# spreadtab

v0.61

## User's manual

Christian Tellechea
unbonpetit@netc.fr

14 March 2025

---

***Résumé***

This package allows to use spreadsheet features in any ”table” environment with LaTeX.

The main functionality is being able to write formulas in the cells of a table that refer to other cells, calculate the formulas contained in the cells and display the numerical results of these formulas in the table.

---

# Contents

# 1 Presentation

This manual[1] describes the spreadtab package. It lets you build spreadsheet-like tables. Table cells have coordinates (column and row) which can be used in formulas to calculate values in other cells. To load the package, write in the preamble

```
\usepackage{spreadtab}
```

This package requires the LaTeX $2_\varepsilon$ format after 2022/06/01 as well as the `xstring` and `simplekv` packages.

This package is intended to be compatible with all table environments, provided that column separators are "&" and line breaks are "\\". spreadtab acts completely independently of the table environment. This means that reading the table, processing and calculating formulas takes place before the table environment takes over and "sees" the body of the table.

Therefore, spreadtab proceeds in 3 steps:

- First, spreadtab reads the body of the array. If the macro `\STxp` is present one or more times, its argument is expanded to the maximum. The body of the table is then divided into rows and cells;
- the formulas contained in the cells are then calculated, having first calculated the dependent cells;
- the body of the table is finally reconstructed, having replaced each formula with the numerical value previously calculated, and the whole is given to the table environment specified by the user.

The syntax is:

```
\begin{spreadtab}[⟨options⟩]{{⟨tabname⟩}{⟨colspec⟩}}
    table with formulas and numbers
\end{spreadtab}
```

where ⟨*tabname*⟩ represents the name of any array environment available with LaTeX or with a package. You can replace `\begin{spreadtab}...\end{spreadtab}` with `\spreadtab...\endspreadtab`.

The ⟨*options*⟩ must be in the form `key = ⟨value⟩` to apply only to the current array. To specify ⟨*options*⟩ valid for all future arrays, use

```
\STset{⟨key⟩=⟨value⟩}
```

The list of `keys` and associated ⟨*values*⟩ is available on page 18.

Although the spreadsheet-like functionality of LaTeX is appreciable, you should not lose sight of the fact that the 3 steps described above are time-consuming. As a result, compilation times are much longer than with a conventional table.

# 2 New in v0.61

## 2.1 Deletion

The macro functions `ifeq`, `iflt` and `ilgt` have been removed, since `l3fp` provides the ternary operator

```
⟨test⟩?⟨true⟩:⟨false⟩
```

The following macros, available in previous versions, have been removed: `\STautoround`, `\STsetdecimalsep`, `\STeol`, `\STsetdisplaymarks`, `\STmessage`, `\STnumericfieldmarker`, `\STtextcell` and `\STtranposechar`. All settings are now made via `\STset{⟨keys⟩=⟨values⟩}`.

## 2.2 Macro `\STrep`

See page 6.

---

[1] What you are reading is only a translation from French done by AIs (DeepL, ChatGPT, and Mistral Chat). I have barely reviewed it, so there may be errors or inconsistencies. For *reliable* information, please refer to the French manual.

### 2.3 Macro-function `test`

See page 12.

## 3 Common Features

### 3.1 Calculation

A cell can now contain any calculation understood by `l3fp`: basic operations, trigonometry and inverses, exponentials and logarithms, comparisons, logical connectors[2], factorials, random numbers, etc. See the documentation for `interface3`.

As a result, the macro-functions `rand`, `rnd`, and `fact` are still usable but are *no longer* part of `spreadtab`: they are now directly understood by `l3fp`.

### 3.2 Line Breaks

By default, `spreadtab` considers a line to end with `\\`, which is standard in tables. The optional argument `[⟨dimension⟩]` that follows is taken into account. This line-break marker can be modified using the key `tabline sep`. For example, you can write:

```
tabline sep = \tabularnewline
```

The line breaks that will be inserted into the final table will always be ”`\\`”, even if the line break marker was modified when `spreadtab` *reads* the table.

Just after the line break, `spreadtab` recognizes most horizontal rules as well as their optional arguments: `\hline`, `\cline`, `\hhline`, `\noalign`, `\toprule`, `\midrule`, `\bottomrule`, `\cmidrule`, `\addlinespace`, `\morecmidrules`, and `\specialrule`.

For other cases, see page 15 for further instructions.

### 3.3 Absolute References

In the table, cells are identified by their absolute references as follows:

- the column is represented by a letter, either uppercase or lowercase, from `a` to `z`, where `a` represents the first column on the left: we are thus initially limited to 26 columns, which should be sufficient for the vast majority of cases;
- immediately following the letter, a strictly positive integer represents the row number, with row number 1 being the topmost row.

An absolute reference is thus written, for example: ”`b4`”, ”`C1`”, or ”`d13`”. Visually, this can be illustrated by a table similar to a spreadsheet, here intentionally limited to 5 rows and 5 columns:

|  | **A** | **B** | **C** | **D** | **E** |
|---|---|---|---|---|---|
| **1** |  |  |  |  |  |
| **2** |  |  |  |  |  |
| **3** |  |  |  |  |  |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |

In this example, we calculate the sum of each row and column, and then the total sum:

---

[2] If you wish to use the logical connector ”and” in `l3fp`, denoted by ”&&”, it is *mandatory* to enclose this operator in braces within a table to prevent the & tokens from being interpreted as column separators. In a table cell, for example, to check if the content of the cell `a1` belongs to an interval, you would write: ”`a1>1 {&&} a1<10`”.

```
\begin{spreadtab}{{tabular}{rr|r}}
    22       & 54         & a1+b1 \\
    43       & 65         & a2+b2 \\
    49       & 37         & a3+b3 \\
    \hline
    a1+a2+a3 & b1+b2+b3   & a4+b4
\end{spreadtab}
```

| | | |
|---|---|---|
| 22 | 54 | 76 |
| 43 | 65 | 108 |
| 49 | 37 | 86 |
| 114 | 156 | 270 |

In this other example, we calculate a few lines of Pascal's triangle:

```
\begin{spreadtab}{{tabular}{ccccc}}
    1  &       &       &       &    \\
    a1 & a1    &       &       &    \\
    a2 & a2+b2 & b2    &       &    \\
    a3 & a3+b3 & b3+c3 & c3    &    \\
    a2 & a4+b4 & b4+c4 & c4+d4 & d4
\end{spreadtab}
```

| | | | | |
|---|---|---|---|---|
| 1 | | | | |
| 1 | 1 | | | |
| 1 | 2 | 1 | | |
| 1 | 3 | 3 | 1 | |
| 1 | 4 | 6 | 4 | 1 |

## 3.4 Relative references

To reference a cell, you can specify its position relative to the cell containing the formula. Thus, the "relative" coordinates of a cell are 2 relative numbers written according to this syntax: `[`$\langle x\rangle$`,`$\langle y\rangle$`]`, where $\langle x\rangle$ is the horizontal offset from the cell containing the formula and $\langle y\rangle$ is the vertical offset. Thus, `[-2,3]` refers to the cell 2 columns before (left) and 3 rows after (lower) the cell containing the formula.

