context musings

hans hagen

Content

Introduction

This is a collection of articles and wrap-ups that don't suit in other manuals or collections. Some are published, some meant as draft for a presentation.

The "Children of TEX" article is the framework for a presentation at BachoTEX 2017 in Poland, and covers the main theme of the conference. In the aftermath of that conference I wrote "Advertising TEX" and later "Why use TEX?". The 2018 BachoTEX conference theme is explored in "What's to stay, what's to go". After a short discussion on the CONTEXT mailing list about stability (at the moment that MKII had been frozen for more than a decade but is still used without problems) I wrote "Stability".

Many of the thoughts in these articles are influenced by discussions with my colleagues Ton Otten and Kees van Marle, users and developers. Operating in a similar arena, they provide me the reflection needed to sort out my thoughts on these matters.

The order in this document is not chronological. In the meantime we also put some development related stories in this collection, just because they have to fit in somewhere.

Not all musings are checked and copy-edited so let me know if there are errors and typos in them.

Hans Hagen
Hasselt NL
2017–2023+

Children of TEX 1

1.1 The theme

Nearly always TEX conferences carry a theme. As there have been many conferences the organizers have run out of themes involving fonts, macros and typesetting and are now cooking up more fuzzy ones. Take the BachoTUG 2017 theme:

Premises The starting point, what we have, what do we use, what has been achieved?
Predilections How do we act now, how do we want to act, what is important to us and what do we miss?
Predictions What is the future of TEX, what we'd like to achieve and can we influence it?

My first impression with these three P words was: what do they mean? Followed by the thought: this is no longer a place to take kids to. But the Internet gives access to the Cambridge Dictionary, so instead of running to the dusty meter of dictionaries somewhere else in my place, I made sure that I googled the most recent definitions:

premise an idea or theory on which a statement or action is based
predilection if someone has a predilection for something, they like it a lot
prediction a statement about what you think will happen in the future

I won't try to relate these two sets of definitions but several words stand out in the second set: idea, theory, action, like, statement and future. Now, as a preparation for the usual sobering thoughts that Jerzy, Volker and I have when we staring into a BachoTEX campfire I decided to wrap up some ideas around these themes and words. The books that I will mention are just a selection of what you can find distributed around my place. This is not some systematic research but just the result of a few weeks making a couple of notes while pondering about this conference.

1.2 Introduction

One cannot write the amount of TEX macros that I've written without also liking books. If you look at my bookshelves the topics are somewhat spread over the possible spectrum of topics: history, biology, astronomy, paleontology, general science but surprisingly little math. There are a bunch of typography-related books but only some have been read: it's the visuals that matter most and as there are no real developments I haven't bought new ones in over a decade, although I do buy books that look nice for our office display but the content should be interesting too. Of course I do have a couple of books about computer (related) science and technology but only a few are worth a second look. Sometimes I bought computer books expecting to use them (in

history futurology science fiction

Figure 1.1

some project) but I must admit that most have not been read and many will soon end up in the paper bin (some already went that way). I'll make an exception for Knuth, Wirth and a few other fundamental ones that I (want to) read. And, I need to catch up on deep learning, so that might need a book.

My colleagues and I have many discussions, especially about what we read, and after a few decades one starts seeing patterns. Therefore the last few years it was a pleasant surprise for me to run into books and lectures that nicely summarize what one has noticed and discussed in a consistent way. My memory is not that good, but good enough to let some bells ring.

The first book that gave me this "finally a perfect summary of historic developments" feeling is "Sapiens" by Yuval Noah Harari. The author summarizes human history from a broad perspective where modern views on psychology, anthropology and technical developments are integrated. It's a follow up on a history writing trend started by Jared Diamond. The follow up "Homo Deus" looks ahead and is just as well written. It also integrates ideas from other fields, for instance those related to development of artificial intelligence (Dennett, Bostrom, etc.).

Another inspiration for this talk and article is the 50 hour lecture series on behavioral biology by Robert Sapolsky of Stanford University, brought to my attention by my nephew Bram who visited a few TEX conferences with me and who is now also forced to use TEX for assignments and reports. (How come self-published books used at universities often look so bad?)

The title of this talk is inspired by the book "Children of Time" by Adrian Tchaikovsky that I read recently. There are science fiction writers who focus on long term science and technology, such as some of Alastair Reynolds, while others follow up on recent development in all kind of sciences. One can recognize aspects of "Superintelligence" by Bostrom in Neal Asher's books, insights in psychology in the older Greg Bear books, while in the mentioned "Children of Time" (socio)biological insights dominate. The main thread in that book is the development of intelligence, social behaviour, language, script and cooperation in a species quite different from us: spiders. It definitely avoids the anthropocentric focus that we normally have.

So how does this relate to the themes of the BachoTEX conference? I will pick out some ways to approach them using ideas from the kind of resources mentioned above. I could probably go on and on for pages because once you start relating what you read and hear to this TEX ecosystem and community, there is no end. So, consider this a snapshot, that somehow relates to the themes:

premise Let's look at what the live sciences have to say about TEX and friends and let's hope that I don't offend the reader and the field.

predilection Let's figure out what brings us here to this place deeply hidden in the woods, a secret gathering of the TEX sect.

prediction Let's see if the brains present here can predict the future because after all, according to Dennett, that is what brains are for.

At school I was already intrigued by patterns in history: a cyclic, spiral and sinusoid social evolution instead of a pure linear sequence of events. It became my first typeset-by-typewriter document: Is history an exact science? Next I will use and abuse patterns and ideas to describe the TEX world, not wearing a layman's mathematical glasses, but more from the perspective of live sciences, where chaos dominates.

1.3 The larger picture

History of mankind can be roughly summarized as follows. For a really long time we were hunters but at some point (10K years ago) became farmers. As a result we could live in larger groups and still feed them. The growing complexity of society triggered rules and religion as instruments for stability and organization (I use the term religion in its broadest sense here). For quite a while cultures came and went, and climate changes are among the reasons.

After the industrial revolution new religions were invented (social, economic and national liberalism) and we're now getting dataism (search for Harari on youtube for a better summary). Some pretty great minds seem to agree that we're heading to a time when humans as we are will be outdated. Massive automation, interaction between the self and computer driven ecosystems, lack of jobs and purpose, messing around with our genome. Some countries and cultures still have to catch up on the industrial revolution, if they manage at all, and maybe we ourselves will be just as behind reality soon. Just ask yourself: did you manage to catch up? Is TEX a stone age tool or a revolutionary turning point?

A few decades ago a trip to BachoTEX took more than a day. Now you drive there in just over half a day. There was a time that it took weeks: preparation, changing horses, avoiding bad roads. Not only your own man-hours were involved. It became easier later (my first trip took only 24 hours) and recently it turned into a piece of cake: you don't pick up maps but start your device; you don't need a travel agent but use the Internet;

there are no border patrols, you can just drive on. (Okay, maybe some day soon border patrols at the Polish border show up again, just like road tax police in Germany, but that might be a temporary glitch.)

Life gets easier and jobs get lost. Taxi and truck drivers, travel agents, and cashiers become as obsolete as agricultural workers before. Next in line are doctors, lawyers, typesetters, printers, and all those who think they're safe. Well, how many people were needed 400 years ago to produce the proceedings of a conference like this in a few days' time span? Why read the introduction of a book or a review when you can just listen to the author's summary on the web? How many conferences still make proceedings (or go for videos instead), will we actually need editors and typesetters in the future? How much easier has it become to design a font, including variants? What stories can designers tell in the future when programs do the lot? The narrower your speciality is, the worse are your changes; hopefully the people present at this conference operate on a broader spectrum. It's a snapshot. I will show some book covers as reference but am aware that years ago or ahead the selection could have been different.

1.4 Words

Words (whatever they represent) found a perfect spot to survive: our minds. Then they made it from speech (and imagination) into writing: carved in stone, wood, lead. At some point they managed to travel over wires but no matter what happened, they are still around. Typesetting as visualization is also still surrounding us so that might give us a starting point for ensuring a future for TEX to work on, because TEX is all about words. There is a lot we don't see; imagine if our eyes had microscopic qualities. What if we could hear beyond 20KHz. Imagine we could see infrared. How is that with words. What tools, similar in impact as TEX, can evolve once we figure that out. What if we get access to the areas of our brain that hold information? We went from print to screen and TEX could cope with that. Can it cope with what comes next?

The first printing press replaced literal copying by hand. Later we got these linotype-like machines but apart from a few left, these are already thrown out of windows (as we saw in a movie a few BachoTEX's ago). Photo-typesetting has been replaced too and because a traditional centuries old printing press is a nice to see item, these probably ring more bells than that gray metal closed box typesetters. Organizers of TEX conferences love to bring the audience to old printing workshops and museums. At some point computers got used for typesetting and in that arena TEX found its place. These gray closed boxes are way less interesting than something mechanical that at least invites us to touch it. How excited can one be about a stack of TEX Live DVDs?

1.5 Remembering

Two times I visited the part of the science museum in London with young family members: distracted by constantly swiping their small powerful devices, they didn't have the least interest in the exhibited computer related items, let alone the fact that the

couch they were sitting on was a Cray mainframe. Later on, climbing on some old monument or an old cannon seemed more fun. So, in a few decades folks will still look at wooden printing presses but quickly walk through the part of an exhibition where the tools that we use are shown. We need to find ways to look interesting. But don't think we're unique: how many kids find graphical trend-setting games like Myst and Riven still interesting? On the other hand a couple of month ago a bunch of nieces and nephews had a lot of fun with an old Atari console running low-res bitmap games. Maybe there is hope for good old TEX.

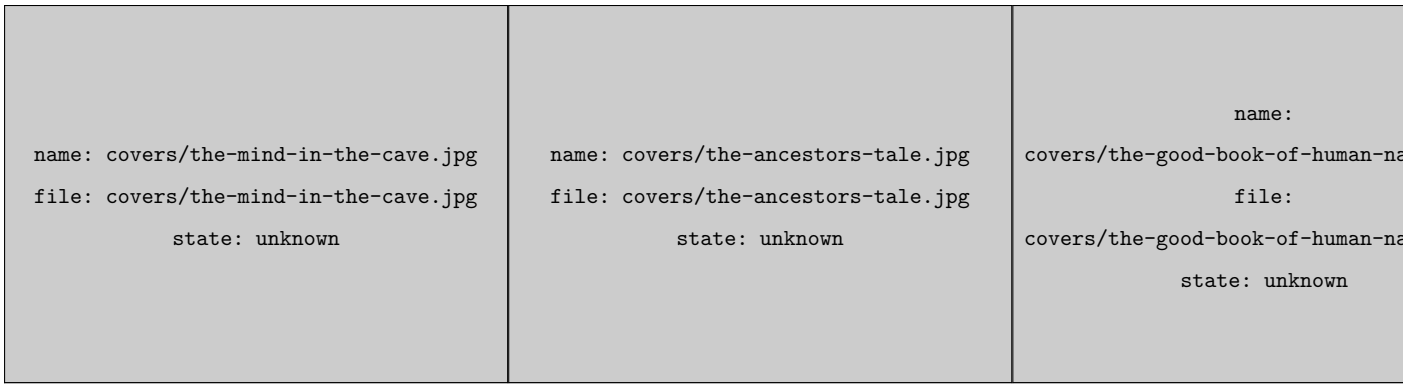
If indeed we're heading to a radically different society one can argue if this whole discussion makes sense. When the steam engine showed up, the metaphor for what went on in our heads was that technology, It's a popular example of speakers on this topic: "venting off steam". When electricity and radio came around metaphors like "being on the same wavelength" showed up. A few decades ago the computer replaced that model although in the meantime the model is more neurobiological: we're a hormone and neurotransmitter driven computer. We don't have memory the way computers do.

How relevant will page breaks, paragraph and line breaks be in the future? Just like "venting off steam" may make no sense to the youth, asking a typesetter to "give me a break" might not make much sense soon. However, when discussing automated typesetting the question "are we on the same page" still has relevance.

Typesetting with a computer might seem like the ultimate solution but it's actually rather dumb when we consider truly intelligent systems. On the large scale of history and developments what we do might get quite unnoticed. Say that mankind survives the next few hundred years one way or the other. Science fiction novels by Jack McDevitt have an interesting perspective of rather normal humans millennia ahead of us who look back on these times in the same way as we look back now. Nothing fundamental changed in the way we run society. Nearly nothing from the past is left over and apart from being ruled by AIs people still do sort of what they do now. TEX? What is that? Well, there once was this great computer scientist Knuth (in the remembered row of names like Aristotle —I just started reading "The Lagoon" by Armand Leroi— Newton, Einstein, his will show up) who had a group of followers that used a program that he seems to have written. And even that is unlikely to be remembered, unless maybe user groups manage to organize an archive and pass that on. Maybe the fact that TEX was one of the first large scale open source programs, of which someone can study the history, makes it a survivor. The first program that was properly documented in detail! But then we need to make sure that it gets known and persists.

1.6 Automation

In a recent interview Daniel Dennett explains that his view of the mind as a big neural network, one that can be simulated in software on silicon, is a bit too simplistic. He wonders if we shouldn't more tend to think of a network of (selfish) neurons that group together in tasks and then compete with each other, if only because they want to have something to do.



paleontology evolutionary biology anthropology

Figure 1.2

Maybe attempts to catch the creative mindset and working of a typesetter in algorithms is futile. What actually is great typography or good typesetting? Recently I took a look at my bookshelf wondering what to get rid of — better do that now than when I'm too old to carry the crap down (crap being defined as uninteresting content or bad looking). I was surprised about the on-the-average bad quality of the typesetting and print. It's also not really getting better. One just gets accustomed to what is the norm at a certain point. Whenever they change the layout and look and feel of the newspaper I read the arguments are readability and ease of access. Well, I never had such a hard time reading my paper as today (with my old eyes).

Are we, like Dennett, willing to discard old views on our tools and models? When my first computer was a RCA 1802 based kit, that had 256 bytes of memory. My current laptop (from 2013) is a Dell Precision workstation with an extreme quad core processor and 16 GB of memory and ssd storage. Before I arrived there I worked with DEC-10, VAX and the whole range of Intel CPUs. So if you really want to compare a brain with a computer, take your choice.

I started with TEX on a 4 MHz desk top with 640 MB memory and a 10 MB hard disk. Running CONTEXT MKIV with LUATEX on such a machine is no option at all, but I still carry the burden of trying to write efficient code (which is still somewhat reflected in the code that makes up CONTEXT). In the decades that we have been using TEX we had to adapt! Demands changed, possibilities changed, technologies changed. And they keep changing. How many successive changes can a TEX user handle? Sometimes, when I look and listen I wonder.

If you look back, that is, if you read about the tens of thousands of years that it took humans to evolve ("The mind in the cave" by Lewis-Williams is a good exercise) you realize even more in what a fast-paced time we live and that we're witnessing transitions of another magnitude.

In the evolution of species some tools were invented multiple times, like eyes. You see the same in our TEX world: multiple (sub)macro packages, different font technologies, the same solutions but with an alternative approach. Some disappear, some stay around. Just like different circumstances demand different solutions in nature, so

do different situations in typesetting, for instance different table rendering solutions. Sometime I get the feeling that we focus too much on getting rid of all but one solution while more natural would be to accept diversity, like bio-diversity is accepted. Transitions nowadays happen faster but the question is if, like aeons before, we (have to) let them fade away. When evolution is discussed the terms 'random', 'selection', 'fit', and so on are used. This probably also applies to typography: at some point a font can be used a lot, but in the end the best readable and most attractive one will survive. Newspapers are printed in many copies, but rare beautiful books hold value. Of course, just like in nature some developments force the further path of development, we don't suddenly grow more legs or digits on our hands. The same happens with TEX on a smaller timescale: successors still have the same core technology, also because if we'd drop it, it would be something different and then give a reason to reconsider using such technology (which likely would result in going by another path).

1.7 Quality

Richard Dawkins "The Ancestor's Tale" is a non-stop read. In a discussion with Jared Diamond about religion and evolution they ponder this thread: you holding the hand of your mother who is handing her mother's hand and so on till at some point fish get into the picture. The question then is, when do we start calling something human? And a related question is, when does what we call morality creeps in? Is 50% neanderthaler human or not?

So, in the history of putting thoughts on paper: where does TEX fit in? When do we start calling something automated typesetting? When do we decide that we have quality? Is TEX so much different from its predecessors? And when we see aspects of TEX (or related font technology) in more modern programs, do we see points where we cross qualitative or other boundaries? Is a program doing a better job than a human? Where do we stand? There are fields where there is no doubt that machines outperform humans. It's probably a bit more difficult in aesthetic fields except perhaps when we lower the conditions and expectations (something that happens a lot).

For sure TEX will become obsolete, maybe even faster that we think, but so will other typesetting technologies. Just look back and have no illusions. Till then we can have our fun and eventually, when we have more free time than we need, we might use it out of hobbyism. Maybe TEX will be remembered by probably its most important side effect: the first large scale open source, the time when users met over programs, Knuth's disciples gathered in user groups, etc. The tools that we use are just a step in an evolution. And, as with evolution, most branches are pruned. So, when in the far future one looks back, will they even notice TEX? The ancestor's tail turns the tree upside down: at the end of the successful branch one doesn't see the dead ends.

Just a thought: CDs and media servers are recently being replaced (or at least accompanied) by Long Play records. In the shop where I buy my CDs the space allocated to

records grows at the cost of more modern media. So, maybe at some point retro-typesetting will pop up. Of course it might skip TEX and end up at woodcutting or printing with lead.

1.8 What mission

We rely on search engines instead of asking around or browsing libraries. Do students really still read books and manuals or do they just search and listen to lectures. Harari claims that instead of teaching kids facts in school we should just take for granted that they can get all the data they want and that we should learn them how to deal with data and adapt to what is coming. We take for granted that small devices with human voices show us the route to drive to BachoTEX, for instance, although by now I can drive it without help. In fact, kids can surprise you by asking if we're driving in Germany when we are already in Poland.

We accept that computer programs help physicians in analyzing pictures. Some wear watches that warn them about health issues, and I know a few people who monitor their sugar levels electronically instead of relying on their own measurements. We seem to believe and trust the programs. And indeed, we also believe that TEX does the job in the best way possible. How many people really understand the way TEX works?

We still have mailing lists where we help each other. There are also wikis and forums like stack exchange. But who says that even a moderate bit of artificial intelligence doesn't answer questions better. Of course there needs to be input (manuals, previous answers, etc.) but just like we need fewer people as workforce soon, the number of experts needed also can be smaller. And we're still talking about a traditional system like TEX. Maybe the social experience that we have on these media will survive somehow, although: how many people are members of societies, participate in demonstrations, meet weekly in places where ideas get exchanged, compared to a few decades ago? That being said, I love to watch posts with beautiful CONTEXT solutions or listen to talks by enthusiastic users who do things I hadn't expected. I really hope that this property survives, just like I hope that we will be able to see the difference between a real user's response and one from an intelligent machine (an unrealistic hope I fear). Satisfaction wins and just like our neurological subsystems at some point permanently adapt to thresholds (given that you trigger things often enough), we get accustomed to what TEX provides and so we stick to it.

1.9 Intelligence versus consciousness

Much of what we do is automated. You don't need to think of which leg to move and what foot to put down when you walk. Reacting to danger also to a large extent is automated. It doesn't help much to start thinking about how dangerous a lion can be when it's coming after you, you'd better move fast. Our limbic system is responsible for such automated behaviour, for instance driven by emotions. The more difficult tasks and thoughts about them happen in the frontal cortex (sort of).

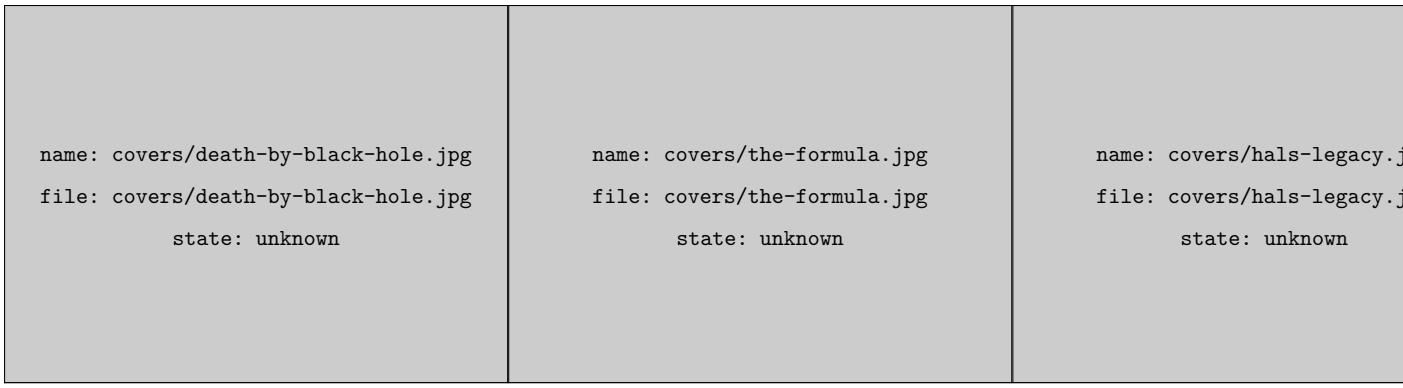


Figure 1.3

For most users TEX is like the limbic system: there is not much thinking involved, and the easy solutions are the ones used. Just like hitting a nerve triggers a chain of reactions, hitting a key eventually produces a typeset document. Often this is best because the job needs to get done and no one really cares how it looks; just copy a preamble, key in the text and assume that it works out well (enough). It is tempting to compare TEX's penalties, badness and other parameters with levels of hormones and neurotransmitters. Their function depends on where they get used and the impact can be accumulated, blocked or absent. It's all magic, especially when things interact.

Existing TEX users, developers and user groups of course prefer to think otherwise, that it is a positive choice by free will. That new users have looked around and arrived at TEX for good reason: their frontal cortex steering a deliberate choice. Well, it might have played a role but the decision to use TEX might in the end be due to survival skills: I want to pass this exam and therefore I will use that weird system called TEX.

All animals, us included, have some level of intelligence but also have this hard to describe property that we think makes us what we are. Intelligence and consciousness are not the same (at least we know a bit about the first but nearly nothing about the second). We can argue about how well composed some music is but why we like it is a different matter.

We can make a well thought out choice for using TEX for certain tasks but can we say why we started liking it (or not)? Why it gives us pleasure or maybe grief? Has it become a drug that we got addicted to? So, one can make an intelligent decision about using TEX but getting a grip on why we like it can be hard. Do we enjoy the first time struggle? Probably not. Do we like the folks involved? Yes, Don Knuth is a special and very nice person. Can we find help and run into a friendly community? Yes, and a unique one too, annoying at times, often stimulating and on the average friendly for all the odd cases running around.

Artificial intelligence is pretty ambitious, so speaking of machine intelligence is probably better. Is TEX an intelligent program? There is definitely some intelligence built in and the designer of that program is for sure very intelligent. The designer is also a conscious entity: he likes what he did and finds pleasure in using it. The program on

the other hand is just doing its job: it doesn't care how it's done and how long it takes: a mindless entity. So here is a question: do we really want a more intelligent program doing the job for us, or do those who attend conferences like BachoTEX enjoy TEXing so much that they happily stay with what they have now? Compared to rockets tumbling down and/or exploding or Mars landers thrashing themselves due to programming errors of interactions, TEX is surprisingly stable and bug free.

1.10 Individual versus group evolution

After listening for hours to Sapolsky you start getting accustomed to remarks about (unconscious) behaviour driven by genes, expression and environment, aimed at "spreading many copies of your genes". In most cases that is an individual's driving force. However, cooperation between individuals plays a role in this. A possible view is that we have now reached a state where survival is more dependent on a group than on an individual. This makes sense when we consider that developments (around us) can go way faster than regular evolution (adaptation) can handle. We take control over evolution, a mechanism that needs time to adapt and time is something we don't give it anymore.

Why does TEX stay around? It started with an individual but eventually it's the groups that keeps it going. A too-small group won't work but too-large groups won't work either. It's a known fact that one can only handle some 150 social contacts: we evolved in small bands that split when they became too large. Larger groups demanded abstract beliefs and systems to deal with the numbers: housing, food production, protection. The TEX user groups also provide some organization: they organize meetings, somehow keep development going and provide infrastructure and distributions. They are organized around languages. According to Diamond new languages are still discovered but many go extinct too. So the potential for language related user groups is not really growing.

