DocCenter — TEXing 11 million documents a year

Joachim Schrod

1 TEX goes banking

Our company's product DocCenter is used to manage the standardized, corporate identity (CI) conformant, written communication of a company. Our reference customer is 1822direkt,[1] a German online bank, which uses it to handle almost all written communication with its customers. Some communication is done via email or displayed as HTML messages in an online postbox, but most is created via LATEX and then printed or provided as PDF.

This experience report presents input methods, output formats, and delivery channels used by our customer. It also reports on related challenges and TEXnical solutions chosen.

But first things first: What is special about customer related documents of an online bank?

- They are small, mostly one or two pages.
- They are simple documents that don't need lots of markup, just some item lists, some fontifying, and some simple tables are sufficient.
- Few images are used, and those are not individually created for an individual document.
- All documents are obliged to use the company's corporate identity formatting rules; it should be hard for an author to break them.
- There are no reports, no books, no highly structured documents, no math — any use case where (LA)TEX is usually the first choice and what TEX experience reports are usually about is missing here.

So, small documents — but there are many of them; last year DocCenter was used to create 11 million of them! Quite a lot of the documents are letters:

- either with standardized content and a few fill-in variable parts
- or completely individual content
- or built from pre-defined text fragments

Even though the content itself is simple, it has to be enriched by corporate identity requirements, demands for standardized greetings and closings, automated addition of signatures of writer and department heads, footnotes with ever-changing advertisements for special opportunities, logos of current ratings from financial journals, etc. This enrichment is standardized and delivered by appropriate LATEX document classes; the content authors don't need to worry about it.

[1] 1822direkt is the online sales company of Frankfurter Sparkasse 1822.

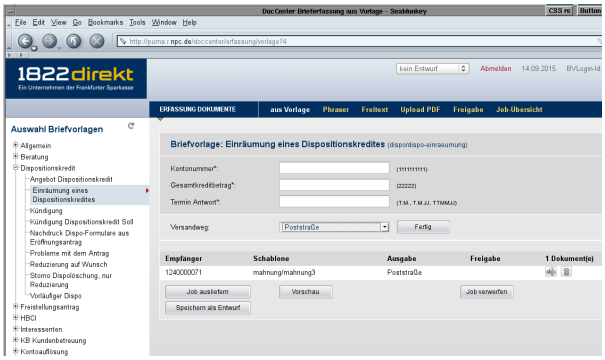

Figure 1: Standard letter input form (cropped).

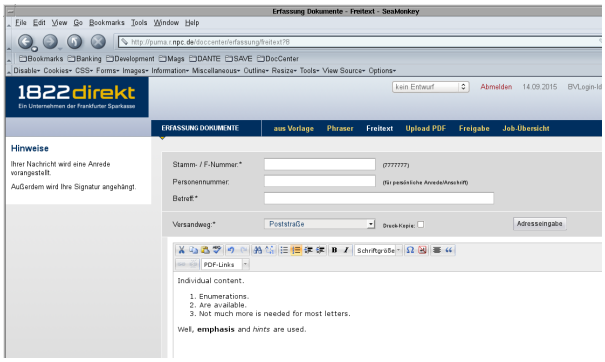

Figure 2: Individual letter input form (cropped).

In fact, the letter idiom is also used for account statements, credit card statements, and share notes. In fact, these are the majority of documents produced. These kinds of documents also often have special requirements concerning delivery, users may need to acknowledge the receipt, a printed version may be needed to be sent by post if no such acknowledgement arrives in due time.

Some other documents add additional requirements. A good example are PIN/TAN letters that must be kept private: they are printed on special paper only on certain in-house printers, they must be archived without the "secret part", etc.

2 DocCenter

As the introduction mentioned, we have two kinds of documents that are created with DocCenter: (1) letters that are created by staff members, and (2) automatically created documents.

2.1 Document creation by humans

DocCenter provides a web-based intranet application to create documents by staff members. Figures 1 and 2 show the basic interface.

Standardized letters are created by application forms that request only the variable parts. No letter content is shown during input of these *document parameters*, and it's not needed either: Staffers write those letters by the dozens or even hundreds; they know the letter's content by heart. Input must be quick and must be checked as closely as possible.

E.g., the author need not input a customer's name, or address, or account details — they are all available to be inserted into the letter content after the customer's account number has been input in the form. The actual letter content comes from so-called *templates*. We postpone describing their content creation to section 2.4, as the same templates are used for automatic document creation as well.