Here again is the Pascal triangle seen above, but the references are relative and the "`matrix`" environment of the `amsmath` package is used:

```
$
\begin{spreadtab}{{matrix}{}}
    1\\
    [0,-1] & [-1,-1]\\
    [0,-1] & [-1,-1]+[0,-1] & [-1,-1]\\
    [0,-1] & [-1,-1]+[0,-1] & [-1,-1]+[0,-1] & [-1,-1]\\
    [0,-1] & [-1,-1]+[0,-1] & [-1,-1]+[0,-1] & [-1,-1]+[0,-1] & [-1,-1]
\end{spreadtab}
$
```

| | | | | |
|---|---|---|---|---|
| 1 | | | | |
| 1 | 1 | | | |
| 1 | 2 | 1 | | |
| 1 | 3 | 3 | 1 | |
| 1 | 4 | 6 | 4 | 1 |

A mixture of absolute and relative references can be used in the same formula.

## 3.5 Text cells

If you want to put text in a cell, simply place the "`@`" character somewhere in the cell. In doing so, the cell is ignored by spreadtab and becomes an inert cell that cannot be referred to anywhere else in the table.

The key `text mark` is used to modify the text marker. For example

```
\STset{text mark=`}
```

will cause a cell containing the character "`" to be understood as a text cell. It's important to note, however, that the `text mark` key detokenizes its value, making the text marker made up of token(s) of catcode 12: to be recognized by spreadtab, these tokens must have this catcode 12 at the time the array is read.

```
\begin{spreadtab}{{tabular}{|r|ccc|}}\hline
    @ values of $x$ & -5                   & -1 & 4 \\
    @ $f(x)=2x$     & \STcopy>{2*[0,-1]}   &    &   \\\hline
\end{spreadtab}\medbreak
\STset{text mark=`}
\begin{spreadtab}{{tabular}{|r|ccc|}}\hline
    `values of $x$ & -5                   & -1 & 4 \\
    `$f(x)=2x$     & \STcopy>{2*[0,-1]}   &    &   \\\hline
\end{spreadtab}
```

| | | | |
|---|---|---|---|
| values of $x$ | -5 | -1 | 4 |
| $f(x) = 2x$ | -10 | -2 | 8 |

| | | | |
|---|---|---|---|
| values of $x$ | -5 | -1 | 4 |
| $f(x) = 2x$ | -10 | -2 | 8 |

If a cell is empty or made up entirely of spaces, then spreadtab will treat it as a text cell.

## 3.6 Mixed Cells

In reality, each cell consists of *two* fields. On one side, the ⟨*numeric field*⟩ contains the formula, and on the other side, the ⟨*text field*⟩ will be ignored by the calculation engine and does not contribute to the calculations:

- In a cell, if nothing is specified, the entire cell is considered the ⟨*numeric field*⟩, and the ⟨*text field*⟩ is empty;
- If the cell contains ”`@`”, then the entire cell is considered the ⟨*text field*⟩. The ⟨*numeric field*⟩ is empty and inaccessible.
- If the cell contains ”`:=`”, then the argument in curly braces that follows is the ⟨*numeric field*⟩, and everything else is the ⟨*text field*⟩. The cell has this structure:

  `⟨text field⟩:={⟨numeric field⟩}⟨text field⟩`

  With the key `numeric mark`, you can change this marker to ”`\=`” with:

  `numeric mark={\=}`

  In this case, the definition of `\=` would have no importance and would not intervene in the process: for `spreadtab`, it is simply a marker for the start of the formula that is searched for and recognized without being expanded.

Once the ⟨*numeric field*⟩ is calculated, only it and the marker ”`:=`” will be replaced by the calculated numeric value.

It is worth noting that the ⟨*numeric field*⟩ can be inside curly braces.

To illustrate the idea, here is a very simple example:

```
\begin{spreadtab}{{tabular}{|c|c||c|}}\hline
    val 1: :={50} & val 2: :={29} & mean: \textbf{:={(a1+b1)/2}}\\\hline
\end{spreadtab}
```

| | | |
|---|---|---|
| val 1: 50 | val 2: 29 | mean: **39.5** |

Note also that ”`:={}`”, which defines an empty formula, has the same effect as ”`@`” in a cell: the cell is understood as a text cell. However, ”`@`” and ”`:={}`” *are not equivalent* because a text cell containing the latter can receive a formula by copying (see the next section), which is impossible with ”`@`”.

## 3.7 Copying formulas

To avoid having to re-enter identical formulas in adjacent cells, `spreadtab` provides the `\STcopy` instruction.

This command is placed in a cell using the syntax `\STcopy{>⟨x⟩,v⟨y⟩}{⟨formula⟩}`, where ⟨*x*⟩ and ⟨*y*⟩ are positive numbers representing horizontal and vertical offsets from the cell containing the instruction. The cell containing the command and the cell obtained by these offsets define a rectangular range of cells that will receive the `⟨formula⟩`[3]. The `\STcopy` *must not* be in a cell where a numeric field marker ”`:=`” is present.

For each target cell, all references in the formula are incremented to account for the offset between the target cell and the source cell. The formula may also contain relative references, which, being relative, will not be modified.

By preceding a coordinate reference with a ”`!`” character, this coordinate remains unchanged during the copy. The character ”`!`” can be changed with `freeze char` = `⟨character⟩`. It is important to note that the `⟨character⟩` is detokenized.

In the syntax `\STcopy{>⟨x⟩,v⟨y⟩}{⟨formula⟩}`, if the number ⟨*x*⟩ is omitted, the copy in the horizontal direction is made to the right up to the right edge of the table. If ⟨*y*⟩ is omitted, the copy in the vertical direction is made down to the bottom of the table.

This allows generating the multiplication table from 1 to 10:

---

[3] The copy can therefore only be made to cells that are to the right and below the cell where the command is located.

```
\begin{spreadtab}{{tabular}{|c|*{10}{c}|}}
    \hline
    @$\times$        & 1                     & \STcopy{>}{b1+1}  & & & & & & & & \\\hline
    1                & \STcopy{>,v}{!a2*b!1} &                   & & & & & & & & \\
    \STcopy{v}{a2+1} &                       &                   & & & & & & & & \\
                     &                       &                   & & & & & & & & \\
                     &                       &                   & & & & & & & & \\
                     &                       &                   & & & & & & & & \\
                     &                       &                   & & & & & & & & \\
                     &                       &                   & & & & & & & & \\
                     &                       &                   & & & & & & & & \\
                     &                       &                   & & & & & & & & \\
                     &                       &                   & & & & & & & & \\\hline
\end{spreadtab}
```

| × | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

When copying a formula, if the target cell already contains a non-empty numeric field, this is not overwritten and the copy is not made.