Some of the problems that we face in this world have become too large to be dealt with by individuals and nations. In spite of what anti-globalists want we cannot deal with our energy hunger, environmental issues, lack of natural resources, upcoming technologies without global cooperation. We currently see a regression in cooperation by nationalistic movements, protectionism and the usual going back to presumed better times, but that won't work.

Local user groups are important but the number of members is not growing. There is some cooperation between groups but eventually we might need to combine the groups into one which might succeed unless one wants to come first. Of course we will get the same sentiments and arguments as in regular politics but on the other hand, we already have the advantage of TEX systems being multi-lingual and users sharing interest in the diversity of usage and users. The biggest challenge is to pass on what we have achieved. We're just a momentary highlight and let's not try to embrace some "TEX first" madness.

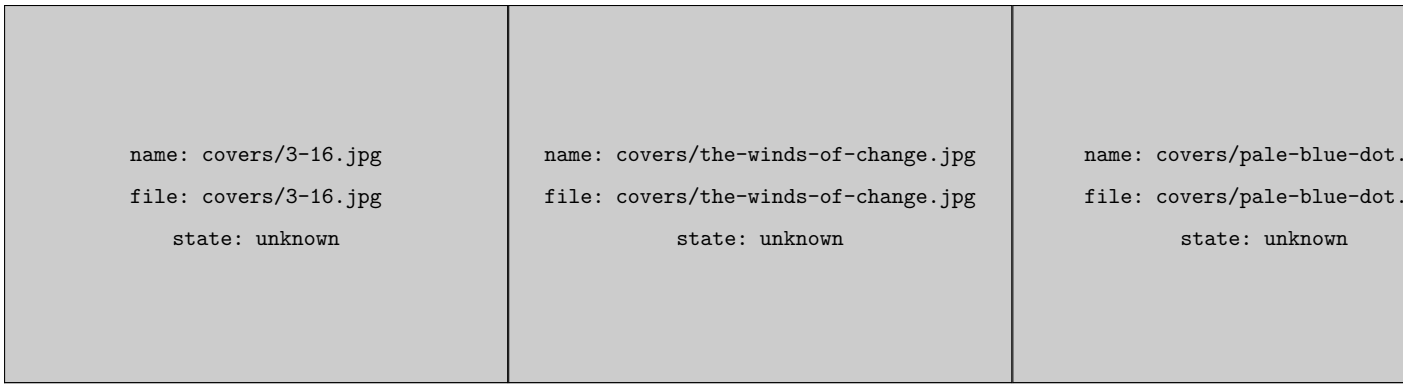


Figure 1.4

1.11 Sexes

Most species have two sexes but it is actually a continuum controlled by hormones and genetic expression: we just have to accept it. Although the situation has improved there are plenty of places where some gender relationships are considered bad even to the extent that one's life can be in danger. Actually having strong ideas about these issues is typically human. But in the end one has to accept the continuum.

In a similar way we just have to accept that TEX usage, application of TEX engines, etc. is a continuum and not a batch versus WYSIWYG battle any more. It's disturbing to read strong recommendations not to use this or that. Of the many macro packages that showed up only a few were able to survive. How do users of outlines look at bitmaps, how do DVI lovers look at PDF. But, as typesetting relates to esthetics, strong opinions come with the game.

Sapolsky reports about a group of baboons where due to the fact that they get the first choice of food the alpha males of pack got poisoned, so that the remaining suppressed males who treated the females well became dominant. In fact they can then make sure that no new alpha male from outside joins the pack without behaving like they do. A sort of social selection. In a similar fashion, until now the gatherings of TEXies managed to keep its social properties and has not been dominated by for instance commerce.

In the animal world often sexes relate to appearance. The word sexy made it to other domains as well. Is TEX sexy? For some it is. We often don't see the real colors of birds. What looks gray to us looks vivid to a bird which sees in a different spectrum. The same is true for TEX. Some users see a command line (shell) and think: this is great! Others just see characters and keystrokes and are more attracted to an interactive program. When I see a graphic made by METAPOST, I always note how exact it is. Others don't care if their interactive effort doesn't connect the dots well. Some people (also present here) think that we should make TEX attractive but keep in mind that like and dislike are not fixed human properties. Some mindsets might as well be the result from our makeup, others can be driven by culture.

1.12 Religion

One of Sapolsky's lectures is about religion and it comes in the sequence of mental variations including depression and schizophrenia, because all these relate to mental states, emotions, thresholds and such (all things human). That makes it a tricky topic which is why it has not been taped. As I was raised in a moderate Protestant tradition I can imagine that it's an uncomfortable topic instead. But there are actually a few years older videos around and they are interesting to watch and not as threatening as some might expect. Here I just stick to some common characteristics.

If you separate the functions that religions play into for instance explanation of the yet unknown, social interactions, control of power and regulation of morals, then it's clear why at TEX user group meetings the religious aspect of TEX has been discussed in talks. Those who see programs as infallible and always right and don't understand the inner working can see it as an almighty entity. In the Netherlands church-going diminishes but it looks like alternative meetings are replacing it (and I'm not talking of football matches). So what are our TEX meetings? What do we believe in? The reason that I bring up this aspect is that in the TEX community we can find aspects of the more extremist aspects of religions: if you don't use the macro package that I use, you're wrong. If you don't use the same operating system as I do, you're evil. You will be punished if you use the wrong editor for TEX? Why don't you use this library (which, by the way, just replaced that other one)? We create angels and daemons. Even for quite convinced atheists (it's not hard to run into them on youtube) a religion only survives when it has benefits, something that puzzles them. So when we're religious about TEX and friends we have to make sure that it's at least beneficial. Also, maybe we fall in Dennett's category of "believers who want to believe": it helps us to do our job if we just believe that we have the perfect tool. Religion has inspired visual and aural art and keeps doing that. (Don Knuth's current musical composition project is a good example of this.)

Scientists can be religious, in flexible ways too, which is demonstrated by Don Knuth. In fact, I'm pretty sure TEX would not be in the position it is in now if it weren't for his knowledgeable, inspirational, humorous, humble, and always positive presence. And for sure he's not at all religious about the open source software that he sent viral.

I'm halfway through reading "The Good Book of Human Nature" (An Evolutionary Reading of the Bible) a book about the evolution of the bible and monotheism which is quite interesting. It discusses for instance how transitions from a hunter to a farmer society demanded a change of rules and introduced stories that made sense in that changing paradigm. Staying in one place means that possessions became more important and therefore inheritance. Often when religion is discussed by behavioral biologists, historians and anthropologists they stress this cultural narrative aspect. Also mentioned is that such societies were willing to support (in food and shelter) the ones that didn't normally fit it but added to the spiritual character of religions. The social and welcoming aspect is definitely present in for instance BachoTEX conferences although a bystander

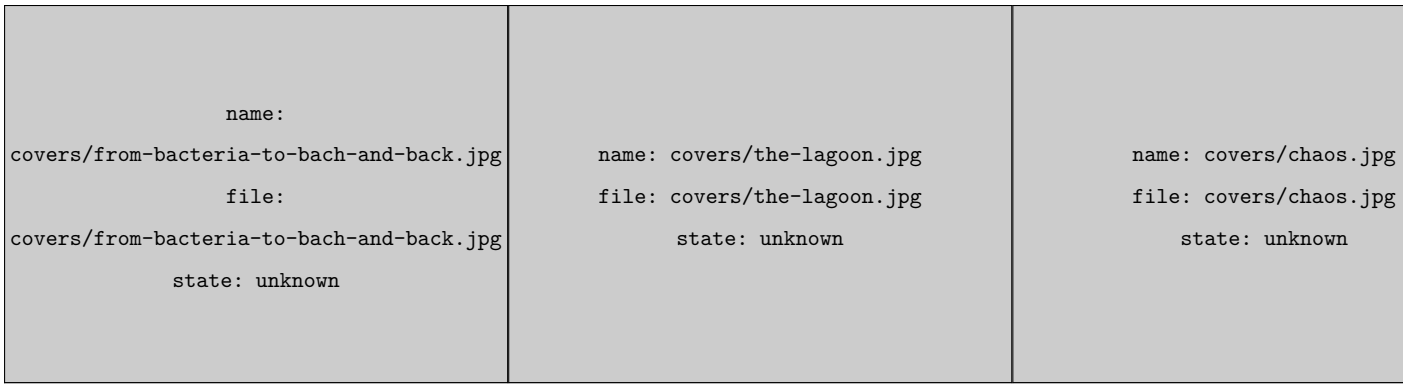


Figure 1.5

can wonder what these folks are doing in the middle of the night around a campfire, singing, drinking, frying sausages, spitting fire, and discussing the meaning of life.

Those who wrap up the state of religious affairs, do predictions and advocate the message, are sometimes called evangelists. I remember a TEX conference in the USA where the gospel of XML was preached (by someone from outside the TEX community). We were all invited to believe it. I was sitting in the back of the crowded (!) room and that speaker was not at all interested in who spoke before and after. Well, I do my share of XML processing with CONTEXT, but believe me: much of the XML that we see is not according to any gospel. It's probably blessed the same way as those state officials get blessed when they ask and pray for it in public.

It can get worse at TEX conferences. Some present here at BachoTEX might remember the PDF evangelists that we had show up at TEX conferences. You see this qualification occasionally and I have become quite allergic to qualifications like architect, innovator, visionary, inspirator and evangelist, even worse when they look young but qualify as senior. I have no problem with religion at all but let's stay away from becoming one. And yes, typography also falls into that trap, so we have to be doubly careful.

1.13 Chaotic solutions

The lectures on “chaos and reductionism” and “emergence and complexity” were the highlights in Sapolsky's lectures. I'm not a good narrator so I will not summarize them but it sort of boils down to the fact that certain classes of problems cannot be split up in smaller tasks that we understand well, after which we can reassemble the solutions to deal with the complex task. Emerging systems can however cook up working solutions from random events. Examples are colonies of ants and bees.

The TEX community is like a colony: we cook up solutions, often by trial and error. We dream of the perfect solutions but deep down know that esthetics cannot be programmed in detail. This is a good thing because it doesn't render us obsolete. At last year's BachoTEX, my nephew Teun and I challenged the anthill outside the canteen to

typeset the TEX logo with sticks but it didn't persist. So we don't need to worry about competition from that end. How do you program a hive mind anyway?

When chaos theory evolved in the second half of the previous century not every scientist felt happy about it. Instead of converging to more perfect predictions and control in some fields a persistent uncertainty became reality.

After about a decade of using TEX and writing macros to solve recurring situations I came to the conclusion that striving for a perfect TEX (the engine) that can do everything and anything makes no sense. Don Knuth not only stopped adding code when he could do what he needed for his books, he also stuck to what to me seems reasonable endpoints. Every hard-coded solution beyond that is just that: a hard-coded solution that is not able to deal with the exceptions that make up most of the more complex documents. Of course we can theorize and discuss at length the perfect never-reachable solutions but sometimes it makes more sense to admit that an able user of a desktop publishing system can do that job in minutes, just by looking at the result and moving around an image or piece of text a bit.

There are some hard-coded solutions and presets in the programs but with LUATEX and MPLIB we try to open those up. And that's about it. Thinking that for instance adding features like protrusion or expansion (or whatever else) always lead to better results is just a dream. Just as a butterfly flapping its wings on one side of the world can have an effect on the other side, so can adding a single syllable to your source completely confuse an otherwise clever column or page break algorithm. So, we settle for not adding more to the engine, and provide just a flexible framework.

A curious observation is that when Edward Lorenz ran into chaotic models it was partially due to a restart of a simulation midway, using printed floating point numbers that then in the computer were represented with a different accuracy than printed. Aware of floating point numbers being represented differently across architectures, Don Knuth made sure that TEX was insensitive to this so that its outcome was predictable, if you knew how it worked internally. Maybe LUATEX introduces a bit of chaos because the LUA we use has only floats. In fact, a few months ago we did uncover a bug in the backend where the same phenomena gave a chaotic crash.

In chaos theory there is the concept of an attractor. When visualized this can be the area (seemingly random) covered by a trajectory. Or it can be a single point where for instance a pendulum comes to rest. So what is our attractor? We have a few actually. First there is the engine, the stable core of primitives always present. You often see programs grow more complex every update and for sure that happened with ε-TEX, PDFTEX, XETEX and LUATEX. However there is always the core that is supposed to be stable. After some time the new kid arrives at a stable state not much different from the parent. The same is true for METAPOST. Fonts are somewhat different because the technology changes but in the end the shapes and their interactions become stable as well. Yet another example is TEX Live: during a year it might diverge from its route but eventually it settles down and enters the area where we expect it to end up. The TEX world is at times chaotic, but stable in the long run.

So, how about the existence, the reason for it still being around? One can speculate about its future trajectory but one thing is sure: as long as we break a text into paragraphs and pages TEX is hard to beat. But what if we don't need that any more? What if the concept of a page is no longer relevant? What if justified texts no longer matter (often designers don't care anyway)? What if students are no longer challenged to come up with a nice looking thesis? Do these collaborative tools with remote TEX processing really bring new long term users or is TEX then just one of the come-and-go tools?

1.14 Looking ahead

In an interview ("World of ideas") Asimov explains that science fiction evolved rapidly when people lived long enough to see that there was a future (even for their offspring) that is different from today. It is (at least for me) mind boggling to think of an evolution of hundreds of thousands of years to achieve something like language. Waiting for the physical being to arrive at a spot where you can make sounds, where the brain is suitable for linguistic patterns, etc. A few hundred years ago speed of any developments (and science) stepped up.

TEX is getting near 40 years old. Now, for software that **is** old! In that period we have seen computers evolve: thousands of times faster processing, even more increase in memory and storage. If we read about spaceships that travel at a reasonable fraction of the speed of light, and think that will not happen soon, just think back to the terminals that were sitting in computer labs when TEX was developed: 300 baud was normal. I actually spent quite some time on optimizing time-critical components of CONTEXT but on this timescale that is really a waste of time. But even temporary bottlenecks can be annoying (and costly) enough to trigger such an effort. (Okay, I admit that it can be a challenge, a kind of game, too.)

Neil Tyson, in the video "Storytelling of science" says that when science made it possible to make photos it also made possible a transition in painting to impressionism. Other technology could make the exact snapshot so there was new room for inner feelings and impressions. When the Internet showed up we went through a similar transition, but TEX actually dates from before the Internet. Did we also have a shift in typesetting? To some extent yes, browsers and real time rendering is different from rendering pages on paper. In what space and time are TEXies rooted?

We get older than previous generations. Quoting Sapolsky "... we are now living well enough and long enough to slowly fall apart." The opposite is happening with our tools, especially software: it's useful lifetime becomes shorter and changes faster each year. Just look at the version numbers of operating systems. Don Knuth expected TEX to last for a long time and compared to other software its core concept and implementation is doing surprisingly well. We use a tool that suits our lifespan! Let's not stress ourselves out too much with complex themes. (It helps to read "Why zebras don't get ulcers".)

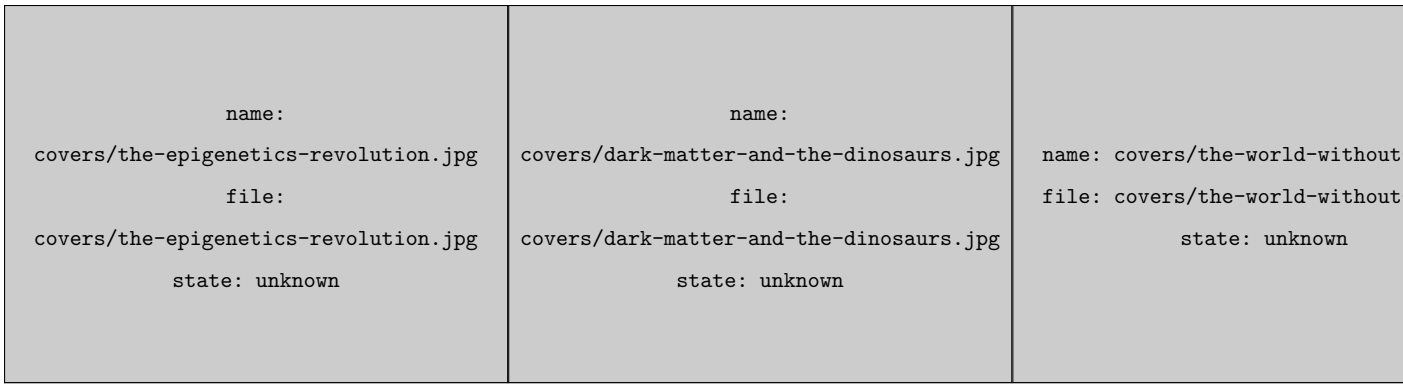


Figure 1.6

1.15 Memes

If you repeat a message often enough, even if it's something not true, it can become a meme that gets itself transferred across generations. Conferences like this is where they can evolve. We tell ourselves and the audience how good TEX is and because we spend so many hours, days, weeks, months using it, it actually must be good, or otherwise we would not come here and talk about it. We're not so stupid as to spend time on something not good, are we? We're always surprised when we run into a (potential) customer who seems to know TEX. It rings a bell, and it being around must mean something. Somehow the TEX meme has anchored itself when someone attended university. Even if experiences might have been bad or usage was minimal. The meme that TEX is the best in math typesetting is a strong survivor.

There's a certain kind of person who tries to get away with their own deeds and decisions by pointing to "fake news" and accusations of "mainstream media" cheating on them. But to what extent are our stories true about how easy TEX macro packages are to use and how good their result? We have to make sure we spread the right memes. And the user groups are the guardians.

Maybe macro packages are like memes too. In the beginning there was a bunch but only some survived. It's about adaptation and evolution. Maybe competition was too fierce in the beginning. Like ecosystems, organisms and cellular processes in biology we can see the TEX ecosystem, users and usage, as a chaotic system. Solutions pop up, succeed, survive, lead to new ones. Some look similar and slightly different input can give hugely different outcomes. You cannot really look too far ahead and you cannot deduce the past from the present. Whenever something kicks it off its stable course, like the arrival of color, graphics, font technologies, PDF, XML, ebooks, the TEX ecosystem has to adapt and find its stable state again. The core technology has proven to be quite fit for the kind of adaptation needed. But still, do it wrong and you get amplified out of existence, don't do anything and the external factors also make you extinct. There is no denial that (in the computer domain) TEX is surprisingly stable and adaptive. It's also hard not to see how conservatism can lead to extinction.

1.16 Inspiration

I just took some ideas from different fields. I could have mentioned quantum biology, which tries to explain some unexplainable phenomena in living creatures. For instance how do birds navigate without visible and measurable clues. How do people arrive at TEX while we don't really advertise? Or I could mention epigenetics and explorations in junk DNA. It's not the bit of the genome that we thought that matters, but also the expression of the genes driven by other factors. Offspring not only gets genetic material passed but it can get presets. How can the TEX community pass on Knuth's legacy? Do we need to hide the message in subtle ways? Or how about the quest for dark matter? Does it really exist or do we want (need) it to exist? Does TEX really have that many users, or do we cheat by adding the users that are enforced during college but don't like it at all? There's enough inspiration for topics at TEX conferences, we just have to look around us.

1.17 Stability

I didn't go into technical aspects of TEX yet. I must admit that after decades of writing macros I've reached a point where I can safely say that there will never be perfect automated solutions for really complex documents. When books about neural networks show up I wondered if it could be applied (but I couldn't). When I ran into genetic algorithms I tried to understand its possible impact (but I never did). So I stuck to writing solutions for problems using visualization: the trial and error way. Of course, speaking of ConTEXt, I will adapt what is needed, and others can do that as well. Is there a new font technology? Fine, let's support it as it's no big deal, just a boring programming task. Does a user want a new mechanism? No problem, as solving a reduced subset of problems can be fun. But to think of TEX in a reductionist way, i.e. solving the small puzzles, and to expect the whole to work in tandem to solve a complex task is not trivial and maybe even impossible. It's a good thing actually, as it keeps us on edge. Also, ConTEXt was designed to help you with your own solutions: be creative.

I mentioned my nephew Bram. He has seen part of this crowd a few times, just like his brother and sister do now. He's into artificial intelligence now. In a few years I'll ask him how he sees the current state of TEX affairs. I might learn a few tricks in the process.

In "The world without us" Weisman explores how fast the world would be void of traces of humankind. A mere 10.000 years can be more than enough. Looking back, that's about the time hunters became farmers. So here's a challenge: say that we want an ant culture that evolves to the level of having archaeologists to know that we were here at BachoTEX . . . what would we leave behind?

Sapolsky ends his series by stressing that we should accept and embrace individual differences. The person sitting next to you can have the same makeup but be just a bit more sensitive to depression or be the few percent with genes controlling schizophrenic behaviour. He stresses that knowing how things work or where things go wrong doesn't mean that we should fix everything. So look at this room full of TEXies: we don't need to

be all the same, use all the same, we don't need some dominance, we just need to accept and especially we need to understand that we can never fully understand (and solve) everything forever.

Predictions, one of the themes, can be hard. It's not true that science has the answer to everything. There will always be room for speculation and maybe we will always need metaphysics too. I just started to read "What we cannot know" by Sautoy. For sure those present here can not predict how TEX will go on and/or be remembered.

1.18 Children of TEX

I mentioned "Children of time". The author lets you see their spidery world through spider eyes and physiology. They have different possibilities (eyesight, smell) than we do and also different mental capabilities. They evolve rapidly and have to cope conceptually with signals from a human surveillance satellite up in the sky. Eventually they need to deal with a bunch of (of course) quarrelling humans who want their place on the planet. We humans have some pre-occupation with spiders and other creatures. In a competitive world it is sometimes better to be suspicious (and avoid and flee) that to take a risk of being eaten. A frequently used example is that a rustle in a bush can be the wind or a lion, so best is to run.

We are not that well adapted to our current environment. We evolved at a very slow pace so there was no need to look ahead more than a year. And so we still don't look too far ahead (and choose politicians accordingly). We can also not deal that well with statistics (Dawkins's "Climbing Mount Probability" is a good read) so we make false assumptions, or just forget.

Does our typeset text really look that good on the long run, or do we cheat with statistics? It's not too hard to find a bad example of something not made by TEX and extrapolate that to the whole body of typeset documents. Just like we can take a nice example of something done by TEX and assume that what we do ourselves is equally okay. I still remember the tests we did with PDFTEX and hz. When Hàn Thế Thành and I discussed that with Hermann Zapf he was not surprised at all that no one saw a difference between the samples and instead was focusing on aspects that TEXies are told to look at, like two hyphens in a row.

A tool like TEX has a learning curve. If you don't like that just don't use it. If you think that someone doesn't like that, don't enforce this tool on that someone. And don't use (or lie with) statistics. Much better arguments are that it's a long-lived stable tool with a large user base and support. That it's not a waste of time. Watching a designer like Hermann Zapf draw shapes is more fun than watching click and point in heavily automated tools. It's probably also less fun to watch a TEXie converge towards a solution.

Spiders are resilient. Ants maybe even more. Ants will survive a nuclear blast (mutations might even bring them benefits), they can handle the impact of a meteorite, a change in climate won't harm them much. Their biggest enemy is probably us, when we

try to wipe them out with poison. But, as long as they keep a low profile they're okay. TEX doesn't fit into the economic model as there is no turnaround involved, no paid development, it is often not seen at all, it's just a hit in a search engine and even then you might miss it (if only because no one pays for it being shown at the top).

We can learn from that. Keeping a low profile doesn't trigger the competition to wipe you out. Many (open source) software projects fade away: some big company buys out the developer and stalls the project or wraps what they bought in their own stuff, other projects go professional and enterprise and alienate the original users. Yet others abort because the authors lose interest. Just like the ideals of socialism don't automatically mean that every attempt to implement it is a success, so not all open source and free software is good (natured) by principle either. The fact that communism failed doesn't mean that capitalism is better and a long term winner. The same applies to programs, whether successful or not.

Maybe we should be like the sheep. Dennett uses these animals as a clever species. They found a way to survive by letting themselves (unconsciously) be domesticated. The shepherd guarantees food, shelter and protection. He makes sure they don't get ill. Speaking biologically: they definitely made sure that many copies of their genes survived. Cows did the same and surprisingly many of them are related due to the fact that they share the same father (something now trying to be reverted). All TEX spinoffs relate to the same parent, and those that survived are those that were herded by user groups. We see bits and pieces of TEX end up in other applications. Hyphenation is one of them. Maybe we should settle for that small victory in a future hall of fame.