A preview is available to check final output — but most often it is not used. Users rely on proper document creation; they need to get the letters out in the most efficient way.

Individual letters may also be created, with boilerplate text parts available to ease that duty. For that task an HTML editor, TinyMCE, is integrated. That editor is configured to provide only formatting capabilities that are CI-conformant. The editor's XHTML result is then converted to LaTeX and output over the chosen channel.

Users of this "frontend" to DocCenter work on a few small documents at a time, and expect very fast system reaction time, both for preview and output creation. This speed aspect is called *latency*: Document generation and formatting on the server must happen fast, with results delivered very quickly.

2.2 Automatic document creation

DocCenter provides an HTTP interface to create jobs with documents. Most prominently, that interface is used for automated document mass production. Those documents are still small, but tens of thousands may be requested in one job.

Such a job's document requests don't include the content to be output. They name a document template which determines the content or how the content is to be generated. Document parameters in the request configure output or content creation. Other request attributes establish the output channel to be used, e.g., printer, online PDF delivery, transfer to a print shop, or others.

Since this interface is not used by humans, quick reaction time is not important. On the other hand, it is important that jobs are finished in some predetermined given time. E.g., when account or credit card statements are created for all customers once a month, there are service level agreements to fulfill — these statements must be delivered to the customer by a specified date.

This is a different kind of performance demand than the requirement of low latency for interactive usage: Commonly called *throughput*, it is concerned with overall processing time for a given set of documents, not about the processing time for a single document.

2.3 Basic architecture

DocCenter uses basically a 4-step lifecycle of document processing that provides a maximum of stability and control:

generate ⇒ format ⇒ output ⇒ archive

Document variants are used to provide needed specialization for one or several of those phases. The most important are variations of document generation. They are implemented as *plugin classes* that may be added to the system as needed. Thus, new demands for specific content generation or formatting can be satisfied easily by realizing and deploying a new plugin, without changing the base system.

Some illustrations:

Account statements use a specific generation step that fetches account data from the database; a document's LaTeX markup is created completely by the application.

Standardized letters generate a LaTeX document file that merely contains parameter declarations and then inputs a LaTeX template file that may use the parameters. (Those parameters are basically macro declarations.)

Individual letters transform XHTML content to LaTeX markup and content during generation; XSLT is used for that.

PIN letters don't archive the actual PIN, just the letter text. Thus bank personnel later can see when a PIN letter was sent to which person, but not the actual secret initial PIN data.

Other document variants provide further specializations that are even more company-specific. Realization of such document variants is the primary method for adapting DocCenter to different customers' demands.

2.4 LaTeX templates

Standardized letters are by far the most typical document that are created by DocCenter — not by document numbers, account statements dwarf that — but by variety that is handled smoothly with our application.

Within a bank, new content for a letter may not be easily created by a staff member on a personal

whim. Professional content (correct terms, factual correctness) is provided by one organizational unit, another checks that CI-style conformant phrases are used, while wording is checked by a third one. Each new letter and each change must go through the required protocols and have approvals recorded, to be available for inspection by auditors.

For that reason, actual document content is created and managed by an editorial team that organizes the process and coordinates the different groups that are involved. This editorial team actually creates LaTeX and XML files with meta-information; no interactive frontend is involved that hides that technology.

An advantage that we did not hesitate to exploit is the simple document structure. TeX, and subsequently LaTeX, assigns special functionality to many characters, be it `$` to introduce math mode, `%` to start a comment, or other characters used to make markup of complex documents more readable. Well, we don't have complex documents, we've got no math to typeset, and comments in documents are an alien concept that's hard to communicate to staff members anyhow. On the other hand, not being able to simply type `$` or `%` in a letter from a bank to get the respective characters in the output — that's a difficult restriction for this kind of user.

For these reasons, we reconfigured LaTeX a bit and added application-specific markup that supports creating simple documents in a way that can easily be learned by non-TeXies while still being LaTeX:

- Very simple LaTeX
- No math
- A minimum of special characters: just `\ { }`
- In particular, $ and % are normal characters, lest they create havoc in our banking context
- Insert document parameters, with optional formatting
- Optional text, controlled by parameters
- Some additional special environments; e.g., creating a pre-filled answer letter to the bank that is appended to the actual letter.

From the usual TeX point of view, these would hinder creating reports or longer documents. For our target use case, creating letters, such a reasonable subset of LaTeX functionality enables users to create new standard letter templates within hours, without a steep learning curve. (The few hours are actually spent by learning what metadata is needed and how to express it, not by creating real content.)

