための関数は `parser` です。型は

```
parser : string ?-> string ->
        ((string list) list) csv-parser-error result
```

です。

`'ok 'err result` 型は `satysfi-base` ライブラリが提供する型の一つです。成功を表す `Ok('ok)` というデータと、失敗を表す `Err('err)` というデータを表現する代数的データ型です。この型を操作するための関数は `satysfi-base` ライブラリ内の `result` パッケージで提供されています。

CSV データの表現は `(string list) list` という型によって行っています。

パーズに失敗したときのエラーを表すための型が `csv-parser-error` です。トップレベルで定義されています。内部の定義は

```
type csv-parser-error =
  | CSVParserErrorUnexpectedChar of int * string
  | CSVParserErrorEOI
```

です。予期しない文字が出現した場合は `CSVParserErrorUnexpectedChar` が返し、「その予期しない文字が出現する位置」と「予期しない文字そのもの」を取り出すことができます。予期しない文字の終了があった場合には `CSVParserErrorEOI` が返ります。

基本的に

```
CSV.parser `1,2,3`
```

のようにして使用します。

### 2.2. パース出来るデータ形式について

基本的に、改行文字（CRLF 若しくは LF）で行ごとに分かれ、カンマ文字でそれぞれの列に

分かれてリストになります。カンマの前後のスペース文字は無視されず、データに含まれます。また、カンマ直後に改行文字があった場合にも、空白文字の列が存在したと解釈します。

例：

```
1,2,3
foo,bar , baz
```

という CSV データをパースすると

```
Ok([[`1`; `2`; `3`]; [`foo`; `bar `; ` baz`]])
```

という (string list) list のデータ構造になります。

ダブルクォーテーションによる表現もパースすることができます。このとき、カンマとダブルクォーテーションの間に他の文字が入った場合はエラーになります。

例：

```
1,"foo",bar
```

という CSV データをパースすると

```
Ok([[`1`; `foo`; `bar`]])
```

というデータを取り出すことができます。しかし、

```
1,a"foo",bar
```

という CSV データをパースすると

```
Error: <pos: 3>, <char: `a`>
```

というエラーが返ります。

ダブルクォーテーションによる表現では、改行文字や区切り文字も含めることができます。

```
1,"foo
bar","baz1,baz2"
```

```
2, foo , 3
```

という CSV データをパースすると

```
Ok([[`1`; `foo  
bar`; `baz1,baz2`]; [`2`; ` foo `; ` 3`]])
```

というデータができ、改行文字と区切り文字がきちんと含まれていることがわかります。

ダブルクォーテーションを二つ重ねることで、CSV データにダブルクォーテーションを含めることができます。

```
1,""foo"bar"",baz
```

という CSV データをパースすると

```
Ok([[`1`; `foo"bar`; `baz`]])
```

となります。

区切り文字はカンマ以外にも変更することができ、例えばコロン区切りのデータを解析する場合は

```
1:2,320:s23  
2:423,232:sdf
```

というデータを、`CSV.parser ?:(`:`) data` のようにしてパースすると、

```
Ok([[`1`; `2,320`; `s23`]; [`2`; `423,232`; `sdf`]])
```

のようにきちんとデータが得られます。

---

---

### 3. CSV データのプリント

---

---

CSV データのプリントもできます。

`CSV.printer : string -> (string list) list -> string` という関数を使うことで、`(string list) list` というデータを CSV 形式の文字列に変換できます。

例えば

```
[[`1`; `foo"bar`; `baz`]; [`2`; `foo  
bar`]; [`3`; `fo"o  
bar`; `baz1,baz2`]]
```

という (string list) list のデータを CSV.printer data とすることで

```
1, "\"foo\"bar\"", baz  
2, "foo  
bar"  
3, "\"fo\"o  
bar\"", "baz1, baz2"
```

という CSV データに変換することができます。

パーサーと同様に区切り文字を変更することができ、先ほどのデータを CSV.printer ?:(`:`) data とすることで

```
1: "\"foo\"bar\"": baz  
2: "foo  
bar"  
3: "\"fo\"o  
bar\"": baz1, baz2
```

のように、コロン区切りの文字列に変換することができます。

---

## 4. 必要なバージョンや依存など

---

必要な SAT<sub>Y</sub>SF<sub>I</sub> のバージョンは 0.0.5 以上、0.0.7 未満です（文字列操作のプリミティブに変更がない場合は 0.0.7 以降のバージョンも使うことができます）。

また、このライブラリは SAT<sub>Y</sub>SF<sub>I</sub> の標準ライブラリと、satysfi-base という外部ライブラリに依存しています。ドキュメントの作成には debug-show-value という外部ライブラリにも依存しています。それぞれのインストールは satyrophos を使用することを想定しています。

---

## 5. バグ報告・修正や機能追加の提案

---

このパッケージはバグが存在するかもしれません。バグを発見した場合は以下の URL に報告してください。

<https://github.com/puripuri2100/satysfi-csv/issues>

このパッケージに対してコードの修正や機能追加の提案をしたい場合は、GitHub の機能を用いて以下の URL にプルリクエストを送ってください。

<https://github.com/puripuri2100/satysfi-csv/pulls>

バグ報告・修正提案・機能追加提案をお待ちしております。

---

## 6. ライセンスとコピーライト

---

このパッケージとドキュメントは MIT ライセンスのもとで配布されます。

Copyright (c) 2021 Naoki Kaneko (a.k.a. "puripuri2100")