When I sit on my balcony and look at the fruit trees in my garden, some simple math can be applied. Say that one of the apple trees has 100 apples per year and say that this tree survives for 25 years (it's one of those small manipulated trees). That makes 2.500 apples. Without human intervention only a few of these apples make it into new trees, otherwise the whole world would be dominated by apple trees. Of course that tree now only survives because we permit it to survive, and for that it has to be humble (something that is very hard for modern Apples). Anyway, the apple tree doesn't look too unhappy.

A similar calculation can be done for birds that nest in the trees and under my roof. Given that the number of birds stays the same, most of energy spent on raising offspring is wasted. Nevertheless they seem to enjoy life. Maybe we should be content if we get one enthusiastic new user when we demonstrate TEX to thousands of potential users.

Maybe, coming back to the themes of the conference, we should not come up with these kinds of themes. We seem to be quite happy here. Talking about the things that we like, meeting people. We just have to make sure that we survive. Why not stay low under the radar? That way nothing will see us as a danger. Let's be like the ants and spiders, the invisible hive mind that carries our message, whatever that is.

When Dennett discusses language he mentions (coined) words that survive in language. He also mentions that children pick up language no matter what. Their minds

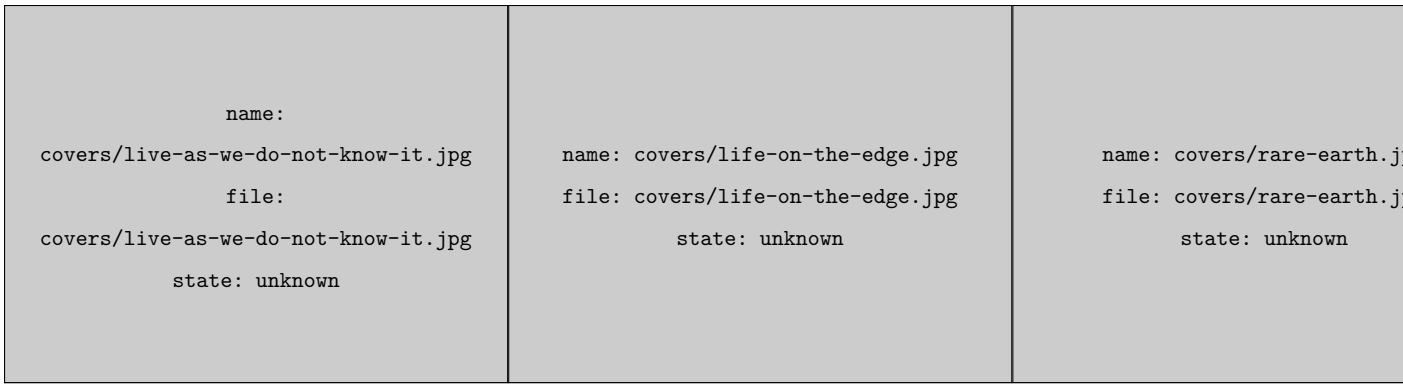


Figure 1.7

are made for it. Other animals don't do that: they listen but don't start talking back. Maybe TEX is just made for certain minds. Some like it and pick it up, while for others it's just noise. There's nothing wrong with that. Predilection can be a user property.

1.19 The unexpected

In a discussion with Dawkins the well-spoken astrophysicist Neil deGrasse Tyson brings up the following. We differ only a few percent in DNA from a chimp but quite a lot in brain power, so how would it be if an alien that differs a few percent (or more) passes by earth. Just like we don't talk to ants or chimps or whatever expecting an intelligent answer, whatever passes earth won't bother wasting time on us. Our rambling about the quality of typesetting probably sounds alien to many people who just want to read and who happily reflow a text on an ebook device, not bothered by a lack of quality.

We tend to take ourselves as reference. In "Rare Earth" Ward and Brownlee extrapolate the possibility of life elsewhere in the universe. They are not alone in thinking that while on one hand applying statistics to these formulas of possible life on planets there might also be a chance that we're the only intelligent species ever evolved. In a follow up, "Life as we do not know it" paleontologist and astrobiologist Ward (one of my favourite authors) discusses the possibility of life not based on carbon, which is not natural for a carbon based species. Carl Sagan once pointed out that an alien species looking down to earth can easily conclude that cars are the dominant species on earth and that the thingies crawling in and out them are some kind of parasites. So, when we look at the things that somehow end up on paper (as words, sentences, ornaments, etc.), what is dominant there? And is what we consider dominant really that dominant in the long run? You can look at a nice page as a whole and don't see the details of the content. Maybe beauty hides nonsense.

When TEXies look around they look to similar technologies. Commands in shells and solutions done by scripting and programming. This make sense in the perspective of survival. However, if you want to ponder alternatives, maybe not for usage but just for fun, a completely different perspective might be needed. You must be willing to accept that communicating with a user of a WYSIWYG program might be impossible. If mutual

puzzlement is a fact, then they can either be too smart and you can be too dumb or the reverse. Or both approaches can be just too alien, based on different technologies and assumptions. Just try to explain TEX to a kid 40 years younger or to an 80 year old grandparent for that matter. Today you can be very clever in one area and very stupid in another.

In another debate, Neil deGrasse Tyson asks Dawkins the question why in science fiction movies the aliens look so human and when they don't, why they look so strange, for instance like cumbersome sluggish snails. The response to that is one of puzzlement: the opponent has no reference of such movies. In discussions old TEXies like to suggest that we should convert young users. They often don't understand that kids live in a different universe.

How often does that happen to us? In a world of many billions TEX has its place and can happily coexist with other typesetting technologies. Users of other technologies can be unaware of us and even create wrong images. In fact, this also happens in the community itself: (false) assumptions turned into conclusions. Solutions that look alien, weird and wrong to users of the same community. Maybe something that I present as hip and modern and high-TEX and promising might be the opposite: backward, old-fashioned and of no use to others. Or maybe it is, but the audience is in a different mindset. Does it matter? Let's just celebrate that diversity. (So maybe, instead of discussing the conference theme, I should have talked about how I abuse LUATEX in controlling lights in my home as part of some IoT experiments.)

1.20 What drives us

I'm no fan of economics and big money talk makes me suspicious. I cannot imagine working in a large company where money is the drive. It also means that I have not much imagination in that area. We get those calls at the office from far away countries who are hired to convince us by phone of investments. Unfortunately mentioning that you're not at all interested in investments or that multiplying money is irrelevant to you does not silence the line. You have to actively kill such calls. This is also why I probably don't understand today's publishing world where money also dominates. Recently I ran into talks by Mark Blyth about the crisis (what crisis?) and I wish I could argue like he does when it comes to typesetting and workflows. He discusses quite well that most politicians have no clue what the crisis is about.

I think that the same applies to the management of publishers: many have no clue what typesetting is about. So they just throw lots of money into the wrong activities, just like the central banks seem to do. It doesn't matter if we TEXies demonstrate cheap and efficient solutions.

Of course there are exceptions. We're lucky to have some customers that do understand the issues at hand. Those are also the customers where authors may use the tools themselves. Educating publishers, and explaining that authors can do a lot, might be a premise, predilection and prediction in one go! Forget about those who don't get

it: they will lose eventually, unfortunately not before they have reaped and wasted the landscape.

Google, Facebook, Amazon, Microsoft and others invest a lot in artificial intelligence (or, having all that virtual cash, just buy other companies that do). They already have such entities in place to analyze whatever you do. It is predicted that at some point they know more about you then you know yourself. Reading Luke Dormehl's "The Formula" is revealing. So what will that do with our so-called (disputed by some) free will? Can we choose our own tools? What if a potential user is told that all his or her friends use WhateverOffice so they'd better do that too? Will subtle pressure lead them or even us users away from TEX? We already see arguments among TEXies, like "It doesn't look updated in 3 years, is it still good?" Why update something that is still valid? Will the community be forced to update everything, sort of fake updates. Who sets out the rules? Do I really need to update (or re-run) manuals every five years?

Occasionally I visit the Festo website. This is a (family owned) company that does research at the level that used to be common in large companies decades ago. If I had to choose a job, that would be the place to go to. Just google for "festo bionic learning network" and you understand why. We lack this kind of research in the field we talk about today: research not driven by commerce, short term profit, long term control, but because it is fundamental fun.

Last year Alan Braslau and I spent some time on BIBTEX. Apart from dealing with all the weird aspects of the APA standard, dealing with the inconsistently constructed author fields is a real pain. There have been numerous talks about that aspect here at BachoTEX by Jean-Michel Hufflen. We're trying to deal with a more than 30-year-old flawed architecture. Just look back over a curve that backtracks 30 years of exponential development in software and databases and you realize that it's a real waste of time and a lost battle. It's fine to have a text based database, and stable formats are great, but the lack of structure is appalling and hard to explain to young programmers. Compare that to the Festo projects and you realize that there can be more challenging projects. Of course, dealing with the old data can be a challenge, a necessity and eventually even be fun, but don't even think that it can be presented as something hip and modern. We should be willing to admit flaws. No wonder that Jean-Michel decided to switch to talking about music instead. Way more fun.

Our brains are massively parallel bio-machinery. Groups of neurons cooperate and compete for attention. Coming up with solutions that match what comes out of our minds demands a different approach. Here we still think in traditional programming solutions. Will new ideas about presenting information, the follow up on books come from this community? Are we the innovative Festo or are we an old dinosaur that just follows the fashion?

1.21 User experience

Here is a nice one. Harari spends many pages explaining that research shows that when an unpleasant experience has less unpleasantness at the end of the period involved, the

overall experience is valued according to the last experience. Now, this is something we can apply to working with TEX: often, the more you reach the final state of typesetting the more it feels as all hurdles are in the beginning: initial coding, setting up a layout, figuring things out, etc.

It can only get worse if you have a few left-over typesetting disasters but there adapting the text can help out. Of course seeing it in a cheap bad print can make the whole experience bad again. It happens. There is a catch here: one can find lots of bad-looking documents typeset by TEX. Maybe there frustration (or indifference) prevails.

I sometimes get to see what kind of documents people make with CONTEXT and it's nice to see a good looking thesis with diverse topics: science, philosophy, music, etc. Here TEX is just instrumental, as what it is used for is way more interesting (and often also more complex) than the tool used to get it on paper. We have conferences but they're not about rocket science or particle accelerators. Proceedings of such conferences can still scream TEX, but it's the content that matters. Here somehow TEX still sells itself, being silently present in rendering and presentations. It's like a rootkit: not really appreciated and hard to get rid of. Does one discuss the future of rootkits other than in the perspective of extinction? So, even as an invisible rootkit, hidden in the workings of other programs, TEX's future is not safe. Sometimes, when you install a Linux system, you automatically get this large TEX installation, either because of dependencies or because it is seen as a similar toolkit as for instance Open (or is it Libre) Office. If you don't need it, that user might as well start seeing it as a (friendly) virus.

1.22 Conclusion

At some point those who introduced computers in typesetting had no problem throwing printing presses out of the window. So don't pity yourself if at some point in the near future you figure out that professional typesetting is no longer needed. Maybe once we let machines rule the world (even more) we will be left alone and can make beautiful documents (or whatever) just for the joy, not bothering if we use outdated tools. After all, we play modern music on old instruments (and the older rock musicians get, the more they seem to like acoustic).

There are now computer generated compositions that experienced listeners cannot distinguish from old school. We already had copies of paintings that could only be determined forgeries by looking at chemical properties. Both of these (artificial) arts can be admired and bring joy. So, the same applies to fully automated typeset novels (or runtime rendered ebooks). How bad is that really? You don't dig channels with your hand. You don't calculate logarithmic tables manually any longer.

However, one of the benefits of the Internet is watching and listening to great minds. Another is seeing musicians perform, which is way more fun that watching a computer (although googling for "animusic" brings nice visuals). Recently I ran into a wooden musical computer made by "Wintergatan" which reminded me of the "Paige Compositor" that we use in a LUATEX cartoon. Watching something like that nicely compensates

for a day of rather boring programming. Watching how the marble machine x (mmx) evolves is yet another nice distraction.

Now, the average age of the audience here is pretty high even if we consider that we get older. When I see solutions of ConTeXt users (or experts) posted by (young) users on the mailing list or stack exchange I often have to smile because my answer would have been worse. A programmable system invokes creative solutions. My criterion is always that it has to look nice in code and has some elegance. Many posted solutions fit. Do we really want more automation? It's more fun to admire the art of solutions and I'm amazed how well users use the possibilities (even ones that I already forgot).

One of my favourite artists on my weekly "check youtube" list is Jacob Collier. Right from when I ran into him I realized that a new era in music had begun. Just google for his name and "music theory interview" and you probably understand what I mean. When Dennett comments on the next generation (say up to 25) he wonders how they will evolve as they grow up in a completely different environment of connectivity. I can see that when I watch family members. Already long ago Greg Bear wrote the novel "Darwin's Children". It sets you thinking and when looking around you even wonder if there is a truth in it.

There are folks here at BachoTeX who make music. Now imagine that this is a conference about music and that the theme includes the word "future". Then, imagine watching that video. You see some young musicians, one of them probably one of the musical masterminds of this century, others instrumental to his success, for instance by wrapping up his work. While listening you realize that this next generation knows perfectly well what previous generations did and achieved and how they influenced the current. You see the future there. Just look at how old musicians reflect on such videos. (There are lots of examples of youth evolving into prominent musicians around and I love watching them). There is no need to discuss the future, in fact, we might make a fool of ourselves doing so. Now back to this conference. Do we really want to discuss the future? What we think is the future? Our future? Why not just hope that in the flow of getting words on a medium we play our humble role and hope we're not forgotten but remembered as inspiration.

One more word about predicting the future. When Arthur Clarke's "2001: A Space Odyssey" was turned into a movie in 1968, a lot of effort went into making sure that the not so far ahead future would look right. In 1996 scientists were asked to reflect on these predictions in "Hal's Legacy". It turned out that most predictions were plain wrong. For instance computers got way smaller (and even smaller in the next 20 years) while (self-aware) artificial intelligence had not arrived either. So, let's be careful in what we predict (and wish for).

1.23 No more themes

We're having fun here, that's why we come to BachoTeX (predilection). That should be our focus. Making sure that TeX's future is not so much in the cutting edge but in

providing fun to its users (prediction). So we just have to make sure it stays around (premise). That's how it started out. Just watch at Don Knuth's 3:16 poster: via TEX and METAFONT he got in contact with designers and I wouldn't be surprised if that subproject was among the most satisfying parts. So, maybe instead of ambitious themes the only theme that matters is: show what you did and how you did it.

Advertising TEX 2

I can get upset when I hear TEXies boast about the virtues of TEX compared to for instance Microsoft Word. Not that I feel responsible for defending a program that I never use(d) but attacking something for no good reason makes not much sense to me. It is especially annoying when the attack is accompanied by a presentation that looks pretty bad in design and typography. The best advertisements for TEX should of course come from outside the TEX community, by people impressed by its capabilities. How many TEXies can really claim that Word is bad when they never tried to make something in it with a similar learning curve as they had in TEX or the same amount of energy spent in editing and perfecting a word-processor-made document.

In movies where computer technology plays a role one can encounter weird assumptions about what computers and programs can do. Run into a server room, pull one disk out of a RAID-5 array and get all information from it. Connect some magic device to a usb port of a phone and copy all data from it in seconds. Run a high speed picture or fingerprint scan on a computer (probably on a remote machine) and show all pictures flying by. Okay, it's not so far from other unrealistic aspects in movies, like talking animals, so maybe it is just a metaphor for complexity and speed. When zapping channels on my television I saw figure 2.1 and as the media box permits replay I could make a picture. I have no clue what the movie was about or what movie it was so a reference is lacking here. Anyway it's interesting that seeing a lot of TEX code flying by can impress someone: the viewer, even if no TEXie will ever see that on the console unless in some error or tracing message and even then it's hard to get that amount. So, the viewer will never realize that what is seen is definitely not what a TEXie wants to see.

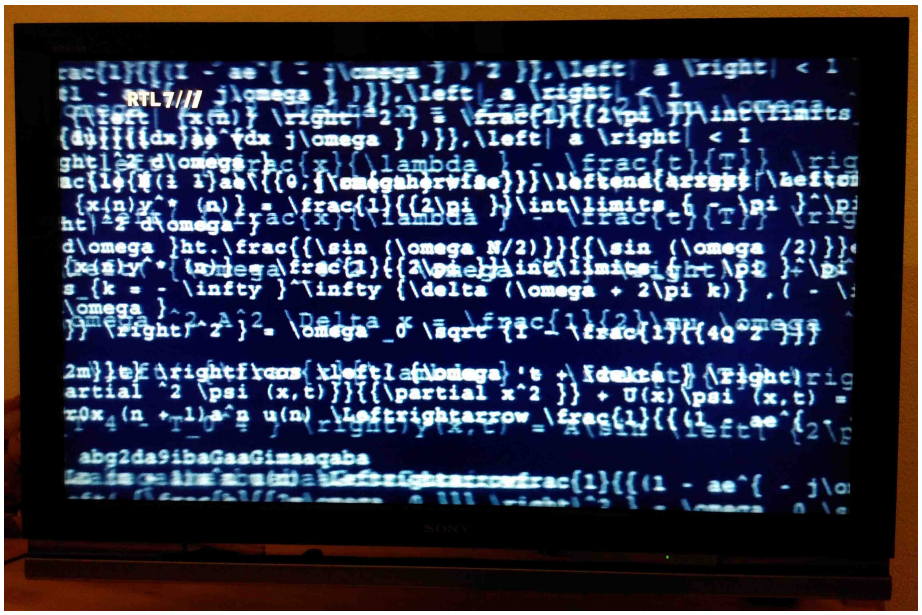


Figure 2.1 TEX in a movie

So, as that kind of free advertisement doesn't promote TEX well, what of an occasional mentioning of TEX in highly-regarded literature? When reading "From bacteria to Bach and back, the evolution of minds" by Daniel Dennett I ran into the following:

> "In Microsoft Word, for instance, there are the typographical operations of superscript and subscript, as illustrated by
>
> $\text{base}^{\text{power}}$
>
> and
>
> $\text{human}_{\text{female}}$
>
> But try to add another superscript to $\text{base}^{\text{power}}$—it *should* work, but it doesn't! In mathematics, you can raise powers to powers to powers forever, but you can't get Microsoft Word to display these (there are other text-editing systems, such as TeX, that can). Now, are we sure that human languages make use of true recursion, or might some or all of them be more like Microsoft Word? Might our interpretation of grammars as recursive be rather an elegant mathematical idealization of the actual "moving parts" of a grammar?"

Now, that book is a wonderfully interesting read and the author often refers to other sources. When one reads some reference (with a quote) then one assumes that what one reads is correct, and I have no reason to doubt Dennett in this. But this remark about TEX has some curious inaccuracies.[1]

First of all a textual raise or lower is normally not meant to be recursive. Nesting would have interesting consequences for the interline space so one will avoid it whenever possible. There are fonts that have superscript and subscript glyphs and even UNICODE has slots for a bunch of characters. I'm not sure what Word does: take the special glyph or use a scaled down copy?

Then there is the reference to TEX where we can accept that the "E" is not lowered but just kept as a regular "e". Actually the mentioning of nested scripts refers to typesetting math and that's what the superscripts and subscripts are for in TEX. In math mode however, one will normally raise or lower symbols and numbers, not words: that happens in text mode.

While Word will use the regular text font when scripting in text mode, a TEX user will either have to use a macro to make sure that the right size (and font) is used, or one can revert to math mode. But how to explain that one has to enter math and then explicitly choose the right font? Think of this:

```
efficient\high{efficient} or
```

[1] Of course one can wonder in general that when one encounters such an inaccuracy, how valid other examples and conclusions are. However, consistency in arguments and confirmation by other sources can help to counter this.

```
efficient$^{\text{efficient}}$ or \par
{\bf efficient\high{efficient} or
efficient$^{\text{efficient}}$}
```

Which gives (in Cambria)

efficient$^{\text{efficient}}$ or efficient$^{\text{efficient}}$ or

efficient$^{\text{efficient}}$ or efficient$^{\text{efficient}}$

Now this,

```
efficient\high{efficient\high{efficient}} or
efficient$^{\text{efficient$^{\text{efficient}}$}}$ or \par
{\bf efficient\high{efficient\high{efficient}} or
efficient$^{\text{efficient$^{\text{efficient}}$}}$}
```

will work okay but the math variant is probably quite frightening at a glance for an average Word user (or beginner in TEX) and I can understand why someone would rather stick to click and point.

efficient$^{\text{efficient}^{\text{efficient}}}$ or efficient$^{\text{efficient}^{\text{efficient}}}$ or

efficient$^{\text{efficient}^{\text{efficient}}}$ or efficient$^{\text{efficient}^{\text{efficient}}}$

Oh, and it's tempting to try the following:

```
efficient{\addff{f:superiors}efficient}
```

but that only works with fonts that have such a feature, like Cambria:

efficient$^{\text{efficient}}$

To come back to Dennett's remark: when typesetting math in Word, one just has to switch to the math editing mode and one can have nested scripts! And, when using TEX one should not use math mode for text scripts. So in the end in both systems one has to know what one is doing, and both systems are equally capable.

The recursion example is needed in order to explain how (following recent ideas from Chomsky) for modern humans some recursive mechanism is needed in our wetware. Now, I won't go into details about that (as I can only mess up an excellent explanation) but if you want to refer to TEX in some way, then expansion[2] of (either combined or not) snippets of knowledge might be a more interesting model than recursion, because much of what TEX is capable of relates to expansion. But I leave that to others to explore.[3]

[2] Expanding macros actually works well with tail recursion.

[3] One quickly starts thinking of how expandafter, `noexpand`, `unexpanded`, `protected` and other primitives can be applied to language, understanding and also misunderstanding.

Now, comparing TEX to Word is always kind of tricky: Word is a text editor with typesetting capabilities and TEX is a typesetting engine with programming capabilities. Recursion is not really that relevant in this perspective. Endless recursion in scripts makes little sense and even TEX has its limits there: the TEX math engine only distinguishes three levels (text, script and scriptscript) and sometimes I'd like to have a level more. Deeper nesting is just more of scriptscript unless one explicitly enforces some style. So, it's recursive in the sense that there can be many levels, but it also sort of freezes at level three.

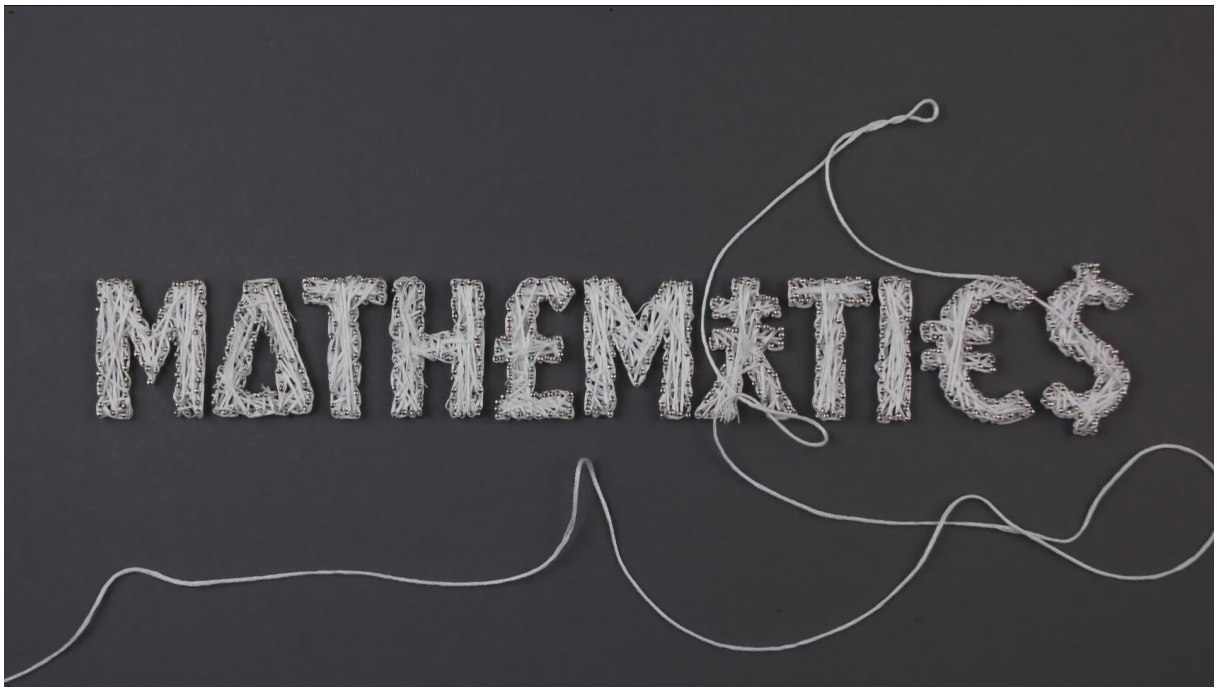


Figure 2.2 Nicer than TEX

I love TEX and I like what you can do with it and it keeps surprising me. And although mathematics is part of that, I seldom have to typeset math myself. So, I can't help that figure 2.2 impresses me more. It even has the so-familiar-to-TEXies dollar symbols in it: the poem "Poetry versus Orchestra" written by Hollie McNish, music composed by Jules Buckley and artwork by Martin Pyper (I have the DVD but you can also find it on YOUTUBE). It reminds me of Don Knuth's talk at a TUG meeting. In TUGBOAT 31:2 (2010) you can read Don's announcement of his new typesetting engine iTEX: "Output can be automatically formatted for lasercutters, embroidery machines, 3D printers, milling machines, and other CNC devices . . .". Now that is something that Word can't do!

