This simple sample document shows possibilities of the handlecsv.lua module.
Imagine, that CSV file `myfirstcsvexamplefile.csv` that I want to use for my print report has the so-called header (ie. the first line contains the names of each columns of the table) and the separator of columns is a semicolon.
When I need process this CSV file, then at first I have to load required module by command:

```
\usemodule[handlecsv] % this command load module into my source code
```

The module now needs to know, whether CSV file contain a header and to know which delimiter used to divide to separate columns. The default setting of the module library setting can be changed to suit my needs.

```
\setheader % this set a header flag i.e. CSV file has header in first line
\setsep{,} % change settings of default separator (this is ; semicolon)
```

Now I can open my CSV file for processing by the command:

```
\opencsvfile{myfirstcsvexamplefile.csv}
```

After opening the file I can find out a lot of useful information. eg typing of line:

```
The file \csvfilename\ has \numcols columns\  and \numrows\ rows (lines).
```

Lists message:
The file myfirstcsvexamplefile.csv has 12columns and 33 rows (lines).

macro.introductionsmacros

```
\resethooks
\setheader
\unsetheader (or syn. \resetheader)
\setsep{;}
\unsetsep (or syn. \resetsep)
\setfiletoscan{filename.csv}
\opencsvfile
\opencsvfile{filename.csv}
```

macro.infomacros

```
\numrows
\numemptyrows
\numnotemptyrows
\numcols
\csvfilename
\csvreport
\csvreport{filename.csv}
\printline
\printall
```

Defining `\newifs` conditionals for processing testing

For ConTeXt testing of header settings are defined two newifs:

```
\ifissetheader
\ifnotsetheader
\ifEOF
\ifnotEOF
\ifemptyline
\ifnotemptyline
\ifemptylinesmarking
\ifemptylinesnotmarking
```

Examples of using predefined conditionals

Here is actual settings of these conditionals:
The following source code:

```
1. \type{issetheader} is \ifissetheader TRUE \else FALSE\fi
2. \type{notsetheader} is \ifnotsetheader TRUE \else FALSE\fi
3. \type{EOF} is \ifEOF TRUE \else FALSE\fi
4. \type{notEOF}  is \ifnotEOF TRUE \else FALSE\fi
5. \type{emptyline} is \ifemptyline TRUE \else FALSE\fi
6. \type{notemptyline} is \ifnotemptyline TRUE \else FALSE\fi
7. \type{emptylinesmarking}  is \ifemptylinesmarking TRUE \else FALSE\fi
8. \type{emptylinesnotmarking}  is \ifemptylinesnotmarking TRUE \else FALSE\fi

%  We can use macros \markemptylines and \notmarkemptylines to set values:
\markemptylines
9. After \type{\markemptylines} settings is \type{emptylinesmarking}  set
to \ifemptylinesmarking TRUE \else FALSE\fi
10. and \type{emptylinesnotmarking}  is \ifemptylinesnotmarking TRUE \else
FALSE\fi
\notmarkemptylines
11. After \type{\notmarkemptylines} settings is \type{emptylinesmarking}
set to \ifemptylinesmarking TRUE \else FALSE\fi
12. and \type{emptylinesnotmarking} is \ifemptylinesnotmarking TRUE \else
FALSE\fi
```

```
\resetmarkemptylines
13. After \type{\resetmarkemptylines} settings is \type{emptylinesmarking}
set to \ifemptylinesmarking TRUE \else FALSE\fi
14. and \type{emptylinesnotmarking} is \ifemptylinesnotmarking TRUE \else
FALSE\fi
```

produces the following output result:

1. `issetheader` is TRUE
2. `notsetheader` is FALSE
3. `EOF` is FALSE
4. `notEOF` is TRUE
5. `emptyline` is FALSE
6. `notemptyline` is TRUE
7. `emptylinesmarking` is FALSE
8. `emptylinesnotmarking` is TRUE

9. After `\markemptylines` settings is `emptylinesmarking` set to TRUE
10. and `emptylinesnotmarking` is FALSE
11. After `\notmarkemptylines` settings is `emptylinesmarking` set to FALSE
12. and `emptylinesnotmarking` is TRUE
13. After `\resetmarkemptylines` settings is `emptylinesmarking` set to FALSE
14. and `emptylinesnotmarking` is TRUE

Predefined macros for column work processing

There are predefined these macros for column information processing:

```
\colname[number]
\xlscolname[number]
\numberxlscolname['XLSColName']
\indexcolname['ColName']
\indexcolname['XLSColName']
\indexcolname['cXLSColName']
\columncontent[number]
\columncontent['ColName']
\columncontent['XLSColName']
\columncontent['cXLSColName']
```

Examples of using predefined columns macros

The following source code:

```
1. \type{\colname[2]}: \colname[2]
2. \type{\xlscolname[5]}: \xlscolname[5]
3. \type{\numberxlscolname['F']}: \numberxlscolname['F']
4. \type{\indexcolname['Firstname']}: \indexcolname['Firstname']
5. \type{\indexcolname['B']}: \indexcolname['B']
6. \type{\indexcolname['cC']}: \indexcolname['cC']
7. \type{\columncontent[2]}: \columncontent[2]
8. \type{\columncontent['A']}: \columncontent['A']
9. \type{\columncontent['cA']}: \columncontent['cA']
10. \type{\columncontent['Firstname']}: \columncontent['Firstname']
11. \type{\columncontent['firstname']}: \columncontent['firstname']
12. \type{\columncontent['123']}: \columncontent['123']
13. \type{\columncontent['!?*']}: \columncontent['!?*']
```

produces the following output result:

1. `\colname[2]`: last_name
2. `\xlscolname[5]`: E
3. `\numberxlscolname['F']`: 6
4. `\indexcolname['Firstname']`:
5. `\indexcolname['B']`: 2

6. `\indexcolname['cC']`: 3
7. `\columncontent[2]`: Butt
8. `\columncontent['A']`: James
9. `\columncontent['cA']`: James
10. `\columncontent['Firstname']`: nil
11. `\columncontent['firstname']`: nil
12. `\columncontent['123']`: nil
13. `\columncontent['!?*']`: nil

---

example.makronames

1: A – James
1: B – Butt
1: C – Benton, John B Jr
1: D – 6649 N Blue Gum St
1: E – New Orleans
1: F – Orleans
1: G – LA
1: H – 70116
1: I – 504-621-8927
1: J – 504-845-1427
1: K – jbutt@gmail.com
1: L – http://www.bentonjohnbjr.com

5: A – Donette
5: B – Foller
5: C – Printing Dimensions
5: D – 34 Center St
5: E – Hamilton
5: F – Butler
5: G – OH
5: H – 45011
5: I – 513-570-1893
5: J – 513-549-4561
5: K – donette.foller@cox.net
5: L – http://www.printingdimensions.com

macro.numrows

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.