## 3.8 Macros provided by spreadtab

### 3.8.1 Macro `\STset`

The `\STset` macro includes in its mandatory argument a csv of `keys` = ⟨*values*⟩ to specify settings that operate for the rest of the document.

The list of `keys` and associated ⟨*values*⟩ is available on the 18 page.

### 3.8.2 The macros `\STxp` and `\STrep`

The presence[4] of `\STxp{<argument>}` in the body of the table has the effect of fulle xpansion of the ⟨*argument*⟩. If one or more parts of the `<argument>` are not expandable (e.g. non-robust macros), it's up to the user to block full expansion using, for example, the `\noexpand` or `\unexpanded` primitives. The `\STxp` macro must not be enclosed in braces: it will be ignored and executed much later, causing a compilation error.

The expandable macro `\STrep` replicates times an arbitrary ⟨*code*⟩:

`\STrep{⟨n⟩}{⟨code⟩}`

Used in the argument of `\STxp`, it allows you to generate empty cells or identical lines in a concise way.

---

[4] `spreadtab` tests its *presence* and then acts on its argument: `\STxp` takes the place of a marker and is not executed when the array is read.

```
\begin{spreadtab}{{tabular}{|c|*{10}{c}|}}
    \hline
    @$\times$        & 1                     & \STcopy{>}{b1+1}  \STxp{\STrep8&}\\\hline
    1                & \STcopy{>,v}{!a2*b!1} \STxp{\STrep9&}\\
    \STcopy{v}{a2+1} \STxp{\STrep9{\STrep{10}&\\}}
    \hline
\end{spreadtab}
```

| × | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

If one or more parts of the `<code>` cannot be expanded (non-robust macros in particular), it is up to the user to stop the full expansion initiated by `\STxp` with, for example, the `\noexpand` or `\unexpanded` primitives.

# 4 Table formatting

## 4.1 Codes executed before or after the table

The `pretab code` = ⟨*code*⟩ and `posttab code` = ⟨*code*⟩ keys define arbitrary codes executed just before and just after table display.

```
A simple
\begin{spreadtab}[pretab code=\begingroup\color{red},
                  posttab code=\endgroup]{{tabular}[t]{r}}
    149\\
    287\\\hline
    a1+a2
\end{spreadtab} addition.
```

A simple 149 addition.
287
436

## 4.2 Decimal Separator

Since `l3fp` gives decimal results with the dot as the decimal separator, it is possible to change this decimal separator so that all results are returned as if they accounted for it.

The key `dec sep` = ⟨*char*⟩ allows changing the decimal separator to a comma, but this choice is not neutral in math mode. In fact, it is considered punctuation, which explains why it is followed by a space.

To avoid this typographic error, you can:

- define the decimal separator in curly braces with `dec sep={{,}}`
- execute the code `\mathcode`,="013B\relax` before displaying the table

The most elegant solution is to use a configurable package specialized in number formatting, such as `siunitx`.

```
\begin{spreadtab}[dec sep={,}]{{tabular}{|*3{>{$}r<{$}}|}}\hline
    @x   & @y  & @\text{Mean}\\\hline
    5    & -4  & (a2+b2)/2\\
    -6.1 & -8  & (a3+b3)/2\\
    9.85 & 3.7 & (a4+b4)/2\\\hline
\end{spreadtab}\par\smallskip
\begin{spreadtab}[dec sep={{,}}]{{tabular}{|*3{>{$}r<{$}}|}}\hline
    @x   & @y  & @\text{Mean}\\\hline
    5    & -4  & (a2+b2)/2\\
    -6.1 & -8  & (a3+b3)/2\\
    9.85 & 3.7 & (a4+b4)/2\\\hline
\end{spreadtab}\par\smallskip
\begin{spreadtab}[dec sep     ={,},
                  pretab code ={\begingroup\mathcode`\,="013B\relax},
                  posttab code={\endgroup}]{{tabular}{|*3{>{$}r<{$}}|}}\hline
    @x   & @y  & @\text{Mean}\\\hline
    5    & -4  & (a2+b2)/2\\
    -6.1 & -8  & (a3+b3)/2\\
    9.85 & 3.7 & (a4+b4)/2\\\hline
\end{spreadtab}
```

| $x$ | $y$ | Mean |
|---:|---:|---:|
| 5 | −4 | 0, 5 |
| −6, 1 | −8 | −7, 05 |
| 9, 85 | 3, 7 | 6, 775 |

| $x$ | $y$ | Mean |
|---:|---:|---:|
| 5 | −4 | 0,5 |
| −6,1 | −8 | −7,05 |
| 9,85 | 3,7 | 6,775 |

| $x$ | $y$ | Mean |
|---:|---:|---:|
| 5 | −4 | 0,5 |
| −6,1 | −8 | −7,05 |
| 9,85 | 3,7 | 6,775 |

## 4.3 Automatic rounding

The `l3fp` engine gives up to 16 significant decimals. If you don't want such precision, you can automatically round off the result of each calculation with the `autoround` key. Acting on this key implies a *deliberate limitation* of the precision of internal calculations.

```
\begin{spreadtab}{{tabular}{*4c}}       \hline
    @$x$           & 3               & 17 & 751 \\\hline
    @$1/x$         & \STcopy>{1/b1}  &    &     \\\hline
    @$x\cdot(1/x)$ & \STcopy>{b2*b1} &    &     \\\hline
\end{spreadtab}
\bigbreak
\begin{spreadtab}[autoround=6]{{tabular}{*4c}}    \hline
    @$x$           & 3               & 17 & 751 \\\hline
    @$1/x$         & \STcopy>{1/b1}  &    &     \\\hline
    @$x\cdot(1/x)$ & \STcopy>{b2*b1} &    &     \\\hline
\end{spreadtab}
```

| $x$ | 3 | 17 | 751 |
|:---:|:---:|:---:|:---:|
| $1/x$ | 0.3333333333333333 | 0.05882352941176471 | 0.00133155792276964 |
| $x \cdot (1/x)$ | 0.9999999999999999 | 1 | 0.9999999999999996 |

| $x$ | 3 | 17 | 751 |
|:---:|:---:|:---:|:---:|
| $1/x$ | 0.333333 | 0.058824 | 0.001332 |
| $x \cdot (1/x)$ | 0.999999 | 1.000008 | 1.000332 |

To return to the default behavior and do no rounding, set an empty value:

```
autoround={}
```

## 4.4 Macro `\STprintnum`

Each number processed by spreadtab is given to the `\STprintnum` macro in the final array. By default, this macro does not modify any of its arguments, and its definition is as follows:

```
\def\STprintnum#1{#1}
```

If you wish, you can modify using the `siunitx` package and its `\num` macro:

```
\def\STprintnum#1{\num[round-mode = places,
                       round-precision = 6,
                       drop-zero-decimal=true,
                       output-decimal-marker = {,}]{#1}%
}
\begin{spreadtab}{{tabular}{*4c}}                 \hline
    @$x$           & 3               & 17 & 751 \\\hline
    @$1/x$         & \STcopy>{1/b1}  &    &     \\\hline
    @$x\cdot(1/x)$ & \STcopy>{b1*b2} &    &     \\\hline
\end{spreadtab}
```

| $x$ | 3 | 17 | 751 |
|---|---|---|---|
| $1/x$ | 0,333 333 | 0,058 824 | 0,001 332 |
| $x \cdot (1/x)$ | 1 | 1 | 1 |

In this case, no change is made to the accuracy of internal calculations.