Why use TEX? 3

Hans Hagen
Hasselt NL
July 2021 (public 2023)

3.1 Introduction

Let's assume that you know what TEX is: a program that interprets a language with the same name that makes it possible to convert (tagged) input into for instance PDF. For many of its users it is a black box: you key in some text, hit a button and get some typeset result in return. After a while you start tweaking this black box, meet other users (on the web), become more fluent and stick to it forever.

But now let's assume that you don't know TEX and are in search of a system that helps you create beautiful documents in an efficient way. When your documents have a complex structure you are probably willing to spend some time on figuring out what the best tool is. Even if a search lets you end up with something called TEX, a three letter word with a dropped E, you still don't know what it is. It helps to search for `\TeX` which is pronounced as `tech`. Advertisement for TEX is often pretty weak. It's rather easy to point to the numerous documents that can be found on the web. But what exactly does TEX do and what are its benefits? In order to answer this we need to know who you are: an author, editor, an organization that deals with documents or needs to generate readable output, like publishers do.

3.2 Authors

We start with authors. Students of sciences that use mathematics don't have much of a choice. But most of these documents hardly communicate the message that "Everyone should use TEX." or that "All documents produced by TEX look great." but they do advocate that for rendering math it is a pretty good system. The source code of these documents often look rather messy and unattractive and for a non-math user it can be intimidating. Choosing some lightweight click-and-ping alternative looks attractive.

Making TEX popular is not going to happen by convincing those who have to write an occasional letter or report. They should just use whatever suits them. On the other hand if you love consistency, long term support, need math, are dealing with a rare language or script, like to reuse content, prefer different styling from one source, use one source for multiple documents, or maybe love open source tools, then you are a candidate. Of course there is a learning curve but normally you can master TEX rather fast and once you get the hang of it there's often no way back. But you always need to invest a bit beforehand.

So what authors are candidates for TEX? It could be that TEX is the only tool that does the job. If so, you probably learned that from someone who saw you struggle or had the same experience and wrote or talked about it somewhere. In that case using TEX for creating just one document (like a thesis) makes sense. Otherwise, you should really wonder if you want to invest time in a tool that you probably have to ditch later on as most organizations stick to standard (commercial) word processing tools.

Talking to customers we are often surprised that people have heard about TEX, or even used it for a few documents in college. Some universities just prescribe the use of TEX for reporting, so not much of a choice there. Memories are normally rather positive in the sense that they know that it can do the job and that it's flexible.

User group journals, presentations at TEX meetings, journals, books and manuals that come with TEX macro packages can all be used to determine if this tool suits an author. Actually, I started using TEX because the original TEXbook had some magic, and reading it was just that: reading it, as I had no running implementation. A few years later, when I had to write (evolving) reports, I picked up again. But I'm not a typical user.

3.3 Programmers

When you are a programmer who has to generate reports, for instance in PDF, or write manuals, then TEX can really be beneficial. Of course TEX is not always an obvious choice, but if you're a bit able to use it it's hard to beat in quality, flexibility and efficiency. I'm often surprised that companies are willing to pay a fortune for functionality that basically comes for free. Programmers are accustomed to running commands and working in a code editor with syntax highlighting so that helps too. They also recognize when something can be done more efficiently.

When you need to go from some kind of input (document source, database, generated) to some rendered output there currently are a few endpoints: a (dynamic) HTML page, a PDF document, something useable in a word processor, or a representation using the desktop user interface. It's the second category where TEX is hard to beat but even using TEX and METAPOST for creating a chart can make sense.

There are of course special cases where TEX fits in nicely. Say that you have to combine PDF documents. There are numerous tools to do that and TEX is one. The advantage of TEX over other tools is that it's trivial to add additional text, number pages, provide headers and footers. And it will work forever. Why? Because TEX has been around for decades and will be around for decades to come. It's an independent component. The problem with choosing for TEX is that the starting point is important. The question is not “What tool should I use?” but “What problem do I need to solve?”. An open discussion about the objectives and possibilities is needed, not some checklist based on assumptions. If you don't know TEX and have never worked with a programmable typesetting environment, you probably don't see the possibilities. In fact, you might even choose for TEX for the wrong reasons.

The problem with this category of users is that they seldom have the freedom to choose their tools. There are not that many jobs where the management is able to recognize the clever programmer who can determine that TEX is suitable for a lot of jobs and can save money and time. Even the long term availability and support is not an argument since not only most tools (or even apis) changes every few years but also organizations themselves change ownership, objectives, and personnel on a whim. The concept of 'long term' is hard to grasp for most people (just look at politics) and it's only in retrospect that one can say 'We used that toolkit for over a decade.'

3.4 Organizations

Authors (often) have the advantage that they can choose themselves: they can use what they like. In practice any decent programmer is able to find the suitable tools but convincing the management to use one of them can be a challenge. Here we're also talking of 'comfort zones': you have to like a tool(chain). Organizations normally don't look for TEX. Special departments are responsible for choosing and negotiating whatever is used in a company. Unfortunately companies don't always start from the open question "We have this problem, we want to go there, what should we do?" and then discuss options with for instance those who know TEX. Instead requirements are formulated and matches are found. The question then is "Are these requirements cut in stone?" and if not (read: we just omit some requirements when most alternatives don't meet them), were other requirements forgotten? Therefore organizations can end up with the wrong choice (using TEX in a situation where it makes no sense) or don't see opportunities (not using TEX while it makes most sense). It doesn't help that a hybrid solution (use a mix of TEX and other tools) is often not an option. Where an author can just stop using a tool after a few days of disappointment, and where a programmer can play around a bit before making a choice, an organization probably best can start small with a proof of concept.

Let's take a use case. A publisher wants to automatically convert XML files into PDF. One product can come from multiple sources (we have cases where thousands of small XML files combine into one final product). Say that we have three different layouts: a theory book, a teachers manual and an answer book. In addition special proofing documents have to be rendered. The products might be produced on demand with different topics in any combination. There is at least one image and table per page, but there can be more. There are color and backgrounds used, tables of contents generated, there is extensive cross referencing and an index. Of course there is math.

Now let's assume an initial setup costs 20K Euro and, what happens often when the real products show up, a revision after one year takes the same amount. We also assume 10K for the following eight years for support. So, we end up with 120K over 10 years. If one goes cheap we can consider half of that, or we can be pessimistic and double the amount.

The first year 10K pages are produced, the second year 20K and after that 30K per year. So, we're talking of 270K pages. If we include customer specific documents and proofing we might as well end up with a multiple of that.

So, we have 120K Euro divided by 270K pages or about half an Euro per page. But likely we have more pages so it costs less. If we double the costs then we can assume that some major changes took place which means more pages. In fact we had projects where the layout changed, all documents were regenerated and the costs were included in the revision, so far from double. We also see many more pages being generated so in practice the price per page drops below half an Euro. The more we process the cheaper it gets and one server can produce a lot of pages!

Now, the interesting bit of such a calculation is that the costs only concern the hours spent on a solution. A TEX based system comes for free and there are no license costs. Whatever alternative is taken, even if it is as flexible, it will involve additional costs. From the perspective of costs it's very hard to beat TEX. Add to that the possibility for custom extensions, long term usage and the fact that one can adapt the system. The main question of course is: does it do the job. The only way to find out is to either experiment (which is free), consult an expert (not free, but then needed anyway for any solution) or ask an expert to make a proof of concept (also not free but relatively cheap and definitely cheaper than a failure). In fact, before making decisions about what solution is best it might be a good idea to check with an expert anyway, because more or less than one thinks might be possible. Also, take into account that the TEX ecosystem is often one of the first to support new technologies, and normally does that within its existing interface. And there is plenty of free support and knowledge available once you know how to find it. Instead of wasting time and money on advertisement and fancy websites, effort goes into support and development. Even if you doubt that the current provider is around in the decade to come, you can be sure that there will be others, simply because TEX attracts people. Okay, it doesn't help that large companies like to out source to far–far–away and expect support around the corner, so in the end they might kill their support chain.

When talking of TEX used in organizations we tend to think of publishers. But this is only a small subset of organizations where information gets transformed into something presentable. For small organizations the choice for TEX can be easy: costs, long term stability, knowing some experts are driving forces. For large organizations these factors seem (at least to us) hardly relevant. We've (had) projects where actually the choice for using a TEX based solution was (in retrospect) a negative one: there was no other tool than this relatively unknown thing called TEX. Or, because the normal tools could not be used, one ended up with a solution where (behind the scenes) TEX is used, without the organization knowing it. Or, it happened that the problem at hand was mostly one that demands in-depth knowledge of manipulating content, cleaning up messy data, combining resources (images or PDF documents), all things that happen to be available in the perspective of TEX. If you can solve a hard to solve problem for them then an organization doesn't care what tool you use. What does matter is that the solution runs forever, that costs are controllable and above all, that it "Just works." And if you can make it work fast, that helps too. We can safely claim that when TEX is evaluated as being a good option, that in the end it always works out quite well.

Among arguments that (large) organizations like to use against a choice for TEX (or something comparable) are the size of the company that they buy their solution from,

the expected availability for support, and the wide-spread usage of the tool at hand. One can wonder if it also matters that many vendors change ownership, change products every few years, change license conditions when they like, charge a lot for support or just abort a tool chain. Unfortunately when that happens those responsible for choosing such a system can have moved on to another job, so this is seldom part of an evaluation. For the supplier the other side of the table is just as much of a gamble. In that respect, an organization that wants to use an open source (and/or free) solution should realize that getting a return on investment on such a development is pretty hard to achieve. So, who really takes the risk for writing open source?

For us, the reason to develop ConTeXt and make it open is that it fits in our philosophy and we like the community. It is actually not really giving us an advantage commercially: it costs way more to develop, support and keep up-to-date than it will ever return. We can come up with better, faster and easier solutions and in the end we pay the price because it takes less time to cook up styles. So there is some back slash involved because commercially a difficult solution leads to more billable hours. Luckily we tend to avoid wasting time so we improve when possible and then it ends up in the distributed code. And, once the solution is there, anyone can use it. Basically also for us it's just a tool, like the operating system, editor and viewer are. So, what keeps development going is mostly the interaction with the community. This also means that a customer can't really demand functionality for free: either do it yourself, wait for it to show up, or pay for it (which seldom happens). Open source is not equivalent with "You get immediately what you want because someone out there writes the code.". There has to be a valid reason and often it's just users and meetings or just some challenge that drives it.

This being said, it is hard to convince a company to use TeX. It has to come from users in the organization. Or, what we sometimes see with publishers, it comes with an author team or acquired product line where it's the only option. Even then we seldom see transfer to other branches in the organizations. No one seems to wonder "How on earth can that XML to PDF project produce whatever output in large quantities in a short period of time" while other (past) projects failed. It probably relates to the abstraction of the process. Even among TeX users it can be that you demonstrate something with a click on a button and that many years afterwards someone present at that moment tells you that they just discovered that this or that can be done by hitting a button. I'm not claiming that TeX is the magic wand for everything but in some areas it's pretty much ahead of the pack. Go to a TeX user meeting and you will be surprised about the accumulated diverse knowledge present in the room. It's user demand that drives ConTeXt development, not commerce.

3.5 Choosing

So, where can one find information about TeX and friends? On the web one has to use the right search keys, so adding `tex` helps: `context tex` or `xml tex pdf` and so on. Can one make a fancy hip website, sure, but it being a life-long, already old and mature

environment, and given that it comes for free, or is used low-budget, not much effort and money can be spent on advertising it. A benefit is that no false promises and hypes are made either. If you want to know more, just ask the right folks.

For all kind of topics one can find interesting videos and blogs. One can subscribe to channels on YouTube or join forums. Unfortunately not that many bloggers or vloggers or podcasters come up with original material every time, and often one starts to recognize patterns and will get boring by repetition of wisdom and arguments. The same is true for manuals. Is a ten year old manual really obsolete? Should we just recompile it to fake an update while in fact there has been no need for it? Should we post twenty similar presentations while one can do? (If one already wants to present the same topic twenty times in the first place?) Maybe one should compare TEX with cars: they became better over time and can last for decades. And no new user manual is needed.

As with blogs and vlogs advertising TEX carries the danger for triggering political discussions and drawing people into discussions that are not pleasant: TEX versus some word processor, open versus closed source, free versus paid software, this versus that operating system, editor such or editor so.

To summarize, it's not that trivial to come up with interesting information about TEX, unless one goes into details that are beyond the average user. And those who are involved are often involved for a long time so it gets more complex over time. User group journals that started with tutorials later on became expert platforms. This is a side effect of being an old and long-term toolkit. If you run into it, and wonder if it can serve your purpose, just ask an expert.

Most TEX solutions are open source and come for free as well. Of course if you want a specific solution or want support beyond what is offered on mailing lists and forums you should be willing to pay for the hours spent. For a professional publisher (of whatever kind) this is not a problem, if only because any other solution also will cost something. It is hard to come up with a general estimate. A popular measure of typesetting costs is the price per page, which can range from a couple of euro's per page to two digit numbers. We've heard of cases where initial setup costs were charged. If not much manual intervention is needed a TEX solution mostly concerns initial costs.

Let's return to the main question "Why use TEX?" in which you can replace TEX by one of the macro packages build on top of it, for instance CONTEXT. If an (somewhat older) organization considers using TEX it should also ask itself, why it wasn't considered long ago already? For sure there have been developments in TEX engines (in CONTEXT we use LUATEX) as well as possibilities of macro packages but if you look at the documents produced with them, there is not that much difference with decades ago. Processing has become faster, some things have become easier, but new technologies have always been supported as soon at they showed up. Advertising is often just repeating an old message.

The TEX ecosystem was among the first in supporting for instance OPENTYPE, and the community even made sure that there were free fonts available. A format like PDF was

supported as soon as it showed up and TEX was the first to demonstrate what advanced features were there and it shows again how it is possible to adapt TEX to changes in its environment. Processing XML using TEX has never been a big deal and if that is a reason to look at this already old and mature technology, then an organization can wonder if years and opportunities (for instance for publishing on demand or easy updating of manuals) have been lost. Of course there are (and have been) alternative tools but the arguments for using TEX or not are not much different now. It can be bad marketing of open and free software. It can be that TEX has been around too long. It can also be that its message was not understood yet. On the other hand, in software development it's quite common to reinvent wheels and present old as new. It's never too late to catch on.

What's to stay, what's to go 4

4.1 Introduction

The following text was written as preparation for a 2018 talk at BachoTEX, which has this theme. It's mostly a collection of thoughts. It was also more meant as a wrapup for the presentation (possibly with some discussions) than an article.

4.2 Attraction

There are those movies where some whiz-kid sits down behind a computer, keys in a few commands, and miracles happen. Ten fingers are used to generate programs that work immediately. It's no problem to bypass firewalls. There is no lag over network connections. Checking massive databases is no big deal and there's even processing power left for real time visualization or long logs to the terminal.

How boring and old fashioned must a regular edit–run–preview cycle look compared to this. If we take this 2018 movie reality as reference, in a time when one can suck a phone empty with a simple connection, pull a hard drive from a raid five array and still get all data immediately available, when we can follow realtime whoever we want using cameras spread over the country, it's pretty clear that this relatively slow page production engine TEX has no chance to survive, unless we want to impress computer illiterate friends with a log flying by on the console (which in fact is used in movies to impress as well).

On YouTube you can find these (a few hours) sessions where Jacob Collier harmonizes live in one of these Digital Audio Workstation programs. A while later on another channel June Lee will transcribe these masterpieces into complex sheets of music by ear. Or you can watch the weekly Wintergatan episodes on building the Marble Machine from wood using drilling, milling, drawing programs etc. There are impressive videos of multi-dimensional led arrays made by hand and controlled by small computers and robots that solve Rubic Cubes. You can be impressed by these Animusic videos, musicians show their craftmanship and interesting informative movies are all over the place. I simply cannot imagine millions of kids watching a TEX style being written in a few hours. It's a real challenge for an attention span. I hope to be proven wrong but I fear that for the upcoming generation it's probably already too late because the 'whow' factor of TEX is low at first encounter. Although: picking up one of Don Knuths books can have that effect: a nice mixture of code, typesetting and subtle graphics, combined with great care, only possible with a system like TEX.

: Biology teaches us that 'cool' is not a recipe for 'survival'. Not all designs by nature look cool, and it's only efficiency and functionality that matters. Beauty sometimes matters too but many functional mechanisms can do without. So far

TEX and its friends were quite capable to survive so there must be something in it that prevents it to be discarded. But survival is hard to explain. So far TEX just stayed around but lack of visual attraction is a missing competitive trait.

4.3 Satisfaction

Biology also teaches us that chemistry can overload reason. When we go for short-term pleasure instead of long-term satisfaction (Google for Simon Sinek on this topic), addiction kicks in (for instance driven by crossing the dopamine thresholds too often, Google for Robert Sapolsky). Cool might relate more to pleasure while satisfaction relates to an effort. Using TEX is not that cool and often takes an effort. But the results can be very satisfying. Where 'cool' is rewarding in the short term, 'satisfaction' is more a long term effect. So, you probably get the best (experience) out of TEX by using it a lifetime. That's why we see so many old TEXies here: many like the rewards.

If we want to draw new users we run into the problem that humans are not that good in long term visions. This means that we cannot rely on showing cool (and easy) features but must make sure that the long term reward is clear. We can try to be 'cool' to draw in new users, but it will not be the reason they stay. Instant success is important for kids who have to make a report for school, and a few days "getting acquainted with a program" doesn't fit in. It's hard to make kids addicted to TEX (which could be a dubious objective).

: As long as the narrative of satisfaction can be told we will see new users. Meetings like BachoTEX is where the narrative gets told. What will happen when we no longer meet?

4.4 Survival

Survival relates to improvements, stability and discarding of weak aspects. Unfortunately that does not work out well in practice. Fully automated multi–columns typesetting with all other elements done well too (we just mention images) is hard and close to impossible for arbitrary cases, so nature would have gotten rid of it. Ligatures can be a pain especially when the language is not tagged and some kind of intelligence is needed to selectively disable them. They are the tail of the peacock: not that handy but meant to be impressive. Somehow it stayed around in automated typesetting, in biology it would be called a freak of nature: probably a goodbye in wildlife. And how about page breaks on an electronic device: getting rid of them would make the floating figures go away and remove boundary conditions often imposed. It would also make widows and clubs less of a problem. One can even wonder if with page breaks the windows and clubs are the biggest problems, and if one can simply live with them. After all, we can live with our own bodily limitations too. After all, (depending on what country you live in) you can also live with bad roads, bad weather, polution, taxes, lack of healthcare for many, too much sugar in food, and more.

: Animals or plants that can adapt to live on a specific island might not survive elsewhere. Animals or plants introduced in an isolated environment might quickly dominate and wipe out the locals. What are the equivalents in our TEX ecosystem?

4.5 Niches

But arguments will not help us determine if TEX is the fittest for survival. It's not a rational thing. Humans are bad in applying statistics in their live, and looking far ahead is not a treat needed to survive. Often nature acts in retrospect. (Climbing mount probability by Richard Dawkins). So, it doesn't matter if we save time in the future if it complicates the current job. If governments and companies cannot look ahead and act accordingly, how can we extrapolate software (usage) or more specifically typesetting demands. Just look at the political developments in the country that hosts this conference. Could we have predicted the diminishing popularity of the EU (and disturbing retrograde political mess in some countries) of 2018 when we celebrated the moment Poland joining the EU at a BachoTEX campfire?

Extrapolating the future quality of versions of TEX or macro packages also doesn't matter much. With machine learning and artificial intelligence around the corner and with unavoidable new interfaces that hook into our brains, who knows what systems we need in the future. A generic flexible typesetting system is probably not the most important tool then. When we discuss quality and design it gets personal so a learning system that renders neutrally coded content into a form that suits an individual, demands a different kind of tool than we have now.

On the short term (our live span) it makes more sense to look around and see how other software (ecosystems) fare. Maybe we can predict TEX's future from that. Maybe we can learn from others mistakes. In the meantime we should not flatter ourselves with the idea that a near perfect typesetting system will draw attention and be used by a large audience. Factors external to the community play a too important role in this.

: It all depends on how well it fits into a niche. Sometimes survival is only possible by staying low on the radar. But just as we destroy nature and kill animals competing for space, programs get driven out of the software world. On a positive note: in a project that provides open (free) math for schools students expressed to favour a printed book over WEB-only (one curious argument for WEB was that it permits easier listening to music at the same time).

4.6 Dominance

Last year I installed a bit clever (evohome) heating control system. It's probably the only "working out of the box" system that supports 12 zones but at the same time it has a rather closed interface as any other. One can tweak a bit via a web interface but that one works by a proxy outside so there is a lock in. Such a system is a gamble because

it's closed and we're talking of a 20 year investment. I was able to add a layer of control (abusing LuaTeX as Lua engine and ConTeXt as library) so let's see. When I updated the boiler I also reconfigured some components (like valves) and was surprised how limited upgrading was supported. One ends up with lost settings and weird interference and it's because I know a bit of programming that I kept going and managed to add more control. Of course, after a few weeks I had to check a few things in the manuals, like how to enter the right menu.

So, as the original manuals are stored somewhere, one picks up the smart phone and looks for the manual on the web. I have no problem with proper PDF as a manual but why not provide a simple standard format document alongside the fancy folded A3 one. Is it because it's hard to produce different instances from one source? Is it because it takes effort? We're talking of a product that doesn't change for years.

: The availability of flexible tools for producing manuals doesn't mean that they are used as such. They don't support the survival of tools. Bad examples are a threat. Dominant species win.

4.7 Extinction

When I was writing this I happened to visit a bookshop where I always check the SciFi section for new publications. I picked out a pocket and wondered if I had the wrong glasses on. The text was wobbling and looked kind of weird. On close inspection indeed the characters were kind of randomly dancing on the baseline and looked like some 150 DPI (at most) scan. (By the way, I checked this the next time I was there by showing the book to a nephew.) I get the idea that quite some books get published first in the (more expensive) larger formats, so normally I wait till a pocket size shows up (which can take a year) so maybe here I had to do with a scan of a larger print scaled down.

What does that tell us? First of all that the publisher doesn't care about the reader: this book is just unreadable. Second, it demonstrates that the printer didn't ask for the original PDF file and then scaled down the outline copy. It really doesn't matter in this case if you use some high quality typesetting program then. It's also a waste of time to talk to such publishers about quality typesetting. The printer probably didn't bother to ask for a PDF file that could be scaled down.

: In the end most of the publishing industry will die and this is just one of the symptoms. Typesetting as we know it might fade away.