3 Challenges

Document creation in the context described above comes with some unusual challenges. Solutions are readily available in the TeX world, if we look beyond common knowledge of how a contemporary TeX system is used.

3.1 Output variations without reformatting

DocCenter has to be able to output formatted results via different *output channels*:

- PDF, to be delivered online to the customer
- Printed in-house, on both PostScript and PCL printers
- Print files for external print shop

Output is *not* the same for these output channels:

- Online PDF usually needs an embedded letterhead, provided as an image.
- Print output uses letterhead paper; thus a letterhead image must not be embedded.
- Documents use different types of paper; e.g., first page on letterhead paper, second page on white paper, maybe third page again on letterhead paper.
- Printer-specific tray control: Each printer may have paper types in different trays; using the right paper type as per requirement above must thus be configured and realized *per printer*.
- Printer calibration: Precise output positioning is important for letters, address fields must fit exactly into window envelopes.
  - Each laser printer feeds paper a bit differently, output doesn't end up on the page where it should be.
  - Experience shows that different printers may stray up to 5mm (0.2in) in all directions; while positioning errors for different sheets in one printer is a magnitude lower.
  - Therefore we need a *per printer configuration* (again) that offsets output on the page.
- Folding machine control: Output on some special printers is fed immediately to a folding machine that controls the completion of all of a letter's pages, folds them, and places them into an envelope. This is controlled by bar codes at the paper's left edge — these bar codes only have to be inserted when printed on these printers, not for any other output channel.
- Print shops need associated metadata (some want them embedded invisibly into PDF files) for paper type control.
- Some print shops always print duplex; extra empty pages may have to be inserted for them.

- Preview output needs watermarks (grey "draft" in the background), to make sure that the four-eyes control workflow cannot be easily circumvented.
- Output of archived documents again needs a watermark, to distinguish original documents sent to the customer from internally produced copies.
- Some letters are archived incompletely, e.g., secret PIN numbers must not be stored. One should still be able to have a partial view of the archived document, without that secret part.

But the biggest issue of all is the requirement that output of a document may have to be repeated on a different output channel after months or even years — taking the peculiar requirements above into account — *without reformatting the document*. Reformatting a document after several years always has the risk that the output may be different, owing to changed LaTeX packages or other internal macro changes. That must not happen; precisely the same document has to be reproduced.

Therefore, to separate format from output phase in DocCenter's document life cycle doesn't just mean that a formatted result, e.g., a PDF file, is sent to some output device. Instead, the output phase transforms the formatting result and implements the output channel specific requirements.

The current standard in TeX world for output format is PDF. Almost all current publications and presentations at conferences take that for granted. While we could realize all this output phase manipulation by transforming TeX-produced PDF files, the overall TeX universe has an older technology available that's better suited for our purpose:

DVI files with `\special`s

TeX specials are used in the DVI format result to declare the need for duplex/simplex, letterheads, paper types, watermarks, etc. DVI drivers interpret them and produce adequate printer-specific output.

As an example, printer-specific tray control is as easy as setting up directories with include files with standardized names; these include files contain printer control commands to access trays for the correct paper type. Inclusion of these files is triggered by appropriate `\special`s in the document.

Printer-specific output placement is even easier to realize: Every DVI driver has options for offset control, a configuration file per printer has values for that option.

Last, but not least, using DVI greatly lowers cost for our document archive. A typical letter with roughly 40 KB in PDF format needs only 2 KB in DVI format. The disk space requirement is thus reduced from 500 GB per year to 25 GB per year. This doesn't sound much in terms of today's USB storage prices where you get multiple TBs for cheap — but you can't use such inexpensive storage easily in a bank's data center. There it still matters if a 10-year storage archive needs 10 TB or 1 TB.

3.2 Latency improvement

Before DocCenter was deployed, a predecessor system was used that was also based on LaTeX. With a rather naive implementation, that needed 1.5 seconds to process a document. Adding the communication latency, delivering a preview from server to user needed up to 3–4 seconds, clearly far too long.

Root cause analysis showed us the reasons for that behavior: Most of the time was spent in boilerplate processing: reading and processing LaTeX class and package files, font configurations, etc. Creating a letter with two paragraphs of text needed processing more than a dozen macro files. Actual time for formatting the document's content was minimal. (SSD disk caches might have helped, but were not readily available in the clustered server architecture that is in use at the customer.) Additional time was spent by processing each document twice, as is common document production practice in the TeX world.