## 4.5 Hide row or column

Sometimes, an entire row or column is used for intermediate calculations that don't need to be displayed in the final table. For this purpose, `\SThiderow` and `\SThidecol` control sequences are available which, when placed in a cell, hide the row or column in which the cell is located.

```
\begin{spreadtab}{{tabular}{|r|ccc|}}                                   \hline
    @ values of $x$        & -1         & 0\SThidecol & 2          & 3          \\\hline
    @$f(x)=2x-1$           & 2*[0,-1]-1 & 2*[0,-1]-1  & 2*[0,-1]-1 & 2*[0,-1]-1 \\
    @$g(x)=x-10$\SThiderow & [0,-2]-10  & [0,-2]-10   & [0,-2]-10  & [0,-2]-10  \\
    @$h(x)=1-x$            & 1-[0,-3]   & 1-[0,-3]    & 1-[0,-3]   & 1-[0,-3]   \\\hline
\end{spreadtab}
```

| values of $x$ | -1 | 2 | 3 |
|---|---|---|---|
| $f(x) = 2x - 1$ | -3 | 3 | 5 |
| $h(x) = 1 - x$ | 2 | -1 | -2 |

You can see how the row containing $g(x)$ and the column corresponding to the value 0 are hidden.

Remember that masked rows and columns are *invisible* for the user's chosen array environment, which explains why in the array preamble only 4 columns (`|r|ccc|`) have been defined and not 5 as shown.

## 4.6 Export cell value

You may need to save the numerical value of a cell in a macro (the assignment is global) to display it in a text field[5] or even outside the table. In this case, use the key `save list`:

`save list={⟨macroA⟩ = ⟨refA⟩, ⟨macroB⟩ = ⟨refB⟩,...}`

where each reference must be absolute.

```
\STset{ save list = { \resultC = c1 , \resultB = b1 }}
\begin{spreadtab}{{tabular}{|c|c|c|c|c|}}\hline
    10 & a1+10 & b1+10 & a1+b1+c1 & @cell c1: \resultC\\\hline
\end{spreadtab}
\par\medskip
Here is cell c1: \resultC\ and b1: \resultB
```

| | | | | |
|---|---|---|---|---|
| 10 | 20 | 30 | 60 | cell c1: 30 |

Here is cell c1: 30 and b1: 20

When the key `aux save list` is used with the same syntax, everything happens as before, but the assignment is also written to the auxiliary file. In this way, the value of a cell can be transmitted between compilations, so that it can be referred to before the array is encountered.

```
Valeur of d1 is \cellD.\par
\STset{ aux save list = { \cellD = d1 }}
\begin{spreadtab}{{tabular}{|c|c|c|c|}}\hline
    1 & a1+10 & b1+10 & a1+b1+c1\\\hline
\end{spreadtab}
```

Valeur of d1 is 33.

| | | | |
|---|---|---|---|
| 1 | 11 | 21 | 33 |

In both cases, the value of the `save cell` and `aux save cell` keys is reset to empty after the table following `\STset` is displayed.

It is also possible, using the macro function `tag`, to export a value to the auxiliary file, see page 14.

---

[5] It's easier to use << and >>, see next section.

## 4.7 Display cell value

To simply display the value of a cell's numeric field in a text field, you can use the syntax `<<⟨reference⟩>>` which will be replaced by the cell's numeric value `⟨reference⟩` where `⟨reference⟩` can be relative or absolute. If what's between << and >> is not a reference, then nothing is done. The `<reference>` must not contain any spaces. If you write `<< a1>>` then the space means that the reference is not recognized.

```
\begin{spreadtab}{{tabular}{|c|c||c|}}\hline
    23 & 32 & Moy $= \frac{<<a1>>+<<b1>>}{2} = :={(a1+b1)/2}$\\\hline
\end{spreadtab}
```

| | | |
|---|---|---|
| 23 | 32 | Moy = $\frac{23+32}{2}$ = 27.5 |

Reference delimiter characters are set to << and >> by default. They can be modified with the key `display marks`. If you write `display marks={|;|}`, you'll have to write `|<reference>|`:

```
\begin{spreadtab}[display marks={|;|}]{{tabular}{|c|c||c|}}\hline
    23 & 32 & Mean $= \frac{|a1|+|b1|}{2} = :={(a1+b1)/2}$\\\hline
\end{spreadtab}
```

| | | |
|---|---|---|
| 23 | 32 | Mean = $\frac{23+32}{2}$ = 27.5 |

## 4.8 Use `\multicolumn`

The spreadtab package supports the `\multicolumn{⟨number⟩}{⟨type⟩}{⟨content⟩}` syntax, which merges `⟨number⟩` cells into a cell of the type and content specified in the arguments.

By using `\multicolumn`, spreadtab even maintains consistency in cell referencing. On this table where cells are merged, the table cells contain the absolute references seen by spreadtab:

<table>
<tr><td>a1</td><td>b1</td><td>c1</td><td>d1</td><td>e1</td><td>f1</td><td>g1</td></tr>
<tr><td>a2</td><td colspan="2">b2</td><td>d2</td><td>e2</td><td>f2</td><td>g2</td></tr>
<tr><td colspan="3">a3</td><td>d3</td><td colspan="2">e3</td><td>g3</td></tr>
</table>

So, whatever the number of cells merged, the next cell has a column number that takes into account the number of cells merged.

On the last line, cells `a3`, `b3` and `c3` are merged, and if cell `a3` contains a formula, cells `b3` and `c3` *doesn't exist* for spreadtab: these cells can't be referred to anywhere.

Here's an example where each number on the top line is the product of the 2 numbers below it:

```
\newcolumntype{K}[1]{@{}>{\centering\arraybackslash}p{#1cm}@{}}
\begin{spreadtab}{{tabular}{*6{K{0.5}}}}
    \cline{2-5}
    &\multicolumn{2}{|K{1}|}{:={a2*c2}} & \multicolumn{2}{K{1}|}{:={c2*e2}} &\\\hline
    \multicolumn{2}{|K{1}}{:=8}&\multicolumn{2}{|K{1}}{:=7}&\multicolumn{2}{|K{1}|}{:=6}\\\hline
\end{spreadtab}
```

<table>
<tr><td></td><td colspan="2">56</td><td colspan="2">42</td><td></td></tr>
<tr><td colspan="2">8</td><td colspan="2">7</td><td colspan="2">6</td></tr>
</table>

Note that the numeric field marker ":=" is required in every cell containing the command `\multicolumn`. If this marker didn't exist, the entire cell - i.e. `\multicolumn{2}{|c|}{⟨formula⟩}` would be considered a formula.