4.8 Desinterest

The newspaper that I read has a good reputation for design. But why do they need to drastically change the layout and font setup every few years? Maybe like an animal marking his or her territory a new department head also has to put a mark on the layout. Who knows. For me the paper became pretty hard to read: a too light font that suits

none of the several glasses that I have. So yes, I spend less time reading the paper. In a recent commentary about the 75 year history of the paper there was a remark about the introduction of a modern look a few decades ago by using a sans serif font. I'm not sure why sans is considered modern (most handwriting is sans) and to me some of these sans fonts look pretty old fashioned compared to a modern elegant serif (or mix).

: If marketing and fashion of the day dominate then a wrong decision can result in dying pretty fast.

4.9 Persistence

Around the turn of the century I had to replace my CD player and realized that it made more sense to invest in ripping the CD's to FLAC files and use a decent DAC to render the sound. This is a generic approach similar to processing documents with TEX and it looks as future proof as well. So, I installed a virtual machine running SlimServer and bought a few SlimDevices, although by that time they were already called SqueezeBoxes.

What started as an independent supplier of hardware and an open source program had gone the (nowadays rather predictable) route of a buy out by a larger company (Logitech). That company later ditched the system, even if it had a decent share of users. This "start something interesting and rely on dedicated users", then "sell yourself (to the highest bidder)" and a bit later "accept that the product gets abandoned" is where open source can fail in many aspects: loyal users are ignored and offended with the original author basically not caring about it. The only good thing is that because the software is open source there can be a follow up, but of course that requires that there are users able to program.

I have 5 small boxes and a larger transporter so my setup is for now safe from extinction. And I can run the server on any (old) LINUX or MS WINDOWS distribution. For the record, when I recently connected the 20 year old Cambridge CD2 I was surprised how well it sounded on my current headphones. The only drawback was that it needs 10 minutes for the transport to warm up and get working.

In a similar fashion I can still use TEX, even when we originally started using it with the only viable quality DVI to POSTSCRIPT backend at that time (DVIPSONE). But I'm not so sure what I'd done if I had not been involved in the development of PDFTEX and later LUATEX. As an average user I might just have dropped out. As with the CD player, maybe someone will dust off an old TEX some day and maybe the only hurdle is to get it running on a virtual retro machine. Although ... recently I ran into an issue with a virtual machine that didn't provide a console after a KVM host update, so I'm also getting pessimistic about that escape for older programs. (Not seldom when a library update is forced into the LUATEX repository we face some issue and it's not something the average user want (or is able to) cope with.)

: Sometimes it's hard to go extinct, even when commerce interfered at some point. But it does happen that users successfully take (back) control.

4.10 Freedom

If you buy a book originating in academia written and typeset by the author, there is a chance that it is produced by some flavour of TEX and looks quite okay. This is because the author could iterate to the product she or he likes. Unfortunately the web is also a source of bad looking documents produced by TEX. Even worse is that many authors don't even bother to set up a document layout properly, think about structure and choose a font setup that matches well. One can argue that only content matters. Fine, but than also one shouldn't claim quality simply because TEX has been used.

I've seen examples of material meant for bachelor students that made me pretend that I am not familiar with TEX and cannot be held responsible. Letter based layouts on A4 paper, or worse, meant for display (or e-book devices) without bothering to remove the excessive margins. Then these students are forced to use some collaborative TEX environment, which makes them dependent on the quality standards of fellow students. No wonder that one then sees dozens of packages being loaded, abundant copy and paste and replace of already entered formulas and interesting mixtures of inline and display math, skips, kerns and whatever can help to make the result look horrible.

: Don't expect enthusiast new users when you impose TEX but take away freedom and force folks to cooperate with those with lesser standards. It will not help quality TEX to stay around. You cannot enforce survival, it just happens or not, probably better with no competition or with a competition so powerful that it doesn't bother with the niches. In fact, keeping a low profile might be best! The number of users is no indication of quality, although one can abuse that statistic selectively?

4.11 Diversity

Diversity in nature is enormous. There are or course niches, but in general there are multiple variants of the same. When humans started breeding stock or companion animals diversity also was a property. No one is forcing the same dog upon everyone or the same cow. However, when industrialization kicks in things become worse. Many cows in our country share the same dad. And when we look at for instance corn, tomatoes or whatever dominance is not dictated by what nature figures out best, but by what commercially makes most sense, even if that means that something can't reproduce by itself any longer.

In a similar way the diversity of methods and devices to communicate (on paper) at some point turns into commercial uniformity. The diversity is simply very small, also in typesetting. And even worse, a user even has to defend her/himself for a choice of system (even in the TEX community). It's just against nature.

: Normally something stays around till it no longer can survive. However, we humans have a tendency to destroy and commerce is helping a hand here. In that respect it's a surprise that TEX is still around. On the other hand, humans also

have a tendency to keep things artificially alive and even revive. Can we revive TEX in a few hundred years given the complex code base and Make infrastructure?

4.12 Publishing

What will happen with publishing? In the production notes of some of my recently bought books the author mentions that the first prints were self-published (either or not sponsored). This means that when a publisher "takes over" (which still happens when one scales up) not much work has to be done. Basically the only thing an author needs is a distribution network. My personal experience with for instance CD's produced by a group of musicians is that it is often hard to get it from abroad (if at all) simply because one needs a payment channel and mail costs are also relatively high.

But both demonstrate that given good facilitating options it is unlikely that publishers as we have now have not much change of survival. Add to the argument that while in Gutenbergs time a publisher also was involved in the technology, today nothing innovative comes from publishers: the internet, ebook devices, programs, etc. all come from elsewhere. And I get the impression that even in picking up on technology publishers lag behind and mostly just react. Even arguments like added value in terms of peer review are disappearing with the internet where peer groups can take over that task. Huge amounts of money are wasted on short-term modern media. (I bet similar amounts were never spend on typesetting.)

: Publishers, publishing, publications and their public: as they are now they might not stay around. Lack of long term vision and ideas and decoupling of technology can make sure of that. Publishing will stay but anyone can publish; we only need the infrastructure. Creativity can win over greed and exploitation, small can win over big. And tools like TEX can thrive in there, as it already does on a small scale.

4.13 Understanding

"Why do you use TEX?" If we limit this question to typesetting, you can think of "Why don't you use MS WORD?" "Why don't you use Indesign?", "Why don't you use that macro package?", "Why don't you use this TEX engine?" and alike. I'm sure that most of the readers had to answer questions like this, questions that sort of assume that you're not happy with what you use now, or maybe even suggest that you must be stupid not to use . . .

It's not that easy to explain why I use TEX and/or why TEX is good a the job. If you are in a one-to-one (or few) sessions you can demonstrate its virtues but 'selling' it to for instance a publisher is close to impossible because this kind of technology is rather unknown and far from the click-and-point paradigm. It's even harder when students get accustomed to these interactive books from wherein they can even run code snippets

although one can wonder how individual these are when a student has the web as a source of solutions. Only after a long exposure to similar and maybe imperfect alternatives books will get appreciated.

For instance speaking of "automated typesetting" assumes that one knows what typesetting is and also is aware that automated has some benefits. A simple "it's an XML to PDF converter" might work better but that assumes XML being used which for instance not always makes sense. And while hyphenation, fancy font support and proper justification might impress a TEX user it often is less of an argument than one thinks.

The "Why don't you" also can be heard in the TEX community. In the worst case it's accompanied by a "... because everybody uses ..." which of course makes no sense because you can bet that the same user will not fall for that argument when it comes to using an operating system or so. Also from outside the community there is pressure to use something else: one can find defense of minimal markup over TEX markup or even HTML markup as better alternative for dissemination than for instance PDF or TEX sources. The problem here is that old–timers can reflect on how relatively wonderful a current technique really is, given changes over time, but who wants to listen to an old-timer. Progress is needed and stimulating (which doesn't mean that all old technology is obsolete). When I watched Endre eNerd's "The Time Capsule" blu-ray I noticed an Ensoniq Fizmo keyboard and looked up what it was. I ended up in interesting reads where the bottom line was "Either you get it or you don't". Reading the threads rang a bell. As with TEX, you cannot decide after a quick test or even a few hours if you (get the concept and) like it or not: you need days, weeks, or maybe even months, and some actually never really get it after years.

: It is good to wonder why you use some program but what gets used by others depends on understanding. If we can't explain the benefits there is no future for TEX. Or more exact: if it no longer provide benefits, it will just disappear. Just walk around a gallery in a science museum that deals with computers: it can be a bit pathetic experience.

Who knows ...

Stability 5

Introduction

How stable is ConTeXt? This question is hard to answer. For instance MkII hasn't changed for years and seems to work quite well: no changes equals stability. Those who use it can do with what it offers. The potentially sensitive dependencies on for instance fonts are probably absent because there is not much development in the 8 bit fonts arena. As long as these are available we're okay, in fact, OpenType fonts are more a moving target and therefore less stable.

What do we mean by stable? The fundamental differences between an 8 bit engine (and fonts) and an Unicode aware engine able to handle OpenType fonts is substantial which is why we dropped some functionality and added some relevant new. One can consider that a problem but in practice using fonts has become easier so no one is hurt by it. Here we need to keep in mind that pdfTeX is really stable: it uses fonts and technology that doesn't change. On the other hand XeTeX and LuaTeX follow new trends. Thereby XeTeX uses libraries, which introduces a dependency and instability, while LuaTeX assumes solutions in Lua which means that users and macro writers can tweak and thereby also introduce instability (but at least one can adapt that code).

Due to the way the user interface is set up, it is unlikely that ConTeXt will change. But the fact that we now have Lua available means that many commands have been touched. Most behave compatible, some have more functionality, and of course we have a Lua interface. We include a lot of support code which also lessens dependencies.

The user input is normally TeX but when you use xml the move to MkIV meant that we dropped the MkII way of dealing with it in favour of a completely new mechanism. I get the impression that those using xml don't regret that change. Talking of stability the MkIV xml interface is typically a mechanism that is stable and might change little. We can add new trickery but the old stays as it is.

If we look at the output, there is dvi and pdf. In MkII the dvi could become PostScript. As there are different dvi post-processors the backend code was using a plug-in model. Contrary to other macro packages there was only one so called format that could adapt itself to the required (engine specific) output. A ConTeXt run has always been managed by a wrapper so users were not bothered much by what TeX engine they used and/or what backend was triggered. This changed with MkIV where we use just LuaTeX, always produce pdf and optionally can export xml. But again the run is managed by a wrapper, which incidentally is written in Lua and thereby avoids dependencies on for instance Perl, Ruby or Python, which are moving targets, use libraries and additional user code, and thereby are potentially instable too.

The PDF code that is produced is a mix of what the engine spits out and what the macro package injects. The code is normally rather simple. This means that it's no big deal to support the so called standards. It also means that we can support advanced interactivity and other features but these also depends on the viewers used. So, stability here is more fluent, for instance because the PDF standard evolves and/or we need to adapt to viewers. Special demands like tagged PDF have been supported right from the start but how that evolves depends mostly on input from users who need it. Again, that is less important (and crucial) for stability than the rendering capabilities.

The fact that we use LUA creates a dependency on that language but the reason that we use it is *because* it is so stable. We follow the updates and so far that worked out well. Now, say that we had a frozen version of CONTEXT 2010 and LUATEX 1.09 that uses LUA 5.3, would that work? First of all, in 2010 LUATEX itself was evolving so the answer is probably "no", unless one adds a few compatibility patches. I'm not going to try it. The change from 5.1 to 5.2 to 5.3 was not really a problem I think and the few issues could be dealt with easily. If you want long term stability and use a lot of LUA code you can take it into account when coding. Avoiding external libraries is a good start.

Fonts are more than before moving targets. So, if you want stability there you should save them with your document source. The processing of them has evolved and has been improved over time. By now it's rather stable. More recent code can catch more issues and fixes are relatively easy. But it's an area that you always need to check when you update an old distribution. The same is true for language related hyphenation patterns and script specific support. The community is no longer leading in the math department either (OPENTYPE math is a MICROSOFT invention). But, the good news is that the TEX ecosystem is always fast to adapt and can also often provide more functionality.

Vertical spacing, in fact spacing in general is an aera that can always be improved, so there is where you can expect changed. The same is true for side floats or mechanisms where content is somehow attached to other moving content, for instance marginal notes.

But code dealing with fonts, color, scripts, structure, and specific features that once written don't need more, will not change that much. As mentioned for fonts, like any resource, we also depend on third parties. Colors can relate to standards, but their main properties are unchanged. Support for specific scripts can (and will) be improved due to user input and demands so there the users also influence stability. Structure doesn't really influence the overall rendering, but the way you set it up does, but that's user styling. Of course during the transition from MKII to MKIV and the evolution of LUATEX things could be broken, but fixing something structural seldom relates to rendering. If for instance we improve the interpretation of BIBTEX input , which can be real messy, that involves data processing, nor rendering. When we improve support for the APA standard, which is complex, it might involve rendering but then that's asked for and expected. One cannot do better than the input permits.

Publishers

When discussing stability and especially stability as requirement we need to look at the way ConTeXt is used. So let's look at a few scenarios. Say that a publisher gets a camera ready book from an author in PDF format. In that case the author can do all tweaks needed. Now say that the publisher also wants the source code in a format that makes reuse possible.

But let's face reality. Will that publisher really reformat the document in PDF again? It's very unlikely. First of all the original PDF can be kept, and second, a reformat only makes sense after updating the content or going for a completely different layout. It's basically a new book then. In that case literal similarity of output is irrelevant. It is a cheap demand without much substance.

When the source is used for a different purpose the tool used to make the PDF is irrelevant. In that case the coding of the source can matter. If it is in some dialect of TeX, fine, one has to convert it anyway (to suit the other usage). If there is an XML export available, fine too as it can be transformed, given that the structure is rich enough, something that is unlikely to have been checked when the original was archived. Then there could have been the demand for a document in some other format and who can guarantee stability of the tools used there? Just look at how Microsoft Word evolved, or for that matter, its competitors. On the average TeX is more stable as one can snapshot a TeX tree and run binaries for years, if needed, in a virtual machine.

So, I don't think that a publisher is of any relevance in the discussion about stability. Even if we can clearly define what a publisher is, I doubt if publishers themselves can be considered long term stable organizations. Not today. I'm not sure if (especially the large) publishers really deserve a place in the discussion about stability but I'm willing to discuss that when I run into one.

The main problem that an author can face when being confronted with the stability issue this way is that the times are long gone that publishers have a clue about what TeX is, how it evolved and how it always had to and did adapt to changing requirements. If you're lucky you will run into someone who does know all this. They're normally a bit older and have seen the organization from any angles and therefore are fun to work with.

But even then, rendering issues are often not high on their agenda. Outsourcing often has become the modus operandi which basically brings us to the second group involved in this discussion: suppliers.

Suppliers

I don't know many suppliers other than the ones we ran into over a few decades. At least where I live the departments that are responsible for outsourcing typesetting like to deal with only a few large suppliers, interestingly because they assume that they are

stable. However, in my experience hardly any of those seem to have survived. (Of course one can wonder if long term commitment really is that important in a world where companies change so fast.) This is somewhat obscured by the fact that publishers themselves merge, reorganize, move people around, etc. so who can check on the stability of suppliers. It is definitely a fact that at least recently hardly any of them played a rol of any relevance in the development of stable tools. In the past the membership of TEX user groups contained people working at publishers and suppliers but that has changed.

Let's focus on the suppliers that somehow use TEX and let's consider two kind of suppliers: small ones, one were only a few people work, and large ones. The small ones depend on stable TEX distributions, like TEXLive where they can get the resources from: styles, fonts, patterns, binaries. If they get the authors TEX files they need to have that access. They have to rework that input into what the customer demands and that likely involves tweaks. So, maybe they have developed their own additional code. For that code, stability is their own responsibility. Did they tweak core code of a macro package? Fine, but you might have it coming when you update. You cannot expect the evolving free meal world to stick to your commercial needs. A supplier can play safe and somehow involve the developers of macro packages or consult them occasionally, but does that really happen often? Interesting is that a few times that I was asked for input it was also wrapped in obscurity, as if some holy grail of styling was involved, while it's quite likely that the developer of a macro package can write such a style (or extra code) easily and probably also better. There really is not that much unique code around.

Small suppliers can be on mailing lists where they can contribute, get feedback, provide testing, etc. They are part of a process and as such have some influence on stability. If they charge by the page, then a change in their tools can be reflected in what they charge. Basically redoing a book (or so) after a decade is doing a new job. And adapting to some new options in a package, as part of a typesetting job is probably no big deal. Is commercial really more stable than open source free software? Probably not, except from open source software developers whose real objective is to eventually sell their stuff to some company (and cash) and even accept it to be ditched. Small suppliers are more flexible.

The large suppliers are a different group. They often guard their secrets and stay in the dark. They probably seldom share (fundamental) code and information. If they are present in a community it can be for marketing reasons. If at some point a large supplier would demand stability, then my first response would be: sure I can make you a stable setup and maybe even provide intermediate patches but put your money where your mouth is. But that never happened and I've come to the conclusion that we can safely ignore that group. The TEX user groups create distributions and have for instance funded font development and it are the common users who paid for that, not the scale ones. To some extent this is actually good because large (software related) organizations often have special agendas that can contradict what we aim at in the long term.

From the authors perspective there is a dilemma here. When you submit to a publisher who outsources, it can be a demand to deliver in a specific TEX format. Often a PDF comes with the source then, so that the intended rendering is known. Then that source goes to a supplier who then (quite likely) redoes a lot of the coding in some stable subset, maybe even in a very old version of the macro package. If I were such an author I'd render the document in 'as stupid as possible mode' because you gain nothing by spending time on the looks. So, stability within the package that you use is easy and translation from one to another probably also. It's best to check beforehand what will happen with your source and let stability, if mentioned, be their problem. After all they get paid for it.

Suppliers seldom know CONTEXT. An interesting question is if they really know the alternatives well, apart from the bit they use. A well structured CONTEXT source (or probably any source) is often easy to convert to another format. You can assume that a supplier has tools for that (although we're often surprised about the poor quality of tools used). Often the strict demand for some kind of format is an excuse for lack of knowledge. Unfortunately you need a large author base to change that attitude.

Authors

Before we move to some variants of the above, first I will look at stability from the authors perspective. When a book is being written the typesetting more or less happens as part of the process. The way it looks can influence the way you write and vise versa. Once the book is done it can go in print and, unless you were using beta versions of CONTEXT and updated frequently. Normally you will try to work in a stable setup. Of course when a user asks for additional features while working on a project, he or she should also accept other beta features and side effects.

After a few years an author might decide to update the book. The worst that can happen is that the code doesn't run with the latest CONTEXT. This is not so likely because commands are upward compatible. However, the text might come out a bit different, for instance because different fonts or patterns are used. But on the average paragraphs will come out the same in TEX. You can encounter differences in the vertical spacing and page breaks, because that is where improvements are still possible. If you use conceptually and implementation wise complex mechanism like side floats, you can also run into compatibility issues. But all these don't really matter much because the text will be updated anyway and fine-tuning of page breaks (if at all) happens at the end. The more you try to compete with desk top publishing, and the more tweaks you apply, the greater is the risk that you introduce instability. It is okay for a one-time job, but when you come back to it after a decade, be prepared for surprises.

Even if you stick to the original coding, it makes sense to sacrifice some of that stability if new mechanisms have become available. For instance, if you use METAPOST, better ways to solve your problem might have become available. Or if you document is 15 years old, a move from MKII to MKIV is a valid option, in which case you might also consider using the latest fonts.

Of course, when you made a style where you patched core code, you can expect problems, because anything not explicitly mentioned in the interface definition files is subjected to change. But you probably see that coming anyway.

So, is an author (or stand alone user) really dependent on stability? Probably less than thought. In fact, the operating system, internet and browsers, additional tools: all change over time and one adapts. It's something one can live with. Just see how people adapt to phones, tablets, social media, electric cars, etc. As long as the document processes and reasonable output is generated it's fine. And that is always what we aim at! After all we need to be able to use it ourselves, don't we?

Projects

Although it is often overlooked as valid alternative in rendering in large scale projects, ConTeXt is perfect as component in a larger whole. Something goes in, something comes out. In a long term project one can just install a minimal distribution, write styles, and run it for ages. Use a virtual machine and we're talking decades without any change. And, when one updates, it's easy to check if all still works. Often the demands and styles are simple and predictable. It's way more likely that a hard coded solution in some large programming environment has stability issues than that the ConTeXt bit has.

If ConTeXt is used in for instance documentation of (say) software, again there is no real issue. Such documents are simple, evolve and therefore have no stable page flow, and updating ConTeXt is not needed if the once decided upon coding is stable. You don't need the latest features. We've written styles and setups for such tasks and indeed they run for ages.

It can make me smile to see how much effort sometimes goes in low quality rendering where ConTeXt could do a way better job with far less investment in time and money but where using some presumed stable toolkit is used instead, one that comes with expensive licensing, from companies that come and go but shine in marketing. (A valid question is to what extent the quality of and care for documentation reflects the core products that a company produces, at least under the hood.)

The biggest hurdle in setting up a decent efficient workflow is that it has to be seen as a project: proper analysis, proper planning, prototyping and testing, etc. You invest first and gain later. When dealing with paper many publishers still think in price per page and have problems seeing that a stable mostly automated flow in the end can result in a ridiculous low price per page, especially in typesetting on demand.

Hybrids

Last I will mention a setup that we sometimes are involved in. An author writes books and uses TeX. The publisher is okay with that and adds some quality assurance but in the end the product comes from the author. Maybe images are oursourced (not always

for the better) but these can be handled easily. It can be that a copy-editor is involved and that person also then has to use TEX of course, or feedback to the author.

Publishers, and this really depends on knowledgeable persons, which as said can be fun to work with, can look beyond paper and also decide for additional materials, for instance web pages, interactive exercises, etc. In that case either CONTEXT input has to be available as XML (an export) or (often better) XML is the starting point for multiple output. Contrary to what is believed, there are authors out there who have no problem coding in XML directly. They think in structured content and reuse! The fact that they can hit a button in the editor and see the result in PDF helps a lot. It just works.

Here stability is either achieved by simply not updating during a project. There are however cases where an update is needed, for instance because demands changed. An example is a project where ASCIIMATH is used which is a moving target. Of course one can update just that module, and often that works, but not when a module uses some new improved core helpers. Another example is additional proofing options.

The budget of such projects seldom permit patching an existing distribution, so we then just update to the latest but not after checking if the used style works okay. There is no author involvement in this. Depending on the workflow, it can even be that the final rendering which involves fine tuning (side) float placement or page breaks (often educational documents have special demands) is done by us using special directives.

Such hybrid workflows are quite convenient for all parties. The publisher works with the author who likes using these tools, the author can do her or his thing in the preferred way, and we do what we're best in: supporting this. And it scales up pretty well too if needed, without much costs for the publishers.

Conclusion

So what can we conclude with respect to the demand for stability? First of course that it's important that our files keep running well. So, functionality should be stable. Freezing a distribution will make sure that during project you don't run into issues. Many CONTEXT users update frequently in order to benefit from the latest additions. Most will not be harmed by this, but when something really breaks it's users like those on the CONTEXT support list (who often also contribute in helping out other users) that are listened to first. Publishers demands play no role in this, if only because they also play no role in typesetting, and if they want to they should also contribute. The same is true for large suppliers. We're talking of free software often written without any compensation so these parties have no say in the matter unless they pay for it. It's small suppliers, authors and general users that matter most. If CONTEXT is part of a workflow that we support, of course stability is guaranteed quite well, and those paying for that never have an issue with better solutions popping up. In fact, CONTEXT is often just a tool then, one that does the job and questions about stability don't matter much in practice, as long as it does the job well.

The main engine we use, LuaTeX, will be quite stable from version 1.10 and we'll try to make sure that newer versions are capable of running an older ConTeXt, which is easier when no fundamental changes happen in the engine. Maybe a stripped down version of LuaTeX for ConTeXt can facilitate that objective even more.

Users themselves can try to stick to standard ConTeXt features. The more tricks you apply, the less stable your future might be. Most mechanism are not evolving but some, like those that deal with columns, might become better over time. But typesetting in columns is often a one-shot adventure anyway (and who needs columns in the future).

Of one thing users can be sure. There will never be a ConTeXt professional or ConTeXt enterprise. There is only one variant. All users get the same functionality and policies don't change suddenly. There will be no lock in to some cloud or web based service either. Of course one can hire us for support of any kind but that's independent of the distributed package. There is support by users for users on mailing lists and other media. That itself can also guard stability.

