Simple Spreadsheets

Introduction
Occasionally a question pops up on the Con\TeX t mailing list and answering it becomes a nice distraction from a boring task at hand. The spreadsheet module is the result of such a diversion. As with more support code in Con\TeX t, this is not a replacement for ‘the real thing’ but just a nice feature for simple cases. The module is loaded with
\usemodule{spreadsheet}
So this is (at least currently) not one of the core functionalities but an add-on. Of course some useful extensions might appear in the future.

Spreadsheet tables
We can use Lua in each cell, because under the hood it is all Lua. There is some basic parsing applied so that we can use the usual A...Z variables to access cells.
\startspreadsheettable[test]
\startrow
\startcell 1.1 \stopcell
\startcell 2.1 \stopcell
\stoprow
\startrow
\startcell 2.1 \stopcell
\startcell 2.2 \stopcell
\stoprow
\startrow
\stoprow
\stopspreadsheettable

The rendering is shown in figure 1. Keep in mind that in Lua all calculations are done using floats, at least in Lua versions with version numbers preceding 5.3. The last cell can also look like this:
\startcell
function()
local s = 0
for i=1,2 do
 for j=1,2 do
  s = s + dat[i][j]
 end
end
return s
end
\stopcell

Figure 1. A simple spreadsheet.
The content of a cell is either a number or a function. In this example we just loop over the (already set) cells and calculate their sum. The dat variable accesses the grid of cells.

```plaintext
\startcell
function()
    local s = 0
    for i=1,2 do
        for j=1,2 do
            s = s + dat[i][j]
        end
    end
    tmp.total = s
\stopcell
```

In this variant we store the sum in the table tmp which is local to the current sheet. Another table is fnc where we can store functions. This table is shared between all sheets. There are two predefined functions:

- `sum(columnname,firstrow,lastrow)`
- `fmt(specification,n)`

The `sum` function works top-down in columns, and roughly looks like this:

```plaintext
function sum(currentcolumn,firstrow,lastrow)
    local r = 0
    for i = firstrow, lastrow do
        r = r + cells[currentcolumn][i]
    end
    return r
end
```

The last two arguments are optional:

- `sum(columnname,lastrow)`
- `sum(columnname)`

This is equivalent to:

```plaintext
function sum(currentcolumn,firstrow,lastrow)
    local r = 0
    for i = firstrow, lastrow do
        r = r + cells[currentcolumn][i]
    end
    return r
end

while:

sum(columnname)
```

boils down to:

```plaintext
function sum(currentcolumn)
    local r = 0
    for i = 1, currentrow do
        r = r + cells[currentcolumn][i]
    end
    return r
end
end
```
Figure 2. Cells can be (complex) functions.

Let’s see this in action:
\begin{spreadsheettable}[test]
\startrow
\startcell 1.1 \stopcell \startcell 2.1 \stopcell \stoprow
\startrow
\startcell 2.1 \stopcell \startcell 2.2 \stopcell \stoprow
\startrow
\startcell
function()
local s = 0
for i=1,2 do
for j=1,2 do
s = s + dat[i][j]
end
end
context.bold(s)
end
\stopcell
\startcell
function()
local s = 1
for i=1,2 do
for j=1,2 do
s = s * dat[i][j]
end
end
context.bold(fmt("@.1f",s))
end
\stopcell
\stoprow
\stopspreadsheettable

The result is shown in figure 2. Watch the fmt call: we use an at sign instead of a percent to please \TeX. Keep in mind that we’re typesetting and that doing complex calculations is not our main objective. A typical application of this module is in making bills, for which you can combine it with the correspondence modules. We leave that as an exercise for the reader and stick to a simple example.
\begin{spreadsheettable}[test]
\startrow
\startcell[align=flushleft,width=8cm] "item one" \stopcell
\startcell[align=flushright,width=3cm] @ "0.2f EUR" 3.50 \stopcell
\stoprow
\startrow
\startcell[align=flushleft] "item two" \stopcell
\startcell[align=flushright] @ "0.2f EUR" 8.45 \stopcell
\stoprow
\startrow
\startcell[align=flushleft] "tax 19\percent" \stopcell
\startcell[align=flushright] @ "0.2f EUR" 0.19 * (B[1]+B[2]) \stopcell
\stoprow
\stoprow
\startrow
Here (and in figure 3) you see a quick and more readable way to format cell content. The @ in the template is optional, but needed in cases like this:

@ "(@0.2f) EUR" 8.45

An @ is only prepended when no @ is given in the template.