# 5 Macro-functions

## 5.1 Math macro-functions

### 5.1.1 Sum cells

The `sum` function is used to sum one or more cell ranges.

Its syntax is `sum(⟨range 1⟩;⟨range 2⟩;...;⟨range n⟩)`, where a cell range is:

- either an isolated cell such as ”`a1`” or ”`[2,1]`”;
- either a rectangular area delimited by the upper left and lower right cell in this way `⟨cell 1⟩:⟨cell 2⟩`.

In cell ranges, empty cells or cells containing only text are considered to contain the number 0. The same applies to cells masked by `\multicolumn`.

Relative and absolute references can be used together, as the user sees fit. Here’s an example of summing Pascal’s triangle coefficients:

```
\begin{spreadtab}{{tabular}{*5c}}
    \multicolumn{5}{c}{sum=:={sum(a2:e6)}}\\
    [0,1] & [-1,1]+[0,1] & [-1,1]+[0,1] & [-1,1]+[0,1] & [-1,1]\\
    [0,1] & [-1,1]+[0,1] & [-1,1]+[0,1] & [-1,1]       &       \\
    [0,1] & [-1,1]+[0,1] & [-1,1]       &              &       \\
    [0,1] & [-1,1]       &              &              &       \\
    1     &              &              &              &
\end{spreadtab}
```

| sum=31 | | | | |
|---|---|---|---|---|
| 1 | 4 | 6 | 4 | 1 |
| 1 | 3 | 3 | 1 | |
| 1 | 2 | 1 | | |
| 1 | 1 | | | |
| 1 | | | | |

### 5.1.2 Macro-function `sumprod`

The `sumprod` function multiplies the corresponding elements of 2 or more ranges and then adds these products together.

Its syntax is: `sumprod(⟨range 1⟩;⟨range 2⟩;...;⟨range n⟩)`. All rectangular ranges must have the same dimensions.

Here’s a simple example of calculating the average age of a group of children aged between 10 and 15:

```
\begin{spreadtab}{{tabular}{r*6c}}
    @Ages   & 10 & 11 & 12 & 13 & 14 & 15\\
    @Number & 5  & 8  & 20 & 55 & 9  & 3 \\\hline
    \multicolumn{7}{c}{Mean = :={sumprod(b1:g1;b2:g2)/sum(b2:g2)}}
\end{spreadtab}
```

| Ages | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|
| Number | 5 | 8 | 20 | 55 | 9 | 3 |
| Mean = 12.64 | | | | | | |

As with the `sum` macro function, text cells, whether empty or merged by a `\multicolumn`, are considered to contain the number 0.

### 5.1.3 GCD and LCM

The macro functions `gcd` and `lcm` calculate the Greatest Common Divisor (GCD) and the Least Common Multiple (LCMM) of numbers passed as arguments and separated by commas:

`gcd(⟨nombre1⟩,⟨nombre2⟩,...,⟨nombreN⟩)`
`lcm(⟨nombre1⟩,⟨nombre2⟩,...,⟨nombreN⟩)`

```
\begin{spreadtab}{{tabular}{|r|r|r||c|c|}}\hline
    \multicolumn3{|c||}{@Numbers}& @GCD & @LCM \\\hline
    24 & 18 & 12 & \STcopy{v}{gcd(a2,b2,c2)} & \STcopy{v}{lcm(a2,b2,c2)}\\
    15 & 10 & 25 & & \\
    16 & 12 & 15 & & \\\hline
\end{spreadtab}
```

| Numbers | | | GCD | LCM |
|---|---|---|---|---|
| 24 | 18 | 12 | 6 | 72 |
| 15 | 10 | 25 | 5 | 150 |
| 16 | 12 | 15 | 1 | 240 |

### 5.1.4 Scientific Notation

The macro function `scitodec` allows converting a number written in scientific notation to a decimal number. The syntax is `scitodec(`*⟨text⟩*`)`, where *⟨text⟩* is:

1. a sequence of characters in the syntax *⟨mantissa⟩*`EE`*⟨exponent⟩*, where the *⟨mantissa⟩* is a decimal number and the *⟨exponent⟩* is an integer. The ”E”s can be uppercase or lowercase;
2. a reference to the *text* field of a cell, which must contain characters following the syntax described in the previous point.

```
\begin{spreadtab}{{tabular}{|r|r|}}\hline
    @Scientific notations    & @Decimal \\\hline
    @4EE2                    & \STcopy{v}{scitodec([-1,0])}\\
    @-3.1EE-3                & \\
    @15ee5                   & \\
    @-0.025ee7               & \\
    @2.125EE0                & \\
    @3.1575EE-4              & \\\hline
\end{spreadtab}
```

| Scientific notations | Decimal |
|---:|---:|
| 4EE2 | 400 |
| -3.1EE-3 | -0.0031 |
| 15ee5 | 1500000 |
| -0.025ee7 | -250000 |
| 2.125EE0 | 2.125 |
| 3.1575EE-4 | 0.00031575 |

The `l3fp` engine natively understands numbers written in scientific notation in the form *⟨a⟩*`e`*⟨b⟩* but using this syntax is *impossible* to use with spreadtab because the number `4e3`, which is 4000, would be understood by spreadtab as 4 followed by the content of the cell `e3`.

### 5.1.5 Identity

The simplest macro function is `id`, since `id(`*⟨number⟩*`)` returns the *⟨number⟩* enclosed in brackets. Mathematically, it's of little use, but with spreadtab, it lets you pass mathematical expressions to macros that don't admit them. For example, it can be used in the cell ranges of `sum`.

In the code below, the `id` macro is used to determine the cell range to be added with `sum`. In this example, the value of the numeric field in cell `a2` is 8. Therefore, `sum([0,-1]:[id(a2-1),-1])/a2` is equivalent to `sum([0,-1]:[7,-1])/8`:

```
\begin{spreadtab}{{tabular}{r*{10}c}}
    @Int from 1 to 10      & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10\\
    Mean of :={8} first int & sum([0,-1]:[id(a2-1),-1])/a2         \\
\end{spreadtab}
```

| | | | | | | | | | | |
|---:|:-:|:-:|:-:|:-:|:-:|:-:|:-:|:-:|:-:|:-:|
| Int from 1 to 10 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Mean of 8 first int | 4.5 | | | | | | | | | |

## 5.2 Comparison macro-functions

There are no more comparison macros since the `l3fp` engine has a ternary operator *⟨a⟩*`?`*⟨b⟩*`:`*⟨c⟩*. If the test *⟨a⟩* is true, *⟨b⟩* is returned; otherwise, *⟨c⟩* is returned.