But, always keep in mind that stability and progress, either of not driven by the environment that we operate in, can be in conflict.

MetaTeX, a roadmap 6

6.1 Introduction

Here I will shortly wrap up the state of LuaTeX and ConTeXt in fall 2018. I made the first draft of this article as preparation for the ConTeXt meeting where we also discussed the future. I updated the text afterwards to match the decisions made there. It's also a personal summary of thoughts and discussions with team members about where to move next.

6.2 The state of affairs

After a dozen years the development of LuaTeX has reached a state where adding more functionality and/or opening up more of the internals makes not much sense. Apart from fixes and maybe some minor extensions, version 1.10 is what you get. Users can do enough in Lua and there is not much to gain in convenience and performance. Of course some of the code can and will be cleaned up, as we still see the effects of going from Pascal to cweb to C. In the process consistency is on the radar so we might occasionally add a helper. But we also don't want to move too far away from the original code, which is for instance why we keep names, keys and other properties found in original TeX, which in turn leads to some inconsistencies with extensions added over time. We have to accept that.

Because LuaTeX development is closely related to ConTeXt development, especially MkIV, we've also reached the moment that we can get rid of some older code and assume the latest LuaTeX to be used. Because we do so much in Lua the question is always to what extent the benefits outweigh the drawbacks. Just in case you wonder why we use Lua extensively, the main reason is that it is easier and more efficient to manage data in this language and modern typesetting needs much data. It also permits us to extend regular TeX functionality. But, one should not overrate the impact: we still let TeX do what TeX is best at!

Performance is quite important. It doesn't make sense to create a powerful typesetting system where processing a page takes a second. We have discussed performance before since one of the complaints about LuaTeX is that it is slow. A simple, basic test is this:

```
\starttext
    \dorecurse{1000}{\input tufte \par}
\stoptext
```

This involves 1000 times loading a file (and reporting that on the console, which can influence runtime), typesetting paragraphs, splitting of a page and of course loading

fonts and saving to the PDF file. When I run this on a modest machine, I get these (relative) timings for the (about) 225 pages:

TEX engine used	**PDFTEX**	**LUATEX**	**LUAJITTEX**	**XETEX**
runtime in seconds	2.0	3.9	3.0	8.4

Now, as expected the 8 bit PDFTEX is the winner here but LUATEX is not doing that bad. I don't know why XETEX is so much slower, maybe because its 64 bit binary is less optimal. I once noticed that a 64 bit PDFTEX performed worse on such a test than LUATEX, for which I always use 64 bit binaries.

If you consider that often much more is done than in this example, you can take my word that LUATEX quickly outpaces PDFTEX on more complex tasks. In that sense it is now our benchmark. It must be said that the MKIV code is probably a bit more efficient than the MKII code but that doesn't matter much in this simple test because hardly any macro magic happens here; it mostly tests basic font processing, paragraph building and page construction. I don't think that I can squeeze out more pages per second, at least not without users telling me where they encounter bottlenecks that don't result from their style coding. It's no problem to write inefficient macros (or styles) so normally a user should first carefully check her/his own work. Using a more modern CPU with proper caching and an SSD helps too.

So, to summarize, we can say that with version 1.10 LUATEX is sort of finished. Our mission is now to make LUATEX robust and stable. Things can be added and improved, but these are small and mostly consistency related.

6.3 More in LUA

Till now I always managed to add functionality to CONTEXT without hampering performance too much. Of course the biggest challenge is always in handling fonts and common features like color because that all happens in LUA. So, the question is, what if we delegate more of the core functionality to LUA? I will discuss a few options because the CONTEXT developers and users need to agree on the path to follow. One question there is, are the possible performance hits (which can be an inconvenience) compensated by better and easier typesetting.

Fonts, colors, special typesetting features like spaced kerning, protrusion, expansion, but also dropped caps, line numbering, marginal notes, tables, structure related things, floats and spacing are not open for much discussion. All the things that happen in LUA combined with macros is there and will stay. But how about hyphenation, paragraph building and page building? And how about a leaner and meaner, future safe engine?

Hyphenation is handled in the TEX core. But in CONTEXT already for years one can also use a LUA based variant. There is room for extensions and improvements there. Interesting is that performance is more or less the same, so this is an area where we might switch to the LUA method eventually. It compares to fonts, where node mode is more or less the standard and base mode the old way.

Building the paragraphs in LUA is also available in MKIV, although it needs an update. Again performance is not that bad, so when we add features not possible (or hard to do) in regular TEX, it might actually pay of to default to the par builder written in LUA.

The page builder is also doable in LUA but so far I only played a bit with a LUA based variant. I might pick up that thread. However, when we would switch to LUA there, it might have a bit of a penalty, unless we combine it with some other mechanisms which is not entirely trivial, as it would mean a diversion from the way TEX does it normally.

How about math? We could at some point do math rendering in LUA but because the core mechanism is the standard, it doesn't really makes much sense. It would also touch the soul of TEX. But, I might give it a try, just for fun, so that I can play with it a bit. It's typically something for cold and rainy days with some music in the background.

We already use LUA in the frontend: locating and reading files in TEX, XML, LUA and whatever input format. Normalization and manipulation is all active and available. The backend is also depending on LUA, like support for special PDF features and exporting to XML. The engine still handles the page stream conversion, font inclusion and object management.

The inclusion of images is also handled by the engine, although in CONTEXT we can delegate PDF inclusion to LUA. Interesting is that this has no performance hit.

With some juggling the page stream conversion can also be done in LUA, and I might move that code into the CONTEXT distribution. Here we do have a performance hit: about one second more runtime on the 14 seconds needed for the 300 page LUATEX manual and just over more than half a second on a 11 second LUAJITTEX run. The manual has lots of tables, verbatim, indices and uses color as well as a more than average number of fonts and much time is spent in LUA. So there is a price to pay there. I tried to speed that up but there is not much to gain there.

So, say that we default to LUA based hyphenation, which enables some new functionality, LUA based par building, which permits some heuristics for corner cases, and LUA based page building, which might result in more control over tricky cases. A total performance hit of some 5% is probably acceptable, especially because by that time I might have replaced my laptop and won't notice the degrade. This still fits in the normal progress and doesn't really demand a roadmap or wider acceptance. And of course we would still use the same strategies as implemented in traditional TEX as default anyway.

6.4 A more drastic move

More fundamental is the question whether we delegate more backend activity to LUA code. If we decide to handle the page stream in LUA, then the next question is, why not also delegate object management and font inclusion to LUA. Now, keep in mind that this is all very CONTEXT specific! Already for more than a decade we delegate a lot

to LUA, and also we have a rather tight control over this core functionality. This would mean that CONTEXT doesn't really need the backend code in the engine.[4]

That situation is actually not unique. For instance, already for a while we don't need the LUATEX font loader either, as loading the OPENTYPE files is done in LUA. So, we could also get rid of the font loader code. Currently some code is shared with the font inclusion in the backend but that can be isolated.

You can see a TEX engine as being made from several parts, but the core really concerns only two processes: reading, storing and expanding macros on the one hand, and converting a stream of characters into lines, paragraphs, pages etc. Fonts are mostly an abstraction: they are visible in so called glyph nodes as font identifier (a number) and character code (also a number) properties. The result, nowadays being PDF, is also an abstraction: at some point the engine converts the to be shipped out box in PDF instructions, and in our case, relatively simple ones. The backend registers which characters and fonts are used and also includes the right resources. But, the backend is not part of the core as such! It has been introduced in PDFTEX and is a so called extension.

So, what does that all mean for a future version of CONTEXT and LUATEX? It means that we can decide to follow up with a CONTEXT that does more in LUA, which means not hard coded in a binary, on the one hand, but that we can also decide to strip the engine from non-core code. But, given that LUATEX is also used in other macro packages, this would mean a different engine. We cannot say that LUATEX is stable when we also experiment with core components.

We've seen folks picking up experimental versions assuming that it is a precursor to official code. So, in order to move on we need to avoid confusion: we need to use another name. Choosing a name is always tricky but as Taco already registered the METATEX domain, and because in the CONTEXT distribution you will find references to METATEX, we will use that name for the future engine. Adding LUA to that name makes sense but then the name would become too long.

The main difference between METATEX and LUATEX would be that the former has no file lookup library, no hardcoded font loader, and no backend generator (but possibly some helpers, and these need time to evolve). We're basically back where TEX started but instead of coding these extensions in PASCAL or C we use LUA. We're also kind of back to when we first started experimenting with LUATEX in CONTEXT where test, write and rewrite were going in parallel. But, as said, we cannot impose that on a wide audience.

If we go for such a lean and mean follow up, then we can also do a more drastic cleanup of obsolete code in CONTEXT (dating from ε-TEX, PDFTEX, ALEPH, etc.). We then are sort of back to where it all started: we go back to the basics. This might mean dropping some primitives (one can define them as dummy). Of course we could generalize some of the

[4] For generic packages like TikZ we (can) provide some primitive emulators, which is rather trivial to implement.

CONTEXT code to provide the kicked out functionality but would that pay of? Probably not.

Just for the record: replacing the handling of macros, registers, grouping, etc. to LUA is not really an option as the performance hit would make a large system like CONTEXT sort of unusable: it's no option and not even considered (although I must admit that I have some experimental LUA based TEX parser code around).

It is quite likely that building METATEX from source for the moment will be an option to the build script. But we can also decide to simplify that process, which is possible because we only need one binary. But in general we can assume that one can generate METATEX and LUATEX from the same source. A first step probably is a further isolation of the backend code. The fontloader and file handling code already can be made optional.

Given that we only need one binary (it being LUATEX or METATEX) and nowadays only use OPENTYPE fonts, one can even start thinking of a mini distribution, possibly with a zipped resource tree, something we experimented with in the early days of LUATEX.

Another though I have been playing with is a better separation between low level and high level CONTEXT commands, and whether the low level layer should be more generic in nature (so that one can run specific packages on top of it instead of the whole of CONTEXT) but that might not be worth the trouble.

6.5 Interlude

If we look at the future, it's good to also look at the past. Opening up TEX the way we did has many advantages but also potential drawbacks. It works quite well in CONTEXT because we ship an integrated package. I don't think that there are many users who kick in their own callbacks. It is possible but completely up to the user to make sure things work out well. Performance hits, interference, crashes: those who interfere with the internals can sort that out themselves. I'm not sure how well that works out in other macro packages but it is a time bomb if users start doing that. Of course the documented interfaces to use LUA in CONTEXT are supported. So far I think we're not yet bitten in the tail. We keep this aspect out of the discussion.

Another important aspect is stability of the engine. Sometimes we get suggestions for changes or patches that works for a specific case but for sure will have side effects on CONTEXT. Just as we don't test LATEX side effects, LATEX users don't check CONTEXT. And we're not even talking of users who expect their code to keep working. A tight control over the source is important but cannot be we will not be around for ever. This means that at some point LUATEX should not be changed any more, even when we observe side effects we want to get rid of, because these side effects can be in use. This is another argument for a stripped down engine. The less there is to mess with, the less the mess.

6.6 Audience

So how about CONTEXT itself? Of course we can make it better. We can add more examples and more documentation. We can try to improve support. The main question for us (as developers) is who actually is our audience. From the mails coming to the CONTEXT support list it looks like a rather diverse group of users.

At TEX meetings there are often discussions about promoting TEX. I can agree on the fact that even for simple documents it makes a lot of sense to use TEX, but who will take the first hurdles? How many people really produce a lot of documents? And how many need TEX after maybe a short period of (enforced) usage at the university?

It's not trivial to recognize the possibilities and power of the LUATEX-CONTEXT combination. We never got any serious requests for support from large organizations. In fact, we do use this combination in a few projects for educational publishers, but there it's actually the authors and editors doing the work. It's seldom company policy to use tools that efficiently automate typesetting. I dare to say that publishers are not really an audience at all: they normally delegate the task. They might accept TEX documents but let them rekey or adapt far-far-away and as cheap as possible. Thinking of it, the main reason for Don Knuth for writing TEX in the first place was the ability to control the look and feel and quality. It were developments at typesetters and publishers that triggered development of TEX. It was user demand. And the success of TEX was largely due to the unique personality and competence of the author.

System integrators qualify as audience but I fear that TEX is not considered hip and modern. It doesn't seem to matter if you can demonstrate that it can do a wonderful job efficiently and relatively cheap. Also the fact that an installation can be very stable on the long run is of no importance. Maybe that audience (market place) is all about "The more we have to program and update regularly, the merrier.". Marketing TEX is difficult.

Those who render multiple products, maintain manuals, have to render many documents automatically qualify as audience. But often company policies, preferred suppliers, so called standard tools etc. are used as argument against TEX. It's a missed opportunity.

One needs a certain mindset to recognize the potential and the question is, how do we reach that audience. Drawing a roadmap for that is not easy but worth discussing. We're open for suggestions.

6.7 Conclusion

At the CONTEXT user meeting those present agreed that moving forward this way makes sense. This means that we will explore a lean and mean METATEX alongside LUATEX. There is no rush and it's all volunteer work so we will take our time for this. It boils down to some reshuffling of code so that we can remove the built-in font loader, file

handling, and probably also SYNCTEX because we can emulate that. Then the backend with its font inclusion code will be cleaned up a bit (we even discussed only supporting modern wide fonts). It's no big deal to adapt CONTEXT to this (so it can and will support both LUATEX and METATEX). Eventually the backend might go away but now we're talking years ahead. By then we can also explore the option to make METATEX start out as a LUA function call (the main control loop) and become reentrant. There will probably not be many changes to the opened up TEX kernel, but we might extend the METAPOST part a bit (some of that was discussed at the meeting) especially because it is a nice tool to visualize big data.

As with LUATEX development we will go in small steps so that we keep a working system. Of course LUATEX is always there as stable fallback. The experiments will mostly happen in the experimental branch and binaries will be generated using the compile farm on the CONTEXT garden, just as happens now. This also limits testing and exploring to the CONTEXT community so that there are no side effects for mainstream LUATEX usage.

Nowadays, instead if roadmaps, we tend to use navigational gadgets that adapt themselves to the situation. On the road by car this can mean a detour and when walking around it can be going to suggested points of interest. During the excursion at the meeting, we noticed that after the drivers (navigators) synchronized their gadget with Jano, the routes that were followed differed a bit. We saw cars in front of going a different direction and cars behind us arriving from a different direction. So, even when we talk about roadmaps, our route can be adapted to the situation.

Now here is something to think about. If you look at the TEX community you will notice that it's an aging community. User groups seem to loose members, although the CONTEXT group is currently still growing. Fortunately we see a new generation taking interest and the CONTEXT users are a pleasant mix and it makes me stay around. I see it as an 'old timers' responsibility to have TEX and its environment in a healthy state by the time I retire from it (although I have no plans in that direction). In parallel to the upcoming development I think we will also see a change in TEX use and usage. This aspect was also discussed at the meeting and for sure will get a follow up on the mailing lists and future meetings. It might as well influence the decisions we make the upcoming years. So far TEX has never failed us in it's flexibility and capacity to adapt, so let's end on that positive note.

What's in a name 7

Hans Hagen
Hasselt NL
May 2019

7.1 TEX

I sometimes wonder how much the fact that English is the language mostly used in programming environments influences the way one looks at a program. For instance, translating the names of an operating system 'windows', an image manipulation program 'photoshop' or a text editing program 'wordperfect' to Dutch makes them sound kind of silly to me. The name can influence what you buy or are willing to use. These are examples of commercial programs but there are plenty examples of such naming in the open source universe too. I write this in my own bad English so that other non-English speakers can try to do a similar exercise.

So, when I was reading an article about CPU technology called 'thread-ripper' and after a while also saw the usual talk of yet another bunch of technologies marked as 'stack' and translated that to Dutch it again made me feel somewhat puzzled about such names. From there it was a small step to wondering about programming languages, and especially the ones I use: TEX, METAPOST, and LUA.

One can even wonder to what extent the quality of programming is influenced by the names of commands and keywords. A language name 'BASIC' sounds less serious than 'C'. A meaningless 'LUA' sounds different than 'PYTHON'. Does using your native tongue make a difference? In Dutch and German words tend to get long. When I look at my French dictionary it is rather thin, but we might need accented characters. Words in a language like Polish can differ per usage. What if German or Spanish had been chosen as the language for what is now the United States? How would we perceive programming and what would look natural to us?

7.2 TEX

The TEX language comes with a lot of so called primitives built in. Many of these relate to concepts in the program. For instance, a movement in horizontal or vertical direction that can stretch or shrink depending on what the boundary conditions demand, is called 'glue'. When discussing this in Dutch the word 'lijm' can be used and after seeing it a few times it might sound ok. We can probably use 'elastiek' ('elastic').

This internal concept is actually represented to the user via the interface name 'skip', take: `\abovedisplayskip` and `\belowdisplayskip`. Here the word 'display' refers

to math that gets vertical space around it and is normally typeset in a somewhat larger way compared to 'inline'. The word 'skip' can be translated to 'sprong' (translated back we could as well get 'jump'). But how to translate 'display'? An internet translation can be 'tentoonspreiding' but apart from it being a long word it sounds pretty weird for something math. The combined translation of such a command will not work well I think so probably complete different words has to be made up to describe these quantities. Taco suggested that `\bovenuitstallingkortesprong` might work for `\abovedisplayshortskip` but luckily no ordinary TEX user will not set such parameters in a document source.

In CONTEXT we use the somewhat typographical term `wit` or `witruimte` for vertical spacing. Some parameters like `\baselineskip` can be translated directly to the Dutch `\regelafstand` which is a proper typographical term (TEX has no concept of line height). Okay, it can become messy when we translate `\lineskip` by `\interlinespace` as that could be seen as the baseline skip too ('interlinie' comes to mind). Quite a mess. In many cases we probably would not handle the `skip` part in parameter: `leftskip` could become `\linkermarge` and `\parfillskip` can become `\paragraafuitvulling`.

Another concept is that of 'penalty', or in Dutch `boete`. It's probably harder to get the combinations right, simply because they have no typographical meanings, they're more process controllers. I fear that most translations would sound pretty weird to me. So, how do they sound to a native English speaker? Words like 'club' or 'widow' can be translated to their Dutch gender neutral counterparts 'wees' and 'weduw' but how strange does `weduwboete` sound?

The counter variables are easier. When they end on `char` that can become `karakter`. However, translating `\escapechar` with `\ontsnappingskarakter` might look a bit weird, but as that one is used very seldom, a weird one doesn't matter much. Operators like `\advance` and `\multiply` can become `\verhoog` and `\vermenigvuldig` which doesn't sound that strange in this context.

There are 'rule's and 'box'es. The first one can be translated to 'lijn' which sounds quite good. But what to do with the second one. We can use 'blok' (which translates back to 'block') which is good when we start stacking things, but also with 'doos' which is more literal but sounds to me somewhat silly: `\hdoos{whatever}`. I'm not so sure if I would have seen that in a book about TEX, I'd looked further into the language. The optional keywords 'width' etc. can be translated well into 'breedte' etc., so no problem there.

There are all kinds of very peculiar aspects that need a translation. For instance the (for new users intimidating) primitive `\futurelet`. The 'future' part is no problem as 'toekomst' isn't that weird but the 'let' will for sure become something very long in Dutch, so we end up with `\toekomstigetoekenning`, but seeing that long one, we can consider `kijkvooruit` as reasonable alternative. It definitely leads to more verbose programming.

Expansion is a tricky one. I have no clue what would make nice translations of the primitives \noexpand and \expandafter. The Dutch 'uitbreiden' simply is not sounding good here. Taco Hoekwater came up with a good alternative 'uitvouwen' for 'expand' and I like that one because we then can let bookmaker (a somewhat dubious term in itself) Willi Egger organize a workshop in unfolding (instead of folding).

Talking of 'macros' is less a problem because there is no Dutch word for it. There are more words with no real translations: \kern for instance probably would need some thinking but there might be a typographical equivalent that can be used.

The ε-TEX and LUATEX extensions introduce new names, like \detokenize, \boundary and \attribute. The first one is hard to translate because again it relates to an internal concept: tokens. I get the feeling that translating each occurrence of token by teken kind of makes everything look less serious. To strip something from its special meaning, which is actually what \detokenize does can give weird translations: \onttekenen is not really a Dutch word so a complete different one has to be found that describes what happens, like \ontwaarderen. On the other hand, \boundary and \attribute can translate directly into \grens and \attribuut where the last one sounds mostly okay.

Just to get you thinking: how would you translate \looseness (losheid, related to linebreaking), \deadcycles (\zinlozelus, in the perspective of building pages), \pretolerance (again line break related, here we can use something tolerantie) and \prevgraf (which is actually even in English a weird one but hardly used anyway, so Taco likes voorloopregels)? The easy ones are \omit, \meaning, \number, maybe even \mark. The for users often difficult to grasp 'catcode' can be simplified to 'code' which is proper Dutch. Concepts like 'align' translate well to 'uitlijnen'. Short ones like \wd could be a problem but any two letter combination can look bad, so \br could do. In the same fashion \def is ok as it is also the start of the Dutch 'definitie'. Mathematical terms like 'text', 'script' and 'scriptscript' can be confusing: 'tekst' will do but 'schrift' is strange.

Conditionals are not the hardest part: \if becomes \als, \else becomes \anders and \or is \of. However, turning \ifcase into \inhetgevaldat can be over the top. The \every... register variables can also be translated quite well, by using the \elk or \elke prefix. They are seldom seen at the user level so no real problem there.

The 'group' related commands are easy as 'groep' is a good Dutch equivalent. Even 'global' operations translate well (globaal). A dubious one is \font because we can use \lettertype but it's not really a translation. The internet translations tend towards 'fountain' kind of things.

The concept of 'discretionary' again needs a decent typographical translation although \hyphenation can become \afbreking, translating \discretionary needs some imagination. The concept of 'leaders' is again something that can best be bound to something more typographical because \leaders turned \leidinggevende is not an option nor is \leiders.

The prefix \un as used in \unhbox can become \ont so that we get \ontdoos but I get the feeling that this one can be source of jokes. The more verbose \pakdoosuit (equivalent to \unpackbox) would do better. To translate \unvcopy into the gibberish \ontdoosdecopie is simply ridiculous and \copieeruitgepaktedoos is a bit long. The \lower and \raise on the other hand translate well to \verlaag and \verhoog. Keeping \relax untranslated sounds ok to me, because \ontspan really makes a language silly.

7.3 MetaPost

The TEX language is driving a macro system while LUA is a procedural language. The METAPOST language sits somewhere in between. It is still expanding all along but it looks a bit more like a programming language with its loops, assignments, conditionals, expressions and (sort of) functions. As a consequence some of what I mentioned in the previous section applies here.

Translation of for instance truecorners into echtehoeken can give the language a bit less serious image. Words like linejoin, linecap and miterlimit relate directly to the POSTSCRIPT language so translating them also relates to translated POSTSCRIPT.

The primary, secundary keywords can be nicely translated into serious counterparts primaire and secundaire which are words that are not really of Dutch origin anyway. The precontrol and postcontrol words relate to concepts but even there the verbose controlepuntvoor and controlepuntachter could do. However punt as translation for point can be confusing because we also use that for period. Translating controls and curl needs some imagination. Words like tension becoming spanning is still acceptable soundwise. However:

```
voor i=1 stap 2 tot 10:
   .....
eindvanvoor; % or: eindvoor
```

Kind of interesting is translating if into als because fi then becomes sla which is 'lettuce' or, when see as verb, 'hit'. The true and false keywords becoming goed and fout is no problem.

Turning atleast into opzijnminst at first sight looks strange but actually I can appreciate that one. And tussendoortje as translation of interim, I can live with that one too as it sounds funny. Concepts like 'suffixes' need thinking but uitdr(ukking) or more literal expr(essie) for expr(ession) are okay. The expandafter, scantokens and similar keywords share the problem with TEX that they relate to concepts that are hard to translate.

The redpart and similar keywords could be translated into rooddeel but roodkanaal (meaning redchannel) might be better or maybe rodecomponent. As with TEX grouping related keywords are no problem.

A pencircle becomes pencirkel, odd becomes oneven, reverse becomes omgekeerd (or andersom or tegengesteld). For length we use lengte, and so on. All these sound professional enough, just like 'corner' related keywords becoming 'hoek', although there a clash with 'angle' is possible. I'm less sure about clipped becoming afgeknepen or begrensd but bounded then needs some thinking as these all are more or less the same. The concept of 'stroke' maps onto 'tekenen' or 'vegen' but lucky us that one is not really used, contrary to draw that can map onto teken, while fill and vul match well too I guess.

