Simple Spreadsheets

ConTeXt MkIV

Hans Hagen

May 12, 2016

Contents

1 Introduction 1
2 Spreadsheet tables 1
3 Normal tables 6
4 A few settings 7
5 The LUA end 8
6 Helper macros 10

1 Introduction

Occasionally a question pops up on the ConTEXt 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 ConTEXt, 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.

2 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
    \startcell A[1] + B[1] \stopcell
  \stoprow
  \startrow
    \startcell 2.1         \stopcell
    \startcell 2.2         \stopcell
    \startcell A[2] + B[2] \stopcell
  \stoprow
  \startrow
    \startcell A[1] + B[1] \stopcell
    \startcell A[2] + B[2] \stopcell
    \startcell A[3] + B[3] \stopcell
  \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.

1.1	2.1	3.2
2.1	2.2	4.3
3.2	4.3	7.5

Figure 1 A simple spreadsheet.

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
```

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.

```
\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
end
\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:

```
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)
```

This is equivalent to:

```
function sum(currentcolumn,lastrow)
  local r = 0
  for i = 1, lastrow do
    r = r + cells[currentcolumn][i]
  end
  return r
end
```

While:

```
sum(columnname)
```

boils down to:

```
function sum(currentcolumn)
  local r = 0
  for i = 1, currentrow do
    r = r + cells[currentcolumn][i]
  end
  return r
end
```

Empty cells or cells that have no numbers are skipped. Let’s now see these functions in action:

```
\startspreadsheettable[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.

1.1	2.1
2.1	2.2
7.5	**10.7**

Figure 2 Cells can be (complex) functions.

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.

```
\startspreadsheettable[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
  \startrow
    \startcell[align=flushleft] "total 1" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" sum(B,1,3) \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "total 2" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" B[1] + B[2] + B[3] \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "total 3" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" sum(B) \stopcell
  \stoprow
\stopspreadsheettable
```

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
```

A `@` is only prepended when no `@` is given in the template.

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

Figure 3 Cells can be formatted by using `@` directives.

In practice this table we can be less specific and let `\sum` behave more automatical. That way the coding can be simplified (see figure 4) and also look nicer.

```
\startspreadsheettable[test][frame=off]
  \startrow
    \startcell[align=flushleft,width=8cm] "The first item" \stopcell
    \startcell[align=flushright,width=3cm] @ "0.2f EUR" 3.50 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "The second item" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" 8.45 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "The third item" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" 5.90 \stopcell
  \stoprow
  \startrow[topframe=on]
    \startcell[align=flushleft] "VAT 19\percent" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" 0.19 * sum(B) \stopcell
  \stoprow
  \startrow[topframe=on]
    \startcell[align=flushleft] "\bf Grand total" \stopcell
    \startcell[align=flushright] @ "0.2f EUR" sum(B) \stopcell
  \stoprow
\stopspreadsheettable
```

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.

There are a few more special start characters. This is demonstrated in figure 5. An = character is ignored.[1] 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] '\type{= 2}' \stopcell
    \startcell[align=flushright] = 2 \stopcell
  \stoprow
  \startrow
    \startcell[align=flushleft] "third" \stopcell
    \startcell[align=flushleft] '\type{! 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
  \startrow
    \startcell[align=flushleft] "\bf total two" \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.

3 Normal tables

In the previous examples we used TEX 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.

```
\bTABLE[align=middle]
  \bTR
```

[1] Taco suggested to support this because some spreadsheet programs use that character to flush a value.

first	`@ "[@i]" 1`	[1]
second	`= 2`	2
third	`! 3`	
fourth	`4`	4
total one	`sum(C)`	10
total two	`= sum(C)`	20

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

```
    \bTD \getspr{100} \eTD \bTD test \setspr{30} \eTD
  \eTR
  \bTR
    \bTD \getspr{20} \eTD \bTD \getspr{4+3} \eTD
  \eTR
  \bTR
    \bTD \getspr{A[1] + A[2]} \eTD
    \bTD \getspr{B1 + B2} \eTD
  \eTR
  \bTR
    \bTD[nx=2] \bf \getspr{(A[3] + B[3]) /100} \eTD
  \eTR
  \bTR
    \bTD[nx=2] \bf \getspr{fmt("@0.3f",(A[3] + B[3]) /100)} \eTD
  \eTR
  \bTR
    \bTD[nx=2] \bf \getspr{fmt("@0.3f",(sum(A,1,2)) / 10)} \eTD
  \eTR
\eTABLE
```

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.

100	test
20	7
120	37
1.57	
1.570	
12.000	

Figure 6 A sheet can be filled and accessed from regular tables.

4 A few settings

It's possible to influence the rendering. The following example demonstrates this. We don't use any

formatting directives.

```
\startspreadsheettable[test]
  \startrow
    \startcell   123456.78 \stopcell
  \stoprow
  \startrow
    \startcell  1234567.89 \stopcell
  \stoprow
  \startrow
    \startcell A[1] + A[2] \stopcell
  \stoprow
\stopspreadsheettable
```

123456.78
1234567.89
1358024.67

Figure 7 Formatting (large) numbers.

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:

```
\definehighlight[BoldAndRed]  [style=bold,color=darkred]
\definehighlight[BoldAndGreen][style=bold,color=darkgreen]

\setupspreadsheet
  [test]
  [period={\BoldAndRed{.}},
   comma={\BoldAndGreen{,}},
   split=yes]
```

123,456.78
1,234,567.89
1,358,024.67

Figure 8 Formatting (large) numbers with style and color.

5 The LUA end

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

```
\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.startcell() context("A[1] + A[2]") context.stopcell()
  context.stoprow()
context.stopspreadsheettable()
\stopluacode
```

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

```
\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")
    set("test",3,1,"A[1] + A[2]")
  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.bTR()
      context.bTD(options)
        context("%0.2f",A2)
      context.eTD()
    context.eTR()
    context.bTR()
      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 gets the numbers typeset nicely.

123,456.78	123456.78	123456.78	123456.78
1,234,567.89	1234567.89	1234567.89	1234567.89
1,358,024.67	1358024.67	1358024.67	1358024.67

Figure 9 Spreadsheets purely done as ConTEXt Lua Document.

6 Helper macros

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

```
\inspectspreadsheet[test]
\showspreadsheet   [test]
```

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

```
t={
 { 123456.78, 1234567.89, 1358024.67 },
}
```

Another helper function is `\doifelsespreadsheetcell`, You can use this one to check if a cell is set.

```
(1,1): \doifelsespreadsheetcell[test]{1}{1}{set}{unset}
```

```
(2,2): \doifelsespreadsheetcell[test]{2}{2}{set}{unset}
(9,9): \doifelsespreadsheetcell[test]{9}{9}{set}{unset}
```

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.

Colofon

author Hans Hagen, PRAGMA ADE, Hasselt NL
version May 12, 2016
website www.pragma-ade.nl – www.contextgarden.net