In practice this table we can be less specific and let \( \sum \) behave more automatic. This can be simplified (see figure 4) and made a bit nicer looking.

\begin{spreadsheettable}[test][frame=off]
\begin{row}
  \startcell[align=flushleft,width=8cm] "The first item" \stopcell
  \startcell[align=flushright,width=3cm] @ "0.2f EUR" 3.50 \stopcell
\end{row}
\begin{row}
  \startcell[align=flushleft] "The second item" \stopcell
  \startcell[align=flushright] @ "0.2f EUR" 8.45 \stopcell
\end{row}
\begin{row}
  \startcell[align=flushleft] "The third item" \stopcell
  \startcell[align=flushright] @ "0.2f EUR" 5.90 \stopcell
\end{row}
\begin{row}[topframe=on]
  \startcell[align=flushleft] "VAT 19\%" \stopcell
  \startcell[align=flushright] @ "0.2f EUR" 0.19 * sum(B) \stopcell
\end{row}
\begin{row}[topframe=on]
  \startcell[align=flushleft] "\bf Grand total" \stopcell
  \startcell[align=flushright] @ "0.2f EUR" sum(B) \stopcell
\end{row}
\end{spreadsheettable}

\begin{table}
\begin{tabular}{l|l}
\hline
\textbf{Item} & \textbf{Price} \\
\hline
item one & 3.50 EUR \\
item two & 8.45 EUR \\
tax 19\% & 2.27 EUR \\
total 1 & 14.22 EUR \\
total 2 & 14.22 EUR \\
total 3 & 42.66 EUR \\
\hline
\end{tabular}
\end{table}

\textit{Figure 3.} Cells can be formatted by using @ directives.
The first item 3.50 EUR
The second item 8.45 EUR
The third item 5.90 EUR
VAT 19% 3.39 EUR
Grand total 21.24 EUR

Figure 4. The sum function accumulates stepwise.

<table>
<thead>
<tr>
<th>first</th>
<th>0 &quot;[@i]&quot; 1</th>
<th>[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>second</td>
<td>= 2</td>
<td>2</td>
</tr>
<tr>
<td>third</td>
<td>! 3</td>
<td></td>
</tr>
<tr>
<td>fourth</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>total one</td>
<td>sum(C)</td>
<td>10</td>
</tr>
<tr>
<td>total two</td>
<td>= sum(C)</td>
<td>20</td>
</tr>
</tbody>
</table>

Figure 5. Cells can be hidden by ! and can contain strings only.

There are a few more special start characters. This is demonstrated in figure 5. An = character is ignored.\footnote{Taco suggested to support this because some spreadsheet programs use that character to flush a value.} When we start with an !, the content is not typeset. Strings can be surrounded by single or double quotes and are not really processed.

\startspreadsheettable[test][offset=1ex]
\startrow
  \startcell[align=flushleft] "first" \stopcell
  \startcell[align=flushleft] '\type{@ "[@i]" 1}' \stopcell
  \startcell[align=flushright,width=3cm] @ "[@i]" 1 \stopcell
\stoprow
\startrow
  \startcell[align=flushleft] "second" \stopcell
  \startcell[align=flushleft] '=' 2' \stopcell
  \startcell[align=flushright] = 2 \stopcell
\stoprow
\startrow
  \startcell[align=flushleft] "third" \stopcell
  \startcell[align=flushleft] '! 3' \stopcell
  \startcell[align=flushright] ! 3 \stopcell
\stoprow
\startrow
  \startcell[align=flushleft] "fourth" \stopcell
  \startcell[align=flushleft] '\type{4}' \stopcell
  \startcell[align=flushright] 4 \stopcell
\stoprow
\startrow
  \startcell[align=flushleft] "bf total one" \stopcell
  \startcell[align=flushleft] '\type{sum(C)}' \stopcell
  \startcell[align=flushright] sum(C) \stopcell
\stoprow
\stopspreadsheettable
The sum function is clever enough not to include itself in the summation. Only preceding cells are taken into account, given that they represent a number.

Normal tables
In the previous examples we used \text{T}_{\alpha X} commands for structuring the sheet but the content of cells is Lua code. It is also possible to stick to a regular table and use specific commands to set and get cell data.

\begin{table}[align=center]
\begin{tabular}{|c|c|}
\hline
\text{A[1]} + \text{A[2]} & \text{B1 + B2} \\
\hline
\text{fmt("@0.3f",(sum(A,1,2)) / 10)} & \\
\hline
\end{tabular}
\end{table}

The method to use depends on the complexity of the table. If there is more text than data then this method is probably more comfortable.