The transformations are no problem but I'd use a directive instead: rotated or roteer, slanted or schuin, scaled or schaal, and transform or transformeer. As you can see, these have a reasonable word length too.

The concept of a picture is known in Dutch as plaatje or tekening: not an easy choice. Using kleur for color is no problem at all. A coordinate pair becomes a paar: close enough not to give subjective side effects. The inner and outer keywords translate well to binnen and buiten but in code it might look a bit strange.

So, in general, the translated commands are not that weird but still a graphic defined in Dutch keywords instead of English to me might look less serious.

7.4 LUA

We now arrived at a more traditional programming language. The LUA language only has a few keywords. I suppose that it's just a matter of time before one gets accustomed to als ... dan ... anders ... eind instead of if ... then ... else ... end. The loops also translate rather well: zolang ... doe, herhaal .. totdat, voor ... in ... doe ... einde are all not that verbose. Also, with proper syntax highlighting they stand out and become abstract words. But because examples for kids are normally in Dutch, using a Dutch programming language might give a toy language feeling.

The local directive is a bit of a problem because it should be lokale variabele in order to sound ok in a sentence. The goto should become ganaar which is also two words with no space in between. The function keyword can become functie. A coroutine is a challenge (also conceptually); we do have routine but how about the co part?

Because LUA is such a clean language it doesn't really end up bad. In C there are some more issues due to the abbreviated struct, int, char, enum and typedef. A literal translation of void to leegte to me sounds a bit strange. What to do with unsigned? Coming up with something (short) Dutch for return is not easy either. Translating switch into schakelaar looks like a bad idea but after consulting Taco using keuze came up. The break then can be klaar which roundtrips to 'finished' and default can be anders which roundtrips to 'otherwise' which is indeed what some languages provide.. But, there are programming languages out there that have plenty keywords and that are more challenging. But as I'm a happy LUA user I don't have to worry about them.

7.5 Conclusion

Looking at a program source in Dutch the general feeling probably will be different. A low level bit of TEX is the worst. For METAPOST it's bearable and for LUA it is kind of okay. But in all cases, I'm not convinced that it would give me the same feeling. The abstraction of the language due to it not being my native tongue makes a difference. This problem is not much different than what we have with popular music and songs: for non-native speakers it's basically sounds, but for a native speaker it is more clear when nonsense is sung. The same can happen to me with movies, where watching some scandinavian series is different from watching a Dutch one. In the last case one picks up different nuances, not necessarily for the best. But it can be worse: post synchronized (audio) translations can be pretty unbearable and might compare well to programs translated to for instance Dutch. So let's not discuss the way Germans would deal with this.

7.6 Side notes

We now see monospaced fonts showing up that provide ligatures for e.g. `<=` and I've seen examples where ligatures kicked in for `fi`. One can wonder about that but ligatures are definitely something to keep in mind when translating.

The CONTEXT macro package is normally used with the English user interface. But the design is such that one can provide different ones too; after all it started out Dutch. It is beyond the scope of this musing to discuss the problems with translating typographical concepts between languages, especially when there are no distinctive words. But it can (and has) been done.

About what ConTeXt isn't 8

8.1 Introduction

It really puzzles me why, when someone someplace asks if ConTeXt is suitable for her or is his needs, there are answers like: "You need to think of ConTeXt as being kind of plain TeX: you have to define everything yourself." That answer probably stems from the fact that for LaTeX you load some style that defines a lot, which you then might need to undefine or redefine, but that's not part of the answer.

In the following sections I will go into a bit more detail of what plain TeX is and how it influences macro packages, especially ConTeXt. I'm sure I have discussed this before so consider this another go at it.

The `plain.tex` file start with the line:

```
% This is the plain TeX format that's described in The TeXbook.
```

A few lines later we read:

```
% And don't modify the file under any circumstances.
```

So, this format related to the TeX reference. It serves as a template for what is called a macro package. Here I will not go into the details of macro programming but an occasional snippet of code can be illustrative.

8.2 Getting started

The first code we see in the plain file is:

```
\catcode`\{=1 % left brace is begin-group character
\catcode`\}=2 % right brace is end-group character
\catcode`\$=3 % dollar sign is math shift
\catcode`\&=4 % ampersand is alignment tab
\catcode`\#=6 % hash mark is macro parameter character
\catcode`\^=7 \catcode`\^^K=7 % circumflex and uparrow are for superscripts
\catcode`\_=8 \catcode`\^^A=8 % underline and downarrow are for subscripts
\catcode`\^^I=10                          % ascii tab is a blank space
\chardef\active=13    \catcode`\~=\active % tilde is active
\catcode`\^^L=\active \outer\def^^L{\par} % ascii form-feed is "\outer\par"
```

Assigning catcodes to the braces and hash are needed in order to make it possible to define macros. The dollar is set to enter math mode and the ampersand becomes a separator in tables. The superscript and subscript also relate to math. Nothing demands

these bindings but they are widely accepted. In this respect ConTeXt is indeed like plain.

The tab is made equivalent to a space and a tilde is made active which means that later on we need to give it some meaning. It is quite normal to make that an unbreakable space, and one with the width of a digit when we're doing tables. Now, nothing demands that we have to assume ASCII input but for practical reasons the formfeed character is made equivalent to a `\par`.

Now what do these `^^K` and similar triplets represent? The `^^A` represents character zero and normally all these control characters below decimal 32 (space) are special. The `^^I` is the ASCII tab character, and `^^L` the formfeed. But, the ones referred to as uparrow and downarrow in the comments have only meaning on certain keyboards. So these are typical definitions that only made sense for Don Knuth at that time and are not relevant in other macro packages that aim at standardized input media.

```
% We had to define the \catcodes right away, before the message line, since
% \message uses the { and } characters. When INITEX (the TeX initializer) starts
% up, it has defined the following \catcode values:
%
% \catcode`\^^@=9  % ascii null is ignored
% \catcode`\^^M=5  % ascii return is end-line
% \catcode`\\=0    % backslash is TeX escape character
% \catcode`\%=14   % percent sign is comment character
% \catcode`\ =10   % ascii space is blank space
% \catcode`\^^?=15 % ascii delete is invalid
% \catcode`\A=11 ... \catcode`\Z=11 % uppercase letters
% \catcode`\a=11 ... \catcode`\z=11 % lowercase letters
% all others are type 12 (other)
```

The comments above speak for themselves. Changing catcodes is one way to adapt interpretation. For instance, in verbatim mode most catcodes can best be made letter or other. In ConTeXt we always had so called catcode regimes: for defining macros, for normal text, for XML, for verbatim, etc. In MkIV this mechanism was adapted to the new catcode table mechanism available in that engine. It was one of the first things we added to LuaTeX. So, again, although we follow some standards (expectations) ConTeXt differs from plain.

```
% We make @ signs act like letters, temporarily, to avoid conflict between user
% names and internal control sequences of plain format.

\catcode`@=11
```

In ConTeXt we went a step further and when defining macros also adapted the catcode of `!` and `?` and later in MkIV `_`. When we're in unprotected mode this applies. In addition to regular text input math is dealt with:

```
% INITEX sets up \mathcode x=x, for x=0..255, except that
```

```
%
% \mathcode x=x+"7100, for x = `A to `Z and `a to `z;
% \mathcode x=x+"7000, for x = `0 to `9.

% The following changes define internal codes as recommended in Appendix C of
% The TeXbook:

\mathcode`\^^@="2201 % \cdot
\mathcode`\^^A="3223 % \downarrow
\mathcode`\^^B="010B % \alpha
\mathcode`\^^C="010C % \beta
.....................
\mathcode`\|="026A
\mathcode`\}="5267
\mathcode`\^^?="1273 % \smallint
```

Here we see another set of definitions but the alphabetic ones are not defined in ConTeXt, they are again bindings to the authors special keyboard.

```
% INITEX sets \sfcode x=1000 for all x, except that \sfcode`X=999 for uppercase
% letters. The following changes are needed:

\sfcode`\)=0 \sfcode`\'=0 \sfcode`\]=0

% The \nonfrenchspacing macro will make further changes to \sfcode values.
```

Definitions like this depend on the language. Because original TeX was mostly meant for typesetting English, these things are hard coded. In ConTeXt such definitions relate to languages.

I show these definitions because they also illustrate what TeX is about: typesetting math:

```
% Finally, INITEX sets all \delcode values to -1, except \delcode`.=0

\delcode`\(="028300
\delcode`\)="029301
\delcode`\[="05B302
\delcode`\]="05D303
\delcode`\<="26830A
\delcode`\>="26930B
\delcode`\/="02F30E
\delcode`\|="26A30C
\delcode`\\="26E30F

% N.B. { and } should NOT get delcodes; otherwise parameter grouping fails!
```

Watch the last comment. One of the complications of TEX is that because some characters have special meanings, we also need to deal with exceptions. It also means that arbitrary input is not possible. For instance, unless the percent character is made a letter, everything following it till the end of a line will be discarded. This is an areas where macro packages can differ but in MKII we followed these rules. In MKIV we made what we called `\nonknuthmode` default which means that ampersands are just that and scripts are only special in math (there was also `\donknuthmode`). So, CONTEXT is not like plain there.

8.3 Housekeeping

The next section defines some numeric shortcuts. Here the fact is used that a defined symbolic character can act as counter value. When the number is larger than 255 a math character is to be used. In LUATEX, which is a UNICODE engine character codes can be much larger.

```
% To make the plain macros more efficient in time and space, several constant
% values are declared here as control sequences. If they were changed, anything
% could happen; so they are private symbols.

\chardef\@ne=1
\chardef\tw@=2
\chardef\thr@@=3
\chardef\sixt@@n=16
\chardef\@cclv=255
\mathchardef\@cclvi=256
\mathchardef\@m=1000
\mathchardef\@M=10000
\mathchardef\@MM=20000
```

In CONTEXT we still support these shortcuts but never use them ourselves. We have plenty more variables and constants and nowadays always use verbose names. (There was indeed a time when each extra characters depleted string memory more and more so then using short command names made sense.) The comment is right that using such variables is more efficient, for instance once loaded a macro is a sequence of tokens, so `\@one` takes one memory slot. In the case of the first three the saving is zero and even interpreting a single character token `3` is not less efficient than `\thr@@`, but in the case of `\@cclv` the three tokens `255` take more memory and also trigger the number scanner which is much slower than simply taking the meaning of the `\chardef`'d token. However, the CONTEXT variable `\plusone` is as efficient as the `\@ne` and it looks prettier in code too (and I'm very sensitive for that). So, here CONTEXT is definitely different!

It makes no sense to show the next section here: it deals with managing registers, like counters and dimensions and token lists. Traditional TEX has 255 registers per category. Associating a control sequence (name) with a specific counter is done with `\countdef` but I don't think that you will find a macro package that expects a user to use that

primitive. Instead it will provide a \newcount macro. So yes, here ConTeXt is like plain.

Understanding these macros is a test case for understanding TeX. Take the following snippet:

```
\let\newtoks=\relax % we do this to allow plain.tex to be read in twice
\outer\def\newhelp#1#2{\newtoks#1#1\expandafter{\csname#2\endcsname}}
\outer\def\newtoks{\alloc@5\toks\toksdef\@cclvi}
```

The \outer prefix flags macros as to be used at the outermost level and because the \newtoks is in the macro body of \newtoks it has to be relaxed first. Don't worry if you don't get it. In ConTeXt we have no outer macros so the definitions differ there.

The plain format assumes that the first 10 registers are used for scratch purposes, so best also assume this to be the case in other macro packages. There is no need for ConTeXt to differ from plain here. The definitions of box registers and inserts are special: there is no \boxdef and inserts use multiple registers. Especially the allocation of inserts is macro package specific. Anyway, ConTeXt users never see such details because inserts are used as building blocks deep down.

Right after defining the allocators some more constants are defined:

```
% Here are some examples of allocation.

\newdimen\maxdimen \maxdimen=16383.99999pt % the largest legal <dimen>
```

We do have that one, as it's again a standard but we do have more such constants. This definition is kind of interesting as it assumes knowledge about what is acceptable for TeX as dimension:

```
{\dimen0=16383.99999pt \the\dimen0 \quad \number\dimen0}
{\dimen0=16383.99998pt \the\dimen0 \quad \number\dimen0}
```

16383.99998pt 1073741823
16383.99998pt 1073741823

Indeed it is the largest legal dimension but the real largest one is slightly less. We could also have said the following, which also indicates what the maximum cardinal is:

```
\newdimen\maxdimen \maxdimen=1073741823sp
```

We dropped some of the others defined in plain. So, ConTeXt is a bit like plain but differs substantially. In fact, MkII already used a different allocator implementation and MkIV is even more different. We also have more \new things.

The \newif definition also differs. Now that definition is quite special in plain TeX, so if you want a challenge, look it up. It defines three macros as the comment says:

```
% For example, \newif\iffoo creates \footrue, \foofalse to go with \iffoo.
```

The \iffoo is either equivalent to \iftrue or \iffalse because that is what TEX needs to see in order to be able to skip nested conditional branches. In CONTEXT we have so called conditionals, which are more efficient. So, yes, you will find such defined ifs in the CONTEXT source but way less than you'd expect in such a large macro package: CONTEXT code doesn't look much like plain code I fear.

8.4 Parameters

A next stage sets the internal parameters:

```
% All of TeX's numeric parameters are listed here, but the code is commented out
% if no special value needs to be set. INITEX makes all parameters zero except
% where noted.
```

We use different values for many of them. The reason is that the plain TEX format is set up for a 10 point Computer Modern font system, and for a particular kind of layout, so we use different values for:

```
\hsize=6.5in
\vsize=8.9in
\maxdepth=4pt
```

and

```
\abovedisplayskip=12pt plus 3pt minus 9pt
\abovedisplayshortskip=0pt plus 3pt
\belowdisplayskip=12pt plus 3pt minus 9pt
\belowdisplayshortskip=7pt plus 3pt minus 4pt
```

No, here CONTEXT is not like plain. But, there is one aspect that we do inherit and that is the ratio. Here a 10 point relates to 12 point and this 1.2 factor is carried over in some defaults in CONTEXT. So, in the end we're a bit like plain.

After setting up the internal quantities plain does this:

```
\newskip\smallskipamount \smallskipamount=3pt plus 1pt minus 1pt
\newskip\medskipamount \medskipamount=6pt plus 2pt minus 2pt
\newskip\bigskipamount \bigskipamount=12pt plus 4pt minus 4pt
\newskip\normalbaselineskip \normalbaselineskip=12pt
\newskip\normallineskip \normallineskip=1pt
\newdimen\normallineskiplimit \normallineskiplimit=0pt
\newdimen\jot \jot=3pt
\newcount\interdisplaylinepenalty \interdisplaylinepenalty=100
\newcount\interfootnotelinepenalty \interfootnotelinepenalty=100
```

The first three as well as the following three related variables are not internal quantities but preallocated registers. These are not used in the engine but in macros. In CONTEXT

we do provide them but the first three are never used that way. The last three are not defined at all. So, ConTeXt provides a bit what plain provides, just in case.

8.5 Fonts

The font section is quite interesting. I assume that one reason why some want to warn users against using ConTeXt is because it supports some of the font switching commands found in plain. We had no reasons to come up with different ones but they do different things anyway, for instance adapting to situations. So, in ConTeXt you will not find the plain definitions:

```
\font\tenrm=cmr10 % roman text
\font\preloaded=cmr9
\font\preloaded=cmr8
\font\sevenrm=cmr7
\font\preloaded=cmr6
\font\fiverm=cmr5
```

There is another thing going on here. Some fonts are defined `\preloaded`. So, `cmr9` is defined, and then `cmr8` and `cmr6`. But they all use the same name. Later on we see:

```
\let\preloaded=\undefined % preloaded fonts must be declared anew later.
```

If you never ran into the relevant part of the TeX book or read the program source of TeX, you won't realize that preloading means that it stays in memory which in turn means that when it gets (re)defined later, the font data doesn't come from disk. In fact, as the plain format is normally dumped for faster reload later on, the font data is also retained. So, preloading is a speed up hack. In ConTeXt font loading has always been delayed till the moment a font is really used. This permits plenty of definitions and gives less memory usage. Of course we do reuse fonts once loaded. All this, plus the fact that we have a a system of related sizes, collections of families, support multiple font encodings alongside, collect definitions in so called typescript, etc. makes that the ConTeXt font subsystem is far from what plain provides. Only some of the command stick, like `\rm` and `\bf`.

The same is true for math fonts, where we can have different math font setups in one document. Definitely in MkII times, we also had to work around limitations in the number of available math families, which again complicated the code. In MkIV things are even more different, one can even consider the implementation somewhat alien for a standard macro package, but that's for another article (if at all).

8.6 Macros

Of course ConTeXt comes with macros, but these are organized in setups, environments, instances, etc. The whole process and setup is keyword driven. Out of the box all things work: nothing needs to be loaded. If you want it different, you change some

settings, but you don't need to load something. Maybe that last aspect is what is meant with ConTeXt being like plain: you don't (normally) load extra stuff. You just adapt the system to your needs. So there we proudly follow up on plain TeX.

In the plain macro section we find definitions like:

```
\def\frenchspacing{\sfcode`\.\@m \sfcode`\?\@m \sfcode`\!\@m
  \sfcode`\:\@m \sfcode`\;\@m \sfcode`\,\@m}
\def\nonfrenchspacing{\sfcode`\.3000\sfcode`\?3000\sfcode`\!3000%
  \sfcode`\:2000\sfcode`\;1500\sfcode`\,1250 }
```

and:

```
\def\space{ }
\def\empty{}
\def\null{\hbox{}}

\let\bgroup={
\let\egroup=}
```

and:

```
\def\nointerlineskip{\prevdepth-1000\p@}
\def\offinterlineskip{\baselineskip-1000\p@
  \lineskip\z@ \lineskiplimit\maxdimen}
```

Indeed we also provide these, but apart from the two grouping related aliases their implementation is different in ConTeXt. There is no need to reinvent names.

For a while we kept (and did in MkII) some of the plain helper macros, for instance those that deal with tabs, but we have several more extensive table models that are normally used. We always had our own code for float placement, and we also have more options there. Footnotes are supported but again we have multiple classes, placements, options, etc. Idem for itemized lists, one of the oldest mechanisms in ConTeXt. We don't have `\beginsection` but of course we do have sectioning commands, and have no `\proclaim` but provide lots of descriptive alternatives, so many that I forgot about most of them by now (so plain is a winner in terms of knowing a macro package inside out).

The fact that we use tables, floats and footnotes indeed makes ConTeXt to act like plain, but that's then also true for other macro packages. A fact is that plain sets the standard for how to think about these matters! The same is true for naming characters:

```
\chardef\%=`\%
\chardef\&=`\&
\chardef\#=`\#
\chardef\$=`\$
\chardef\ss="19
```

```
\chardef\ae="1A
\chardef\oe="1B
\chardef\o="1C
\chardef\AE="1D
\chardef\OE="1E
\chardef\O="1F
\chardef\i="10 \chardef\j="11 % dotless letters
```

But we have many more and understandable the numbers are different in ConTeXt because we use different font (encodings). Their implementation is more adaptive. The same is true for accented characters:

```
\def\`#1{{\accent18 #1}}
\def\'#1{{\accent19 #1}}
```

The definitions in MkII are different (in most cases we use native glyphs) and in MkIV we use Unicode anyway. I think that the `\accent` command is only used in a few exceptional cases (like very limited fonts) in MkII and never in MkIV. The implementation of for instance accents (and other pasted together symbols) in math is also quite different.

There are also definitions that seem to be commonly used in macro packages but that we never use in ConTeXt because they interfere badly with all kind of other mechanisms, so you will find no usage of

```
\def\leavevmode{\unhbox\voidb@x} % begins a paragraph, if necessary
```

in ConTeXt. In order to stress that we provide `\dontleavehmode`, a wink to not using the one above.

The macro section ends with lots of math definitions. Most of the names used are kind of standard so again here ConTeXt is like plain, but the implementation can differ as does the level of control.

8.7 Output

Once a page is ready it gets wrapped up and shipped out. Here ConTeXt is very different from plain. The amount of code in plain is not that large but the possibilities aren't either, which is exactly what the objectives demand: a simple (example) format that can be described in the TeXbook. But, as with other aspects of plain, it steered the way macro packages started out as it showed the way. As did many examples in the TeX book.

8.8 Hyphenation

As an afterthought, the plain format ends with loading hyphenation patterns, that is the English ones. That said it will be clear that ConTeXt is not like plain: we support

many languages, and the subsystem deals with labels, specific typesetting properties, etc. too.

```
\lefthyphenmin=2 \righthyphenmin=3 % disallow x- or -xx breaks
\input hyphen
```

We don't even use these patterns as we switched to UTF long ago (also in MKII) if only because we had to deal with a mix of font encodings. But we did preload the lot there. In MKIV again things are much different.

8.9 Conclusion

The plain format does (and provides) what it is supposed to do. It is a showcase of possibilities and part of the specification. In that respect it's nice that CONTEXT is considered to be like plain. But if it wasn't more, there was no reason for its existence. Like more assumptions about CONTEXT it demonstrates that those coming up with answers and remarks like that probably missed something in assessing CONTEXT. Just let users find out themselves what suits best (and for some that actually might be plain TEX).

Let me make on thing clear. If you look at the documents that describe the development of TEX, METAFONT and the related fonts, you can only awe at what was done on hardware that doesn't come close to what we hold now in the palm of our hand. And it was done in a relative short time span. The fact that plain TEX ran on it the way it did is amazing. Anyone who considers criticizing TEX and plain TEX should think (more than) twice.

False promises 9

9.1 Introduction

Hans Hagen
Hasselt NL
July 2019 (public 2023)

The TEX typesetting system is pretty powerful, and even more so when you combine it with METAPOST and LUA. Add an XML parser, a whole lot of handy macros, provide support for fonts and advanced PDF output and you have a hard to beat tool. We're talking CONTEXT.

Such a system is very well suited for fully automated typesetting. There are TEX lovers who claim that TEX can do anything better than the competition but that's not true. Automated typesetting is quite doable when you accept the constraints. When the input is unpredictable you need to play safe!

Some things are easy: turning complex XML into PDF with adaptive graphics, fast data processing, colorful layouts, conditional processing, extensive cross referencing, you can safely say that it can be done. But in practice there is some design involved and those are often specified by people who manipulate a layout on the fly and tweak and cheat in an interactive WYSIWYG program. That is however not an option in automated typesetting. Traditional thinking with manual intervention has to make place for systematic and consistent solutions. Limitations can be compensated by clever designs and getting the maximum out of the system used.

Unfortunately in practice some habits are hard to get rid of. Inconsistent use of colors, fonts, sectioning, image placements are just a few aspects that come to mind. When you typeset educational documents you also have to deal with strong opinions about how something should be presented and what students can't (!) handle, like for instance cross references. One of the most dominant demands in typesetting such documents are so called side floats. In (for instance) scientific publishing references to content typeset elsewhere (formulas, graphics) is acceptable but in educational documents this is often not an option (don't ask me why).

In the next sections I will mention a few aspects of side floats. I will not discuss the options because these are covered in manuals. Here we stick to the challenges and the main question that you have to ask yourself is: “How would I solve that if it can be solved at all?”. It might make you a bit more tolerant for suboptimal outcome.