On the other hand, spreadtab offers the macro function `test` which has a special syntax with 2 arguments:

`test(csv of ⟨tests⟩)⟨optional spaces⟩(csv of ⟨texts⟩)`

The *⟨tests⟩* are performed from left to right and the first one that is `true`, the corresponding *⟨text⟩* is returned. If there are $n$ *⟨test⟩*, the *⟨text⟩* of index $n + 1$ will be returned if all the tests are false.

If the number of *⟨texts⟩* is insufficient compared to the number of *⟨tests⟩*, the missing *⟨texts⟩* are considered empty.

```
\begin{spreadtab}{{tabular}{c|c}}
    \hline
    @Month & @Season \\\hline
    1     & \STcopy v{test("a2=1||a2>10 , a2<5 , a2<8")
                      (Winter,Spring,Summer,Fall)} \\
    \STcopy v{a2+1} \STxp{\STrep{11}{& \\}}% 11 empty lines
    \hline
\end{spreadtab}
```

| Month | Season |
|---|---|
| 1 | Winter |
| 2 | Spring |
| 3 | Spring |
| 4 | Spring |
| 5 | Summer |
| 6 | Summer |
| 7 | Summer |
| 8 | Fall |
| 9 | Fall |
| 10 | Fall |
| 11 | Winter |
| 12 | Winter |

The part of the formula to be copied between the characters ”"” indicates which part should be taken into account for the offset when copying. It can be modified with the key `copy char`.

If you wish to use the logical connector &&, it is *necessary* to enclose it in braces. In this way, the cell to be copied could contain the following equivalent tests

```
\STcopy v{test("a2>2{&&}a2<6,a2>5{&&}a2<9,a2>8{&&}a2<12")
          (Spring,Summer,Fall,Winter)}
```

## 5.3 Macro-functions manipulating dates

### 5.3.1 Date to number with `engshortdatetonum`

The macro `engshortdatetonum` converts a short date like 1789/7/14 to an integer which is the number of days passed since the 1st March of the year 0. It is important to note that this macro-function requires a *textual* argument and not a number or the result of a mathematical calculation. Therefore, if the argument of this macro-function refers to a cell, that cell *must* be a text cell, i.e. a cell containing ‘`@`’ or ‘`:={}`’.

```
\begin{spreadtab}{{tabular}{cc}}
    @1789/7/14          & engshortdatetonum(a1)\\
    2001/1/1 :={}       & engshortdatetonum(a2)\\\hline
\end{spreadtab}
```

| | |
|---|---|
| 1789/7/14 | 653554 |
| 2001/1/1 | 730791 |

Another macro-function provides the same feature but with a long date like ‘December 25, 2024’

```
\begin{spreadtab}{{tabular}{cc}}
    @January 1, 2001   & englongdatetonum(a1)\\
    @December 25, 2024 & englongdatetonum(a2)
\end{spreadtab}
```

| | |
|---|---|
| January 1, 2001 | 730791 |
| December 25, 2024 | 739550 |

### 5.3.2 From a number to a date

Several macro functions can be used to translate a number into a date. All these macro-functions have in common that their result is *text*. As a result, immediately after a number-to-text macro function has been evaluated, any other evaluation in the cell is stopped and the cell becomes a text cell. If there was text in the cell with the formula, the result of the formula will be inserted instead of `:={⟨formula⟩}`. As the cell is made up of text only, it can no longer be processed mathematically.

These macro-functions are:

- `numtoengshortdate` translate a number into a short date like ‘1789/7/14’;
- `numtoenglongdate` translate a number into a long date like ‘July 14, 1789’;
- `numtoengmonth` given a number representing a date, it finds the name of the month;
- `numtoengday` same as above but it finds the name of the day.

Here is an example in which we consider 30 days before and 30 days after 2009/6/1. For each of these 2 dates, we calculate the short date, long date, month and day of the week

```
\begin{spreadtab}{{tabular}{cc}}                                    \hline
    \multicolumn{2}{|c|}{@2009/6/1}                               \\\hline\hline
    30      & numtoengshortdate(engshortdatetonum(a1)+[-1,0])\\
    30      & numtoenglongdate(engshortdatetonum(a1)+[-1,0]) \\
    30      & numtoengmonth(engshortdatetonum(a1)+[-1,0])    \\
    30      & numtoengday(engshortdatetonum(a1)+[-1,0])      \\\hline
    -30     & numtoengshortdate(engshortdatetonum(a1)+[-1,0])\\
    -30     & numtoenglongdate(engshortdatetonum(a1)+[-1,0]) \\
    -30     & numtoengmonth(engshortdatetonum(a1)+[-1,0])    \\
    -30     & numtoengday(engshortdatetonum(a1)+[-1,0])
\end{spreadtab}
```

| 2009/6/1 | |
|---|---|
| 30 | 2009/7/1 |
| 30 | July 1, 2009 |
| 30 | July |
| 30 | wednesday |
| -30 | 2009/5/2 |
| -30 | May 2, 2009 |
| -30 | May |
| -30 | saturday |

## 5.4 Coordinate macro-functions

### 5.4.1 `tag`, `cell`, `value` and `\STtag`

Rather than referring to a cell by its coordinates, which are difficult to remember and change if a row or column is inserted, it is sometimes much more practical to give a cell a name and refer to it later by that name.

The macro function `tag` has the syntax `tag(⟨name⟩)`. This causes the cell in which it is located to be named. It is not really a macro function like the others since it returns nothing when placed in a formula: it disappears without affecting the mathematical result. Thus, `tag(⟨name⟩)` can be written *anywhere* in the numeric field of a cell. The ⟨*name*⟩ can be any alphanumeric string, but it is not recommended to use a letter and a number, which could be interpreted as a cell reference and therefore modified during a copy operation with `\STcopy`. This macro function has an additional action: it globally stores the numeric value of the cell in which it is located via a `\gdef`, so that it can be called later *outside the table* using the expandable command `\STtag{⟨name⟩}`.

Later in the table, instead of using the coordinates of the cell ⟨*name*⟩, one can write `cell(⟨name⟩)`, which is a macro function that returns the coordinates of the previously named cell. For example, if `tag(⟨name⟩)` was written in cell `B3`, then later writing `cell(⟨name⟩)` will cause this macro function to return `B3`.

