TeXcel? An unexpected use for TeX

Federico Garcia-De Castro

Abstract

I recently discovered the surprising fact that TeX seems to be more appropriate for keeping financial records, and especially preparing different kinds of reports (to funders, to the board, by season, by project, by calendar year), than the spreadsheets that I was using, and which had become highly convoluted as years of information accumulated. Here is a description of the task, the problems with the spreadsheets, and the incipient but already useful system I developed in TeX.

1 The problem

As the director of a contemporary chamber music company (`www.aliamusicapittsburgh.org`), I am in charge of designing and executing the budgets for individual projects and for whole concert seasons. And then reporting: to funders, board, government, throughout projects and after their completion.

In retrospect I see that it is this — the wide range of reports, each with its own format, budget lines and subdivisions — that reveals the inadequacy of any static-spreadsheet model for bookkeeping, and the counterintuitive fact that TeX's macro capabilities come in handy for financial tracking and reporting.

1.1 Ways of reporting

Different funders, and therefore different budget reports, focus on different things. Music foundations usually have budget formats that include items that are specific to concert production — things like performance licenses, that in more general purpose budgets go under something like "fees and dues" — while in typical art grant reports a concert's lighting design (a relatively straightforward expense) must be split into, for example, "equipment rental" and "other program professionals".

On the other hand, reports also vary by time scale and scope. Season-wide reports, for example, lump together all season ticket sales, while in project-specific reports for a concert's funder these need to be reported by project. On the other hand, the fees for online ticket sales, which are in principle project-specific expenses, in year-long budgets may better fall under "banking fees".

And so on: the assignment of each transaction to a particular budget line depends on the latter's purpose and format. In general, the only way of implementing this is to record all transactions in a report-independent database, from which each report gathers transactions into the appropriate lines, according to its own format and needs.

1.2 The spreadsheet model

I've long had such a database for Alia Musica, in the form of a spreadsheet:

	Season	Prj1	Prj2	…	Prj3
Donations					
Ticket sales					
⋮					
Grants					
Donations					
⋮					
Performers					
Guests					
⋮					
Ads					
Postcards					
⋮					

…and so on (although much more detailed). Columns specify information relevant to each project (or general season transactions in special season columns), while the rows were arranged following our most common report budget formats. Extra columns were used to gather season totals, for season-wide reports. In fact, since tax returns go by calendar year, which is different from the season year, these 'total columns' come in two flavors (one for each season, one for each calendar year); this also meant needing to have, for each season, two 'general season transaction' columns — one for the part of the season year that fell in one calendar year (September–December), the other one for the other one (January–August). In fact, this detail is just one of the many complications in the spreadsheet model. All of which eventually led me to look for an alternative.

Alia Musica's spreadsheet had been built and in use since around 2010, with ad hoc adjustments here and there as new needs emerged. But in the meantime the organization has changed and has grown significantly. In 2015–16, with two major productions (check out in particular the PITTSBURGH FESTIVAL OF NEW MUSIC, `www.pghnewmusic.com`), the spreadsheet had become clearly obsolete — now, for example, overlaps happened not only among seasons and calendar years, but among the productions themselves — some parts of one being part of another, and so on.

Toward the end of the season and of all of those productions — i.e., at reporting time — I decided to update the system, and started re-designing an array of spreadsheets, more in tune with current needs.

I did not get far: the deficiencies of the spreadsheet model, as I found out, are structural, and not due to a particular design or implementation.

1.3 Deficiencies of the spreadsheet model

Dimensions: A spreadsheet is basically a table in two dimensions. At best, you can use a couple of tricks and count them as two additional pseudo-dimensions:

- The user can attach 'notes' to each cell. In these notes, a total can be split into several components. For instance, the total ticket revenue of a concert can be input into the cell, with a note saying how much was door revenue and how much was online sales.
- What a cell shows on the spreadsheet is the result of the formula contained in the cell. You can make use of this to record some extra information: "`$420`" can be input, say, as "`=0+320+(40+40)+20`". At Alia Musica we used this trick to locate transactions by month: the above would mean "$0 in January; $320 in February; two transactions for $40 each in March; and $20 in April".

Of course these tricks do not provide for real extra dimensions: they afford extra information, but the processing program has no access to it — e.g., reports have no way eventually to take only a subset of the components of the cell's total. The information can be recorded, but its actual use requires user intervention.