Well, that problem was easily tackled with standard TeX techniques from the early ages: We don't have document-specific packages, and our documents are not one-off creations. Instead, all of our 11 million documents use the same set of packages, maybe with some small variation in feature usage. So we created a TeX format file that has LaTeX and all used packages and font definitions preloaded. The format also redefines `\documentclass` and other preamble control sequences to do nothing — class and package files are already loaded, after all.

A second measure was to stop processing documents twice. Analysis showed that we don't need any of LaTeX's features, like cross references, that demand multiple formatting runs. Processing each document once is sufficient.

Reading a format file is *very* fast in TeX, being the equivalent of a memory dump. Document formatting time was reduced from 1.5 seconds per documents to 0.06 seconds per document; a 25 x improvement by using our specific FMT file and doing only one run.

3.3 Throughput improvement

Creating account statements was another challenge. The predecessor system used a tabular layout, as is common with such statements, implemented via

LaTeX's longtable package. Each of the hundreds of thousands of statements that had to be produced was created separately, with a new database connection and queries, creation of a new LaTeX file, running LaTeX twice (owing to usage of longtable), and creation of output files for the customer.

To achieve a service level of maximum processing time of 24 hours, the creation process was spread over 10 systems where it needed a total time of 22 hours. At least half of that time could be attributed to non-optimal usage of TeX technology.

Our first observation was that "looks like a table" doesn't mean that it is a longtable or even a tabular environment in LaTeX markup parlance. Bank statement layout is not at all flexible; column widths are preset and don't change with content. There are some running heads at page breaks, but they don't demand the full power of complex table formatting capabilities.

Instead, we turn towards the most basic formatting capability TeX has: `\hbox` and `\vbox`. Nothing is faster in TeX formatting than using these primitives. A booking entry in the statement is not a table line with columns, it's an `\hbox` that contains `\vbox`es with fixed widths. Voilà, blindingly fast processing by TeX is the result.

Our second observation was, again, the overhead of boilerplate processing for all these document files, as mentioned in the previous section. For this use case, we optimized it even further, beyond using our own FMT file. We generate markup and content for ca. 50,000 documents in one run and feed them directly to LaTeX, without any intervening process. LaTeX then dutifully produces a DVI file with 200,000–250,000 pages.

The choice of DVI files to represent our formatted result comes in quite handy now. In a DVI file, pages are linked from the back to the start. The first 10 TeX counters are stored at such a page start. Our macros store a document ID in one of these counters and so we can detect the start of a new document while jumping from one page to the next. Splitting the single DVI file into 50,000 smaller ones is thus a matter of less than a second, mostly dominated by I/O times on the networked storage in use in such clustered environments.

An interesting point from a TeX point of view is a further optimization that made splitting much easier. A hairy detail of DVI file splitting is the declaration of fonts: they appear at first use and at the end of the file. Rather complicated logic is needed for an arbitrary split algorithm to handle that properly. However, since we are in such a restricted use case scenario, we can add a first page that does nothing but load all fonts needed. Our DVI-splitting algorithm collects all font definitions from that first page, to be output to every produced DVI file at the start, and then may append DVI pages from the respective document without having to worry about appearance of font definitions at all. This makes the split code size really small, robust, and easy to maintain — much easier than comparable code for splitting a large PDF file.

Using this production approach has proved a full success: We achieved a performance improvement of a factor of 100. All statements can be created on one system within 2 hours.

4 Conclusion

Using TeX as the centerpiece for document creation in DocCenter was a full success. We have a robust application that's purring on without production problems in that area. Traditional TeX toolbox solutions like DVI files, specials, FMT files and the like are still immensely useful for the diversified requirements of document generation, even in today's communication demands. It will still work in 10 years, of great importance for a bank. What other typesetting system can say the same?

Still, there were some minor hurdles that we had to overcome:

- A standardized and stable API for TeX processing is missing. While it's accepted that TeX is hard to handle by humans, it's also hard to control by an application.
- Most important, batch mode and error message handling are not perfect for monitoring.
- On the organizational side, there is no staff easily available with sufficient knowledge of (La)TeX technology — personnel for support tasks is even harder to find than for development.
- Customer-specified special formatting for individual documents and one-off-changes are difficult to achieve on-site by the customer's editorial group, without our involvement.
- Quality of DVI drivers is worse than 15 years ago. (`dvips` is the exception.)

But these issues shouldn't stop you from using TeX for similar tasks. Other typesetting systems will come with their own problems, and more of it — we have the scars to prove it, but that's another story.

◇ Joachim Schrod
Net & Publication Consultance GmbH
jschrod (at) acm dot org