Here is an example where we sum cells and give the name `foo` to the first number and `bar` to the last. It can be observed that `tag(bar)` is between 1 and 9, and ultimately, since this macro function disappears, the numeric field of the cell will contain 19:

```
\begin{spreadtab}{{tabular}{r@{}r}}
       & 15tag(foo)             \\
    @+ & 37                     \\
    @+ & 13                     \\
    @+ & 48                     \\
    @+ & 1tag(bar)9             \\\cline{2-2}
       & sum(cell(foo):cell(bar))tag(baz)
\end{spreadtab}

foo=\STtag{foo}, bar=\STtag{bar}, baz=\STtag{baz}
```

| | |
|---|---|
| | 15 |
| + | 37 |
| + | 13 |
| + | 48 |
| + | 19 |
| | 132 |

foo=15, bar=19, baz=132

To pass values between tables calculated by `spreadtab`, it is possible to tag the cell in the first table using the macro function `tag(⟨name⟩)`, and then in the second table, refer to the value *via* the tag with `value(⟨name⟩)`.

```
\begin{spreadtab}{{tabular}{|c|c|c|}}\hline
    100 & 75 & a1+b1tag(ab:sum)\\\hline
\end{spreadtab}

\begin{spreadtab}{{tabular}{|c|c|}}\hline
    value(ab:sum) & 1.20*value(ab:sum)\\\hline
\end{spreadtab}
```

| | | |
|---|---|---|
| 100 | 75 | 175 |

| | |
|---|---|
| 175 | 210 |

If the key `tag to aux` is ⟨*true*⟩, the assignments are also written to the auxiliary file, making them transferable from one compilation to another. It then becomes possible to place `\STtag{⟨name⟩}` before the table where `tag(⟨name⟩)` is specified.

### 5.4.2 `row` and `col`

Although at first sight less useful, spreadtab also provides the macro functions `row(`⟨*name*⟩`)` and `col(`⟨*name*⟩`)` that return the number of the row or column of the cell `tag(`⟨*name*⟩`)`. Here is an example of how to calculate the average of a number of values; the first and last values are tagged 'first' and 'last' and therefore, the number of values is `row(last)-row(first)+1`:

```
mean =
\begin{spreadtab}{{tabular}[b]{r}}
    7tag(first)\\9\\15\\6\\20\\13\\11\\55tag(last)\\
    \hline\hline
    sum(cell(first):cell(last))/(row(last)-row(first)+1)
\end{spreadtab}
```

| | |
|---|---:|
| | 7 |
| | 9 |
| | 15 |
| | 6 |
| | 20 |
| | 13 |
| | 11 |
| | 55 |
| mean = | 17 |

# 6 Particular care

## 6.1 Defining new commands with `\hline`

It may be useful to define a new command to produce, for example, a double horizontal line:

```
\newcommand\dline{\hline\hline}
```

and then try to use it in a table as in this simple example that computes the Fibonnacci sequence in the second line::

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 |

But, if you write the following code, there is a problem when compiling:

```
\newcommand\dline{\hline\hline}
\def\xtracol{&&&&&&}
\begin{spreadtab}{{tabular}{*9c}}
    0     & 1 & \STcopy>{b1+1} \STxp\xtracol\\\dline
    1     & 1 & \STcopy>{a2+b2}\STxp\xtracol
\end{spreadtab}
```

In the log file, you can read that `\FPeval` fails and complains: `! Improper alphabetic constant.`

The reason is simple, `\dline` in line 4 is not recognized by spreadtab as a horizontal rule and therefore, *it is placed in the cell in the next line.* For spreadtab, the cell `b1` contains:

```
\dline 1
```

To compile without error, the cell `a2` *must* contain a numeric field marker:

```
\newcommand\dline{\hline\hline}
\def\xtracol{&&&&&&}
\begin{spreadtab}{{tabular}{*9c}}
    0     & 1 & \STcopy>{b1+1} \STxp\xtracol\\\dline
    :={1} & 1 & \STcopy>{a2+b2}\STxp\xtracol
\end{spreadtab}
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 |

## 6.2 The use of `\multicolumn` and `\SThidecol`

Firstly, in normal use, joint use of `\multicolumn` and `\SThiderow` should not happen, and most users should not encounter this situation and should not read this section.

Here is the problem: first, a hidden column *must not* contain a cell with the command `\multicolumn`! But what happens if a hidden column hides cells merged with `\multicolumn`?

In general, there is no compilation error or error messages, but there are some subtleties about the references that are a bit turned upside down in the line after the `\multicolumn` command…

Let's take an example, and let's say that, in the following table, we want to merge the cell `b2` to `h2` and we also want to hide the colomns `c`, `d` and `f`, here in gray:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a1 | b1 | c1 | d1 | e1 | f1 | g1 | h1 | i1 | j1 |
| a2 | b2 | | | | | | | i2 | j2 |

There are 4 visible merged cells, so we write `\multicolumn{4}` because hidden columns are never taken into account when counting the number of `\multicolumn`.

Then we count 4 letters from `b` (this letter included): we obtain the letter `e`. In the range `b-e`, let's count: 2 gray hidden columns are included (`c` and `d`) and 1 hidden column is not included (`f`). These numbers are important to understand the following, also let's call them $x$ and $y$ in the general case.

The rule is:

- it is necessary to add $y$ signs `&` after `\multicolumn` (in the example above, it would be 1);
- references to columns of cells after the `\multicolumn` will be shifted $x$ to the beginning of the alphabet. For the example given, if we want to refer to the cell `i2`, we should write `g2` instead of `i2`.

Here is an example with a similar structure to the previous ($x = 2$ and $y = 1$) with simple formulas: add 1 to the number above.

```
\begin{spreadtab}{{tabular}{|*{7}{c|}}}\hline
    1   & 2    & \SThidecol3 & \SThidecol4 & 5& \SThidecol6 & 7& 8& 9    & 10  \\\hline
    a1+1& \multicolumn4{l|}{:={b1+1}}&                            & i1+1 & j1+1\\\hline
    a2+1& b2+1 &             &             &  &             &  &  & g2+1 & h2+1\\\hline
\end{spreadtab}
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 7 | 8 | 9 | 10 |
| 2 | 3 | | | | 10 | 11 |
| 3 | 4 | | | | 11 | 12 |

## 6.3 Messages delivered by spreadtab

The package delivers error messages and aborts compilation in these cases:

- a circular reference is found in a cell. In this case, the dependent cells are displayed;
- a cell refers to an empty cell or a text cell when a non-empty numeric field is expected;
- a cell refers to an undefined cell (outside the table);
- a cell refers to a cell merged by a `\multicolumn` command;
- a relative reference has bad syntax.[2]

The package can deliver informative messages (in the log file), which it does by default. The key `messages`, depending on whether it is `true` or `false`, enables or disables the transmission of information messages.

To understand the meaning of these messages, let's take a simple table:

```
\begin{spreadtab}{{tabular}{|cccc|c|}}\hline
    b1+1 & c1+1 & d1+1 & 10 & a1+b1+c1+d1\\\hline
\end{spreadtab}
```

| | | | | |
|---|---|---|---|---|
| 13 | 12 | 11 | 10 | 46 |

Here are the messages deliverd by spreadtab:

```
New spreadtab: \begin{tabular}{|cccc|c|}
* reading tab: ok
* computing formulas:
     cell A1-B1-C1
     cell B1
     cell C1
     cell D1
     cell E1
* building tab: ok
End of spreadtab: \end{tabular}
```

Preceded by a star, we recognize the 3 steps necessary for spreadtab to complete its task: reading the table, calculation of the formulas and building the final table.