User-time decisions: Transactions are input into the spreadsheet at different times (as they happen, or, more systematically, at month-end). That entails the ever-present risk of inconsistencies: where did we put parking expenses — sometimes as meeting expenses, sometimes as travel expenses? What did we decide about the guest's payment, did we put all of it as "guest performer", or did we split it into "guest performer", "lecture honoraries", "per diem", or who knows what else?

Completeness and feedback: Assume, however, that the inconveniences above can somehow be worked around: that the information is all consistently and clearly input into the big repository. Now the reports "simply" have to gather the relevant cells from here and there, according to their design.

An inescapable deficiency of the spreadsheet model shows up at this point: there is no mechanism to ensure that *a*) individual cells are not counted wrongly in two or more report lines, for which they might both be relevant; or *b*) that all relevant transactions for a particular line are included — maybe there was one obscure one (an extra parking expense, an extra bank transfer fee) that doesn't come to mind when manually gathering transactions into the report.

1.4 A necessary component: tags

Such reflection on the deficiencies of a spreadsheet points to one necessary component of any satisfactory system: assigning tags to transactions. A complete description of a transaction should be enough for the system to be able to pull it into whatever budget line each report needs. This in effect implements an open number of dimensions, and it even works toward the problem of user-time decisions: nothing prevents a report to gather tags for both "Fall 2015" and "Fall 15", relieving the user from having to remember a rigid list of possible tags.

Tags are implemented in financial tracking programs. But there are still problems: using tags typically involves dialog boxes, saving buttons, scrolling through lists... And in any case we still have the main problem of a spreadsheet model: nothing checks for completeness or duplications when at a later time the transactions are gathered into reports.

2 TEXcel

After realizing that the problem with my spreadsheet was not my particular spreadsheet, but the spreadsheet model itself, I came to wonder, almost as an afterthought, whether this was a task for TEX. The intuition was strong that there would be many problems, but that in principle something like this would be workable:

```
\deposit: 45.50 (Ticket sales, Festival
   subscription, Spring 2016)
\deposit: 8320 (Foundation Grant, T.W. Dunns
   Char. Fund, Fall 15)
...
\expense: 400 (Performer, Spring 16, Festival)
\expense: 19.80 (Stamps, Office expense,
   Mailing)
\expense: 1.59 (Facebook, Online advertising,
   Fall 2015)
\expense: 950 (Booklet printing, Festival)
...
```

With such a database (in a database "document"), reports could then be requested (in different documents) through something like this for a season-wide report:

```
\begin{expenseline}{Marketing}
  \include{Poster distribution}
  \include{Poster printing}
  \include{Flyer printing}
  \include{Advertising}
  \include{Online advertising}
  \include{Booklet printing}
  \include{Booklet shipping}
  \include{Website}
  \include{Project website}
  \include{Postcards}
\end{expenseline}
...
```

Or for a project report:

```
\begin{expenseline}{Spring 2016}
  \include{Spring 2016}
  \include{Spring 16}
  \include{Sp 2016}
  \include{Sp 16}
\end{expenseline}
...
```

Indeed, this is the backbone of a system I have developed for financial tracking in TeX.

2.1 The basic `\deposit` and `\expense`

So, `\deposit` and `\expense` are at the base of the whole system. They do not really 'mean' anything by default: what exactly TeX does when it finds them depends on the task at hand, as explained in a moment. This flexibility, and in general TeX's ability to define anything as anything, is a key reason why TeX turns out to be an appropriate environment for financial tracking.

Thus, `\deposit` (`\expense` is fully analogous) is defined, at bottom, as follows:

```
\long\def\deposit: #1 (#2){%
    \ifreporting
        \ifnum\yearindex=\z@
            \@deposit: #1 (#2)\relax
        \fi
    \else
        \@addtoacct{\acct}{#1}%
    \fi
    }
```

It's a simple fork: if we are compiling a report (gathering transactions from the repository into the appropriate budget lines of a report), we'll do one thing (`\@deposit`), and we'll need the comma-separated list of tags that comes as `#2`. Otherwise, in the database document, the transaction is merely being recorded, and we'll just update the corresponding account's balance through `\@addtoacc`, for which the tags are unimportant and we focus only on the amount (`#1`).[1]

[1] Now looking at the definition, I can't see a reason for the immediate handling of the arguments; `\deposit` could simply let `\@deposit` and `\@addtoact` pick them up later. This (and the `\long`, by the way) must be a residue from some initial try or a different basic model. The wonders of organic growth.