A few settings
It’s possible to influence the rendering. The following example demonstrates this. We don’t use any formatting directives.

\begin{spreadsheettable}[test]
\begin{tabular}{|c|}
\hline
\text{123456.78} \\
\hline
\text{1234567.89} \\
\hline
\text{A[1]} + \text{A[2]} \\
\hline
\end{tabular}
\end{spreadsheettable}

Figure 6. A sheet can be filled and accessed from regular tables.
Figure 7 demonstrates how this gets rendered by default. However, often you want numbers to be split in parts separated by periods and commas. This can be done as follows:

```latex
\definehighlight{BoldAndRed} [style=bold, color=darkred]
\definehighlight{BoldAndGreen} [style=bold, color=darkgreen]
\setupspreadsheet
  [test]
  [period={\BoldAndRed{.}},
   comma={\BoldAndGreen{,}},
   split=yes]
```

The Lua end

You can also use spreadsheets from within Lua. The following example is rather straightforward:

```lua
\startluacode
context.startspreadsheettable { "test" }
    context.startrow()
    context.startcell() context("123456.78") context.stopcell()
    context.stoprow()
    context.startrow()
    context.startcell() context("1234567.89") context.stopcell()
    context.stoprow()
    context.startrow()
    context.stoprow()
context.stopspreadsheettable()
\stopluacode
```

However, even more Lua-ish is the next variant:

```lua
\startluacode
local set = moduledata.spreadsheets.set
local get = moduledata.spreadsheets.get
moduledata.spreadsheets.start("test")
    set("test",1,1,"123456.78")
    set("test",2,1,"1234567.89")
moduledata.spreadsheets.stop()

context.bTABLE()
    context.bTR()
    context.bTD() context(get("test",1,1)) context.eTD()
    context.eTR()
    context.bTR()
    context.bTD() context(get("test",2,1)) context.eTD()
    context.eTR()
    context.bTR()
    context.bTD() context(get("test",3,1)) context.eTD()
    context.eTR()
context.eTABLE()
\stopluacode
```
Of course the second variant does not make much sense as we can do this way more efficient by not using a spreadsheet at all:

\startluacode
local A1, A2 = 123456.78, 1234567.89
context.bTABLE()
  context.bTR()
  context.bTD() context(A1) context.eTD()
  context.eTR()
  context.bTR()
  context.bTD() context(A2) context.eTD()
  context.eTR()
  context.bTR()
  context.bTD() context(A1+A2) context.eTD()
  context.eTR()
context.eTABLE()
\stopluacode

You can of course use format explicitly. Here we use the normal percent directives because we’re in Lua, and not in \TeX, where percentage signs are a bit of an issue.

\startluacode
local A1, A2 = 123456.78, 1234567.89
local options = { align = "flushright" }
context.bTABLE()
  context.bTR()
  context.bTD(options)
  context("%0.2f",A1)
  context.eTD()
  context.eTR()
  context.bTD(options)
  context("%0.2f",A2)
  context.eTD()
  context.eTR()
  context.bTD(options)
  context("%0.2f",A1+A2)
  context.eTD()
  context.eTR()
context.eTABLE()
\stopluacode

As expected and shown in figure 9, only the first and last variant get the numbers typeset nicely.

\begin{table}
\centering
\begin{tabular}{cccc}
123,456.78 & 123,456.78 & 123,456.78 & 123,456.78 \\
1,234,567.89 & 123,456.78 & 123,456.78 & 123,456.78 \\
1,358,024.67 & 1358024.67 & 1358024.67 & 1358024.67 \\
\end{tabular}
\caption{Spreadsheets purely done as \TeX\ Lua Document.}
\end{table}
**Helper macros**

There are two helper macros that you can use to see what is stored in a spreadsheet:

\texttt{\textbackslash inspectspreadsheet[test]}  
\texttt{\textbackslash showspreadsheet [test]}

The first command reports the content of \texttt{test} to the console, and the second one typesets it in the running text:

\begin{verbatim}
t={
  { 123456.78, 1234567.89, 1358024.67 },
}
\end{verbatim}

Another helper function is \texttt{\textbackslash doifelsespreadsheetcell}, You can use this one to check if a cell is set.

\begin{verbatim}
(1,1): \textbackslash doifelsespreadsheetcell[test]{1}{1}{set}{unset}  
(2,2): \textbackslash doifelsespreadsheetcell[test]{2}{2}{set}{unset}  
(9,9): \textbackslash doifelsespreadsheetcell[test]{9}{9}{set}{unset}
\end{verbatim}

This gives:

(1,1): set
(2,2): unset
(9,9): unset

There is not much more to say about this module, apart from that it is a nice example of a \TeX{} and Lua mix. Maybe some more (basic) functionality will be added in the future but it all depends on usage.

Hans Hagen  
PRAGMA ADE  
Hasselt NL