For the second step, cells are evaluated from top to bottom, left to right: at line 4 in the code above, spreadtab says that it begins by trying to calculate the first cell `A1`. After a dash, we see that for this, it must first compute the cell `B1`, which itself requires that the cell `C1` is calculated: the latter can be calculated since it depends only on `D1` which is a cell containing the number 10.

In the following (lines 5 to 8), there is only one cell per line which means that when spreadtab tries to evaluate the cell, either it contains a number or dependent cells are already calculated.

## 6.4 Debug

To ease the use spreadtab, a debug mode is available. It is activated when the key `debug` is not empty and contains at least one of the words ⟨*formula*⟩, ⟨*text*⟩ or ⟨*cellcode*⟩, separated by commas. These words each cause a debugging table to be displayed:

- ⟨*formula*⟩: displays all the numeric fields and the ends of lines;
- ⟨*text*⟩: displays all the textual fields;
- ⟨*cellcode*⟩ displays the internal code of the cells. Indeed, spreadtab assigns a code to every cell when it reads the table. Here are the possible values of this code:
  - $-1$ if the cell is merged with `\multicolumn`;
  - 0 if the cell is a text cell or is empty;
  - 1 if the numeric field of the cell contains a formula which will be computed later;
  - 2 if the numeric field of the cell contains a number.

You can add the value ⟨*show tab*⟩ to force the display of the resulting table, which is otherwise not displayed.

Here's a table for which we display the 3 debugging tables and also the final table obtained:

```
\STset{debug={text,formula,cellcode,show tab}}
\begin{spreadtab}{{tabular}{|rr|r|}}\hline
    @$x$              &@$y$                & @$x+y$\\\hline\hline
    22                & 54                & \STcopy{v3}{a2+b2} \\
    43                & 65                & \\
    49                & 37                & \\\hline
    $Sx=:={a2+a3+a4}$ & $Sy=:={b2+b3+b4}$ & $Sx+Sy=:={}$\\\hline
    \multicolumn2{|r|}{$Sy-Sx=:={b5-a5}$}  & @\multicolumn1c{}\\\cline{1-2}
\end{spreadtab}
```

| | A | B | C |
|---|---|---|---|
| 1 | $x$ | $y$ | $x+y$ |
| 2 | := | := | := |
| 3 | := | := | := |
| 4 | := | := | := |
| 5 | $Sx=:=$ | $Sy=:=$ | $Sx+Sy=:=$ |
| 6 | \multicolumn 2{\|r\|}{$Sy-Sx=:=$} | | \multicolumn 1c{} |

| | A | B | C | \hline |
|---|---|---|---|---|
| 1 | | | | \\\hline \hline |
| 2 | 22 | 54 | a2`+b2 | \\ |
| 3 | 43 | 65 | a3`+b3 | \\ |
| 4 | 49 | 37 | a4`+b4 | \\\hline |
| 5 | a2`+a3+a4 | b2`+b3+b4 | a5`+b5 | \\\hline |
| 6 | b5`-a5 | | | \\\cline {1-2} |

| | A | B | C |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 2 | 2 | 1 |
| 3 | 2 | 2 | 1 |
| 4 | 2 | 2 | 1 |
| 5 | 1 | 1 | 1 |
| 6 | 1 | -1 | 0 |

| $x$ | $y$ | $x+y$ |
|---|---|---|
| 22 | 54 | 76 |
| 43 | 65 | 108 |
| 49 | 37 | 86 |
| $Sx = 114$ | $Sy = 156$ | $Sx + Sy = 270$ |
| $Sy - Sx = 42$ | | |

These 3 debugging tables can help you understand the inner workings of spreadtab. Table 2 shows that all cells with a numeric field have an internal code of 1 or 2 (see Table 3) and have an associated numeric field marker ”`:=`” (see Table 1). This marker represents the place where the result of the numeric field calculation will be inserted by substitution. Once the numerical fields have been calculated, the cells in the final table are reconstituted from the contents of the textual fields in Table 1, by simple substitution.

In debugging tables, cells containing coordinates are only grayed out if the `colortbl` package has been loaded.

## 7 List of keys

All settings concerning spreadtab are made from version 0.6 onwards by a system of ⟨*keys*⟩=chevronsvalue which can be found:

1. in the `\STset` argument, in which case the settings specified concern all future arrays;
2. in the optional argument of `\begin{spreadtab}` or `spreadtab`, in which case the settings specified concern only the current array.

Here is the list of ⟨*keys*⟩ and their default values:

| ⟨*key*⟩ | ⟨*value*⟩ | see page | Details |
|---|---|---|---|
| `tabline sep` | ⟨\\⟩ | 3 | separator between table rows |
| `text mark` | ⟨@⟩ | 4 | the ⟨*value*⟩ is detokenized: if it is present in a cell, spreadtab understands it as a text cell |
| `numeric mark` | ⟨:=⟩ | 5 | the argument between braces that follows is the numeric field |
| `freeze char` | ⟨!⟩ | 5 | detokenized character which, if present in front of the letter or number of a reference, does not increment it when the formula is copied |
| `pretab code` | ⟨⟩ (empty) | 7 | code executed just before displaying the table |

| ⟨*key*⟩ | ⟨*value*⟩ | see page | Details |
|---|---|---|---|
| `posttab code` | ⟨⟩ (empty) | 7 | code executed just after displaying the table |
| `dec sep` | ⟨.⟩ | 7 | decimal separator for numbers after table calculation |
| `autoround` | ⟨⟩ (empty) | 8 | An empty value is the normal behavior; otherwise, a positive ⟨*integer*⟩ is expected and indicates the rank of the internal calculation rounding. |
| `save list` | ⟨⟩ (empty) | 9 | csv list of the form ⟨*macro*⟩=⟨*absolute reference*⟩: each number contained in the referenced cells is stored (globally) in a macro. This key is reset to empty after each table. |
| `aux save list` | ⟨⟩ (empty) | 9 | same syntax and behavior as above, but assignments are also written to the auxiliary file. This key is reset to empty after each table. |
| `display marks` | ⟨<<;>>⟩ | 10 | markers around an absolute reference in a text field to display the number contained in the referenced cell |
| `copy char` | ⟨"⟩ | 13 | in the argument to `\STcopy`, sets the beginning and end of the part of the formula which must be taken into account for the offsets when copying |
| `tag to aux` | ⟨*false*⟩ | 14 | when ⟨*true*⟩, assignments made by the macro function `tag` are also written to the auxiliary file. |
| `messages` | ⟨*false*⟩ | 16 | allows `spreadtab` to send messages to the log file about what it's doing |
| `debug` | ⟨⟩ (empty) | 17 | must contain words from ⟨*text*⟩, ⟨*formula*⟩ or ⟨*cellcode*⟩ in csv format to display the corresponding debugging table. If the word ⟨*show tab*⟩ is added, the table is also displayed. This key is reset to empty after each table. |

To return to the default values, you can run the `\STreset` macro at any time.

⁂

Please let me know by `email` any bugs, regressions or new features to implement that you think might be useful.