`\@addtoact` has an extra argument, passed on to it by `\acct`, as seen above. This is because the transactions are entered within one of three LaTeX environments, one for each of Alia Musica's accounts: checking (in which case `\acct` is defined as "`chk`"); money market ("`mmk`"); and PayPal ("`ppl`").

In fact, the latter illustrates another use of the flexibility of `\deposit`. In PayPal transactions, every deposit has a fee. Accordingly, the `paypal` environment in TeXcel redefines `\deposit` to take care of both the revenue amount and the associated fee. For example, a ticket sale could be:

```
\deposit: 15-.62 (Ticket sales, Spring 16)
```

Within the `paypal` environment, TeXcel then knows to add \$15 and subtract 62 cents (and, furthermore, it knows that the \$15 is 'ticket sales' and the −.62 is 'bank fees').

2.2 Convenience bundling macros

Transactions like ticket sales are (hopefully) very frequent, and there's no point in requiring the corresponding tag from the user. Ticket sales also come in groups (more than one at a time). So, TeXcel has a further macro, `\tickets` (a straightforward front-end for `\deposit`), that takes care of it:

```
\tickets: 30-1.17, 60-2.04 (Fall 15)
```

There are similar 'shorthand' macros throughout the system, including `\donation` and recurring expenses like `\stamps`, `\servicecharge`, and so on, all of which built on top of the basic `\deposit` and `\expense`.

One such macro is `\check`. TeXcel provides for an independent database of checks, where checks are defined in full detail. The transaction database can then call checks up through

```
\checks1131-1136,1138,1140,1141,1143-1150.
\checks3153,3177-3193,3195-3199.
\check3176
```

(notice the flexibility in syntax!) This saves a lot of time when entering the transactions, and in addition is extremely useful to keep track of checks that have been issued but not cashed yet.

In the spreadsheet model in use until now, uncashed checks have been a source of nearly-intractable discrepancies between projected budgets, reports, and account balances. With TeXcel this is no longer a problem: `\check`, `\checks`, etc., can themselves be redefined for any purpose (just as `\deposit` and `\expense`) — notably, to run through the checks and make a list of pending liabilities.

2.3 Time-based reporting

A further utility worth noting: as mentioned above, reports can go by season (September–August) or by calendar year. This was a tough nut to crack, not least because sometimes a season has revenue and expenses that happen actually before the beginning of the season (a grant that's awarded in July, say), or after its end (a check that we only get in September). This is the reason for `\yearindex` above in the definition of `\deposit`. The internal mechanism will be detailed below. At user entry time, any transaction can be recorded as the argument of `\late` or `\early` — TEXcel will know what to do with it.

This means that TEXcel keeps track of the time transactions occur. In fact, the user actually enters transactions within new LaTeX environments for the months — `\begin{January}` and so on. (So, each month has three nested environments, for each of the accounts.)

This brings a major benefit over the spreadsheet model: the time dimension is preserved. Back in the spreadsheet, with transactions assigned vertically by project/season and horizontally by budget line, there was no way to keep track of exact account balances by date. When discrepancies occurred at reporting time, locating them required going over all the statements manually, one by one, until something popped up. In contrast, with the month environments in TEXcel, the system is able to report account balances at the end of each month — catching a discrepancy is now trivial.

2.4 Automatic consistency checking

Beyond all the above features and benefits, probably the most important utility offered by the new system is the automatic check for duplications and omissions. When compiling a report, the user instructs TEXcel to gather transactions by tag into different budget lines. To repeat an example from above:

```
\begin{expenseline}{Marketing}
    \include{Poster distribution}
    \include{Poster printing}
    \include{Flyer printing}
    \include{Advertising}
    \include{Online advertising}
    \include{Booklet printing}
    \include{Booklet shipping}
    \include{Website}
    \include{Project website}
    \include{Postcards}
\end{expenseline}
```

Other `expenseline` environments include, for example, performer honoraria (tags like 'performer', 'performers', 'conductor', 'soloist', etc.), operation expenses ('insurance', 'office supplies', etc.), and so on.

After the series of user-requested expense (or revenue) lines, TEXcel automatically compiles a further list, "Unassigned Transactions", of those which were not included (probably due to the user's unintentional omission) in any of the environments.

On the other hand, the system keeps track of which transactions have already been assigned, and if a later budget line matches an already used tag, it will warn that the transaction number x on month y was already counted in section z.

With these two features, the system provides reliable consistency and completeness checks, a major advantage over any static tracking system.

2.5 Programming tricks

It is still the case that TEX is not exactly the most adequate programming environment for either database or spreadsheet handling. There are some structural limitations to what TEX can do — no easy alphabetization, for example, and a certain clumsiness in holding information for later use, which all but discourages trying to present the report in table form. (TEXcel makes LaTeX sections for each budget line, an `itemize` environment for each tag included, followed by a total of the transaction amounts for each section.)

Even those features that are implemented required a bit of hacking. Here are some of the most interesting tricks.

Arithmetic We're dealing with dollars and cents, but TEX's arithmetic is limited to integers... At first I used the trick of dealing with dollar amounts in "dimen" registers — TEX is good at arithmetic with lengths and dimensions. It was a little funny that all amounts reported by TEXcel would have "`pt`" after them, but one could live with that.

I thought I was so clever. The problem, of course, is TEX's upper limit — a sum like 35000 is a longer dimension than TEX can handle. I tried to salvage this by working in terms of 'scaled points' (the true internal unit that TEX uses, much smaller physically and therefore with a much much higher upper limit). But TEX still presents dimensions translated into normal `pt` units — so that the numbers reported would be meaningless. (And if you simply try to reconvert `pt` into `sp` right before typesetting, you again exceed TEX's numeric limit.)

There was no other way than to implement decimals "manually". I checked a couple of existing packages, but in general they provide for much more complicated functions (trigonometry, floating point, etc.), not exactly what I needed.

So, in TEXcel all arithmetic is done through two streams of integers: one for the dollars, one for the cents. Then there is a function that converts that into the usual decimal point presentation.

In this context the decimal period has no mathematical meaning. As a result, TEX would read `12.4` as 4 cents, not 40. Very annoying, and very annoying to fix! But it had to be done: leaving the task to the user (possibly months from now) would invite potentially untraceable mistakes.

Feedback mechanism When a report is compiled (selecting out transactions by tag), what happens roughly — very roughly, almost entirely notionally — is that TEX goes through the list of transactions, and creates new commands for each tag. That way, when a tag is requested by the user, it is an active command that 'executes' the corresponding transaction ... but this command also redefines itself, so that when the transaction is called for a second time, it now does something else (warn of the duplication).

Months and years I was amazed at how quickly this could get extremely confusing when I was trying to implement it. The problem is that TEX needs to know, according to what kind of report we're doing (by season or by calendar year) which months to include. You would say it's a matter of adding an offset to the month counter (so that September is 1 when going by season, but 9 when going by calendar year), and so did I. This basic offset is in fact somewhere in the code.

But that's not enough, because transactions outside of the requested year are still relevant: sometimes we have a grant for season n that was actually awarded and cashed toward the end of season $n-1$; sometimes liabilities and receivables overflow to season $n+1$. On the one hand, these outer transactions are still necessary for the full picture of a season; and on the other, they should be kept *out* of the reports for the seasons where they actually took place. (Calendar-year reports do not need this nuance, since they are basically balance-sheet reconciliations for the IRS.)

So in the end the model uses a 'year index:' 0 for the current (requested) season, -1 for the previous one, 1 for the following one. The counter is updated by the month environments: `\begin{September}` (the first month in a season) steps `\yearindex` by 1. Then the program knows what to do.

2.6 Future

The system is complete in the sense that any user (say, an intern) can use it. It is also very flexible — key functions like `\deposit`, `\expense`, and `\check` are essentially black boxes that can be redefined for any future needs that might arise.

But these future needs are not implemented. That is to say, it is only Alia Musica's needs that are implemented right now. In that sense the program is incomplete, and only a model for what could be done for more general purposes.

Were this to be done for a public release, then it would probably be a good idea to translate the code into LuaTEX — Frank Mittelbach's suggestion — so that we get a true general-purpose programming language. Desirable features would include reporting by tables, alphabetization and other sorting capabilities, and, less cosmetically, a more powerful engine to handle the tags. Right now the program compares tags, and when it finds a match it assigns a transaction right away; future matches of the same transaction are discarded (with a warning). It would be great if the user could request more complicated conditions: "include this tag but only if this other one is not there"; or "only include transactions with the indicated tags if in addition they have tag x".

For now, it was a lot of fun, not that hard, and in any case very surprising, to work on implementing Alia Musica's financial needs through TEX. The resulting system is enormously, structurally, superior to any spreadsheet.

◇ Federico Garcia-De Castro
Artistic Director,
Alia Musica Pittsburgh
federook (at) gmail dot com
http://www.garciadecastro.net