9.2 The basics

We start with a simple example. The result is shown in figure 9.1. We have figures, put at the left, with enough text alongside so that we don't have a problem running into the next figure.

```
\dorecurse {8} {
    \useMPlibrary[dum]
    \setuplayout[middle]
    \setupbodyfont[plex]
    \startplacefigure[location=left]
        \externalfigure[dummy][width=3cm]
    \stopplacefigure
    \samplefile{sapolsky}
    \par
}
```

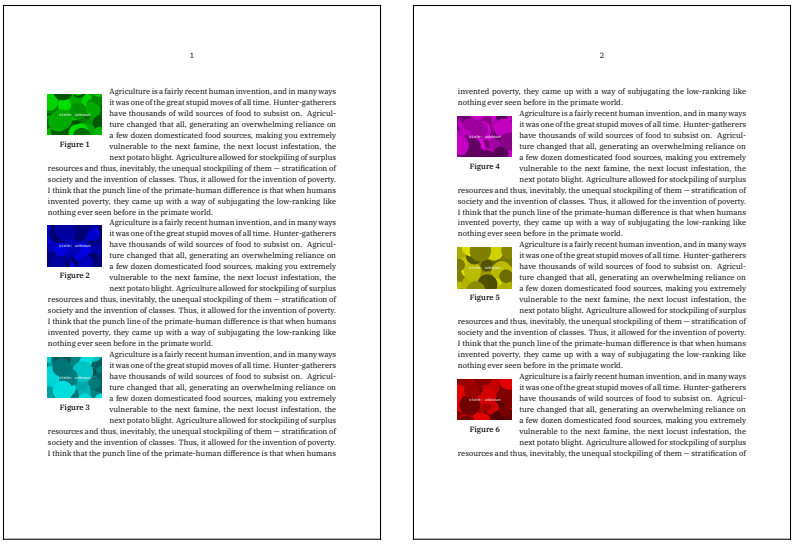


Figure 9.1 A simple example with enough text in a single paragraph.

Challenge: Anchor some boxed material to the running text and make sure that the text runs around that material. When there is not enough room available on the page, enforce a page break and move the lot to the next page.

But more often than not, the following paragraph is not long enough to go around the insert. The worst case is of course when we end up with one word below the insert, for which the solution is to adapt the text or make the insert wider or narrower. Forgetting about this for now, we move to the case where there is not enough text: figure 9.2.

```
\dorecurse {8} {
    \useMPlibrary[dum]
    \setuplayout[middle]
    \setupbodyfont[plex]
    \startplacefigure[location=left]
```

```
        \externalfigure[dummy][width=3cm]
    \stopplacefigure
    \samplefile{ward} \par \samplefile{ward}
    \par
}
```

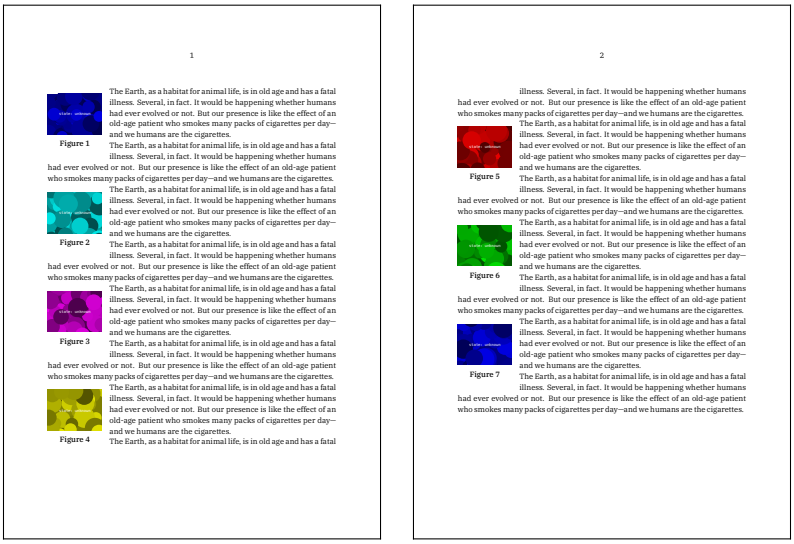


Figure 9.2 A simple example with enough text but multiple paragraphs.

Challenge: At every new paragraph, check if we're still not done with the blob we're typesetting around and carry on till we are behind the insert.

The next example, shown in figure 9.3, has less text. However, the running text is still alongside the figure, so this means that white space need to be added till we're beyond.

```
\dorecurse {8} {
    \useMPlibrary[dum]
    \setuplayout[middle]
    \setupbodyfont[plex]
    \startplacefigure[location=left]
        \externalfigure[dummy][width=3cm]
    \stopplacefigure
    \samplefile{ward}
    \par
}
```

Challenge: When there is not enough content, and the next insert is coming, we add enough whitespace to go around the insert and then start the new one. This is typically something that can also be enforced by an option.

Before we move on to the next challenge, let's explain how we run around the insert. When TEX typesets a paragraph, it uses dimensions like `\leftskip` and `\rightskip` (margins) and shape directives like `\hangindent` and `\hangafter`. There is also the possibility to define a `\parshape` but we will leave that for now. The with of the image

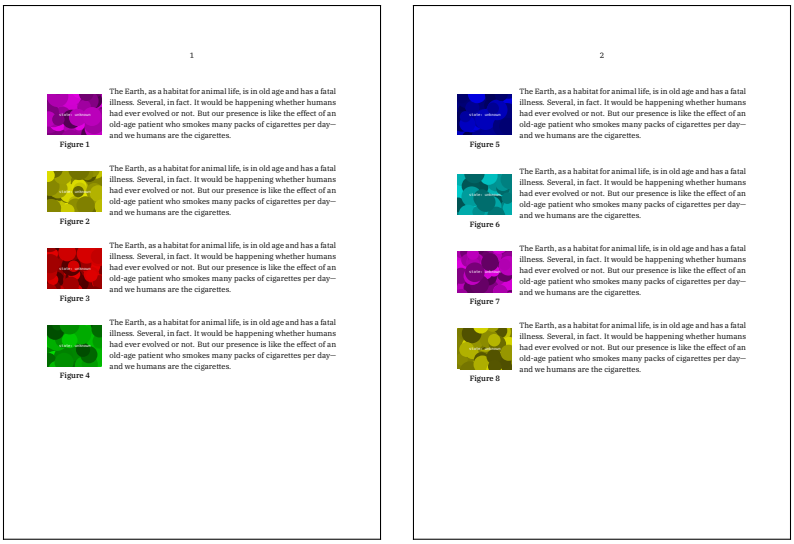


Figure 9.3 A simple example with less text

is reflected in the indent and the height gets divided by the line height and becomes the `\hangafter`. Whenever a new paragraph is started, these parameters have to be set again.[5] In ConTeXt hanging is also available as basic feature.

```
\starthanging[location=left]
    {\blackrule[color=maincolor,width=3cm,height=1cm]}
    \samplefile{carrol}
\stophanging
```

The fraction of fossil olfactory receptor genes is significantly higher in all species with full color vision. This suggests that the evolution of trichromatic vision — which allows these primates to detect food, mates, and danger with visual cues — has reduced their reliance on the sense of smell.

```
\starthanging[location=right]
    {\blackrule[color=maincolor,width=10cm,height=1cm]}
    \samplefile{jojomayer}
\stophanging
```

If we surrender the thing that separates us from machines, we will be replaced by machines. The more advanced machines will be, the more human we will have to become.

The hanging floats are not implemented this way but are hooked into the paragraph start routines. The original approach was a variant of the macros by Daniel Comenetz as published in TUGBoat Volume 14 (1993), No. 1: Anchored Figures at Either Margin. In the meantime they are far from that, so ConTeXt users can safely blame me for any issues.

[5] I still consider playing with a third parameter representing hang height and add that to the line break routine, but I have to admit that tweaking that is tricky. Do I really understand what is going on there?

9.3 Unpredictable dimensions

In an ideal world images will be sort of consistent but in practice the dimension will differ, even fonts used in graphics can be different, and they can have white space around them. When testing a layout it helps to use mockups with a clear border. If these look okay, one can argue that worse looking assemblies (more visual whitespace above of below) is a matter of making better images. In figure 9.4 we demonstrate how different dimensions influence the space below the placement.

```
\dostepwiserecurse {2} {8} {1} {
    \useMPlibrary[dum]
    \setuplayout[middle]
    \setupbodyfont[plex]
    \setupalign[tolerant,stretch]
    \startplacefigure[location=left]
        \externalfigure[dummy][width=#1cm]
    \stopplacefigure
    \samplefile{sapolsky}
    \par
}
```

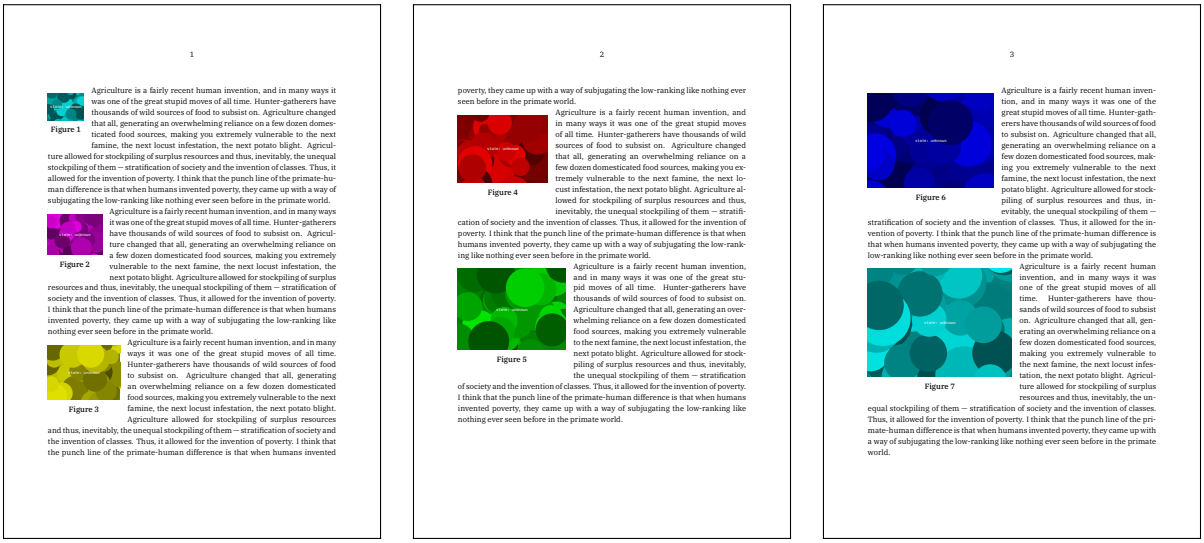


Figure 9.4 Spacing relates to dimensions.

In ConTeXt there are plenty of options to add more space above or below the image. You can anchor the image to the first line in different ways and you can move it some lines down, either or not with text flowing around it. But here we stick to simple cases, we only discuss the challenges.

Challenge: Adapt the wrapping to the right dimensions and make sure that the (optional) caption doesn't overlap with the text below.

9.4 Moving forward

When the insert doesn't fit it has to move, which is why it's called a float. One solution is do take it out of the page stream and turn it into a regular placement, normally centered horizontally somewhere on the page, and in this case probably at the top of one of the next pages. Because we can cross reference this is a quite okay solution. But, in educational documents, where authors refer to the graphic (picture) on the left or right, that doesn't work out well. The following content is bound to the image.

Calculating the amount of available space is a bit tricky due to the way TEX works. But let's assume that this can be done, in CONTEXT we have seen several strategies for this, we then end up at the top of the next page and there different spacing rules apply, like: no spacing at the top at all. In our examples no whitespace between paragraphs is present. The final solutions are complicated by the fact that we need to take this into account.

Challenge: Make sure that we never run off the page but also that we don't end up with weird situations at the top of the next page.

Another possibility is that images so tightly fit a whole number of lines, that a next one can come too close to a previous one. Again, this demands some analysis. Here we use examples with captions but when there are no captions, there is also less visual space (no depth in lines).

Challenge: Make sure that a following insert never runs too close to a previous insert.

Solutions can be made better when we use multi-pass information. Because in a typical TEX run there is only looking back, storing information can actually make us look forward. But, as in science fiction: when you act upon the future, the past becomes different and therefore also the future (after that particular future). This means that you can only go forward. Say you have 10 cases: when case 5 changes because of some feedback, then case 6 upto 10 also can change. So, you might need more than 10 runs to get things right. In a workflow where users are waiting for a result, and a few hundred side floats are used this doesn't sell well: processing 400 pages with a 20 page per second rate takes 20 seconds per run. Normally one needs a few runs to get the references right. Assuming a worst case of 60 seconds, 10 extra runs will bring you close to 15 minutes. No deal.

Of course one can argue for some load-in-memory and optimize in one go, but although TEX can do that well for paragraphs, it won't work for complex documents. Sure, it's a nice academic exercise to explore limited cases but those are not what we encounter.

9.5 Cooperation

When discussing (on YouTube) “Extending Darwin's Revolution” David Sloan Wilson and Robert Sapolsky touch on the fact that in some disciplines (like economics) evolutionary principles are applied. One can apply for instance the concept of a ‘selfish

gene'. However, they argue that when doing that, one actually lags behind the now accepted group selection (which goes beyond the individual benefits). An example is given where aggressive behavior on the short term can turn one in a winner (who takes it all) but which can lead to self destructive in the long run: cooperating seems to works better than terminal competition.

In TEX we have glues and penalties. The machinery likes to break at a glue but a severe penalty can prohibit that. The fact that we have penalties and no rewards is interesting: a break can be stimulated by a negative penalty. I've forgotten most of what I learned about cognitive psychology but I do remember that penalty vs reward discussions could get somewhat out of hand.

So, when we have in the node list a mix of glue (you can break here), penalties (better not break here) and rewards (consider breaking here) you can imagine that these nodes compete. The optimal solution is not really a group process but basically a rather selfish game. Building a system around that kind of cooperation is not easy. In CONTEXT a lot of attention always went into consistent vertical spacing. In MKII there were some 'look back' and 'control forward' mechanisms in place, and in MKIV we use a model of weighted glue: a combination of penalties and skips. Again we look back and again we also try to control the future. This works reasonable well but what if we end up in a real competition?

A section head should not end up at the bottom of a page. Because when it gets typeset it is unknown what follows, it does some checking and then tries to make sure that there is no page break following. Of course there needs to be a provision for the cases that there are many (sub)heads and of course when there are only heads on a page (in a concept for instance) you don't want to run of the page.

Similar situations arise with for instance itemized lists and the tabulate mechanism. There we have some heuristics that keep content together in a way that makes sense given the construct: no single table line at the bottom of a page etc. But then comes the side float. The available space is checked. When doing that the whitespace following the section head has to collapse with the space around the image, but of course at the top of a page spacing is different. So, calculations are done, but even a small difference between what is possible and what is needed can eventually still trigger an unwanted page break. This is because you cannot really ask how much has been accumulated so far: the space used is influenced by what comes next (like whitespace, maybe interline space, the previous depth correction, etc). That in turn means that you have to (sort of) trigger these future space related items to be applied already.

Challenge: Let the side float mechanism nicely cooperate with other mechanisms that have their own preferences for crossing pages, adding whitespace and being bound to following content.

9.6 Easy bits

Of course, once there is such a mechanism in place, user demands will trigger more features. Most of these are actually not that hard to deal with: renumbering due to

moved content, automatic anchoring to the inner or outer margin, horizontal placement and shifting into margins, etc. Everything that doesn't relate to vertical placement is rather trivial to deal with, especially when the whole infrastructure for that is already present (as in ConTeXt). The problem with such extensions is that one can easily forget what is possible because most are rarely used.

Challenge: Make sure that all fits into an understandable model and is easy to control.

9.7 Conclusion

The side float mechanism in ConTeXt is complex, has many low level options, and its code doesn't look pretty. It is probably the mechanism that has been overhauled and touched most in the code base. It is also the mechanism that (still) can behave in ways you don't expect when combined with other mechanisms. The way we deal with this (if needed) is to add directives to (in our case) XML files that tells the engine what to do. Because that is a last resort it is only needed when making the final product. So in the end, we're still have the benefits of automated typesetting.

Of course we can come up with a different model (basically re-implement the page builder) but apart from much else falling apart, it will just introduce other constraints and side effects. Thinking in terms of selfish nodes, glues and penalties, works ok for a specific document where one can also impose usage rules. If you know that a section head is always followed by regular text, things become easier. But in a system like ConTeXt you need to update your thinking to group selection: mechanisms have to work together and that can be pretty complicated. Some mechanisms can do that better than others. One outcome can be that for instance side floats are not really group players, so eventually they might become less popular and fade away. Of course, as often, years later they get rediscovered and the cycle starts again. Maybe a string argument can be made that in fully automated typesetting concepts like side floats should not be used anyway.

If I have to summarize this wrap up, the conclusion is that we should be realistic: we're not dealing with an expert system, but with a bunch of heuristics. You need an intelligent system to help you out of deadlock and oscillating solutions. Given the different preferences you need a multiple personality system. You might actually need a system that wraps your expectations and solutions and that adapts to changes in those over time. But if there is such a system (some day) it probably doesn't need you. In fact, maybe even typesetting is not needed any more by then.

About manuals 10

10.1 Introduction

I'm always puzzled when I read that someone wonders if ConTeXt is still up to date or maintained because some manual has a timestamp of a decade ago. I'm also puzzled by some rants you can run into when searching the web. In the next few paragraphs I'll comment on this.

10.2 Stability

Say that you're an enthusiastic user of console commands like `ls` (`dir`), `cp` (`copy`) or maybe `ssh`, `rsync`, `curl`. How often do you consult a manual on how they evolve? And say that you, for some reason, do consult a manual, there is a good change that it is pretty old. Does that mean that the commands are obsolete? The binaries probably get fixed for bugs but the interface stays the same, which is what you expect. Every time we generate a zip for the ConTeXt distribution, the related website also gets generated, using a bunch of XML files that get transformed to HTML using XSLT and a pretty ancient version of `xsltproc` (why should I update). I never check for a new manual as it keeps doing the job. And additional manuals and reports get added.

So, once some functionality is stable, and a lot of macro code in ConTeXt is just that, there is no need to update a manual! Putting a new time stamp on it is basically fake updating. And often the more introductionary kind of manuals don't need to be updated at all, apart from maybe cultural changes that demand a (political correct) update. Them being a bit old and not being updated is actually a good thing as it signals stability.

It is worth mentioning that the ConTeXt distribution is not the only source of information. There are manuals written by others and there is the Wiki. All is the work of volunteers and updating all that depends on how much time one can allocate.

10.3 Excuses

It is a fact that ConTeXt evolves. New functionality gets added and some mechanism get extended. Often these are described in dedicated manuals or articles that end up in collections, and there are plenty of them in the distribution. For some reason those complaining about a beginners manual with an old time stamp don't check if there is more, and there is quite some more! Don't only look at the ConTeXt garden (the wiki) but also keep an eye on what gets distributed. Some users are very good in track of what gets added, because sometimes I get fixes for typos send within a few hours after uploading a zip.

We appreciate that other users point out that writing manuals takes time and that indeed our time is not without limits. If I could sit down and write manuals whole day, and it would get paid, I might do it. But it is a fact that development of CONTEXT is not paid for at all. I can work on it in company time but much happens in spare time. Most development is a gamble on future use or done because we want to be complete or because code can be improved. So, writing a manual then closely relates to what we like doing: it determines the topics and priorities. If something gets explored and ends up in new functionality then that gets documented in the process. It is the fun factor that drives it. The same is true for LUATEX development.

So, we have as valid excuse that new manuals relate to (new) functionality and old ones stay as they are. Don Knuth remarks somewhere that writing a manual as part of the development is a good thing. We fully agree with that.

10.4 Cutting edge

Does an old manual indicate that nothing happens? Definitely not. Over a decade of LUATEX development is closely related to CONTEXT and there is plenty of reporting about that. Does that mean that we need to rewrite manuals? No, existing functionality remains. And of course users are free to come up with more detailed manuals (which they seldom do). Some developments get published in user group journals but we don't publish much about specific CONTEXT features and usage because it's hard to do that for a diverse audience.

Currently we have what is called CONTEXT MKXL (aka LMTX), but we also have the prelude to that, MKIV, and the frozen predecessor MKII. Apart from changes in technology (most noticeably fonts and encodings) the functionality is accumulative: most old manuals (unless they are specialized into old school fonts for instance) apply to the latest greatest version.

It is a misunderstanding that the development of CONTEXT, LUATEX and LUAMETATEX is somehow funded by projects that we do. This is not true. We can apply both in projects but as we charge by the hour (or day) no customer ever sees development on the bill. Of course during a project we can gain on efficiency (so then development pays back) and because we know the system style writing is efficient too. In fact, in most cases our customers don't know or care what tool we use because tools are expected to be part of the deal. Most projects we can (and could) only do because we can use CONTEXT and that is a side effect of the fact that we do develop beforehand. We're often the only technically and/or affordable way out. It's a chicken-egg issue: we have a tool and therefore get a project. We never get a projects where we can develop a tool. No one pays for TEX development or at least no one ever came to us with specific TEX related demands. It looks like the world takes it for granted that TEX is just there.[6]

[6] There are a few subcomponents of CONTEXT that were partially sponsored by users and we do have some support contracts that permit experiments and development.

The reason for ConTeXt being cutting edge (in terms of TeX) is that we like challenges, that users demand features that are interesting to explore and that we've been part of the TeX scenery for a while now. We just like that.

It's good to know that ConTeXt was and is developed as a toolkit. We started long ago because we needed a way to quickly create and update reports of meetings that we chaired. Next we needed a way to efficiently produce high quality education materials (of various kind) and support maintenance of sets of related (quality assurance) manuals. We could have used wordstar, wordperfect or msword but liked the TeX way much more. As said, most customers didn't even know or care what tool was used because the (often highly interactive PDF) outcome mattered most. In fact, we would not be interested in this kind of work if we were forced to use clumsy tools, but for sure a lot can be done with those as well.

10.5 Continuity

Most development happens at Pragma ADE by me (Hans) with help from my two colleagues (Ton and Kees) and the community (Aditya, Alan, Mikael, Mojca, Luigi, Hraban, Taco, Thomas, Tomas, Willi, Wolfgang, and others).[7] I won't mention those on the mailing list who contribute with ideas, testing and support, but they can't be missed. The biggest danger for continuity is a polluted code base where everyone just pushed code into a repository. So this is closely guarded. A user patch might work well for that user but can break something else.

With TeX you need to keep in mind that once a solution works there is no need to update code or manuals. As long as there is a working LuaTeX (LuaMetaTeX) binary you're fine. Maybe if some specific fonts are used, a filename might need to be adapted.

An example. When we added a new XML subsystem to ConTeXt MkIV we knew that some day we could use it. We now uses it in a few projects and I'm pretty sure that we would not do these projects otherwise as it would demand writing quite complex XSLT style sheets that then would have to be applied to thousands of files per run. To some extend what is available in ConTeXt sort of drives the kind of work you look for. That said: if you consider using ConTeXt for simple or complex documents, either of not in a collection, either or not using TeX or XML input you can be assured that this will work (and might even get better) because we use it ourselves.

If you want to get an idea about development, just look at the (five) documents that describe the development of LuaTeX (LuaMetaTeX). Locating them in the distribution is a good opportunity to explore the documents. They will show you what happened the last decade(s) and give you some trust in ConTeXt. Or come to a ConTeXt meeting and meet those involved.

[7] More names could be here as I write this in 2022.

10.6 Closing remark

So next time someone asks if CONTEXT is maintained because some old manual stays around, return the question if frequently updated manuals are a sign of stability. Also ask if someone looked a bit in the documentation tree. The oldest manual in the TEX world is the TEXbook that describes the oldest stable set of macros: plain TEX. There are happy users out there who love that stability. If it were not for the wonderful personality of Don Knuth this program would already been forgotten. I think that long term stability and unchanged code and manuals are something that we need to cherish and get accustomed to, which is not easy in a time when a phone and its operating system are outdated as soon as you unbox it. It's also not easy in a time of instant communication, more and more confused by what is called artificial intelligent mumbling, but that's for another wrapup.

Hans Hagen
Hasselt NL

(uncorrected so there's something left to complain)

Performance again 11

Hans Hagen
Hasselt NL
Februari 2020 (public 2023)

11.1 Introduction

In a MAPS article of 2019 I tried to answer the question 'Is TEX really slow?'. A while after it was published on the Dutch TEX mailing list a user posted a comment stating that in his experience the LUATEX engine in combination with LATEX was terribly slow: one page per second for a Japanese text. It was also slower than PDFTEX with English, but for Japanese it was close to unusable. The alternative, using a Japanese TEX engine was no option due to lack of support for certain images.

In order to check this claim I ran a test in CONTEXT. Even on my 8 year old laptop I could get 45 pages per second for full page Japanese texts (6 paragraphs with each 300 characters per page): 167 pages took just less than 4 seconds. Typesetting Japanese involves specific spacing and line break handling. So, naturally the question arises: why the difference. Frans Goddijn wondered if I could explain a bit more about that, so here we go.

In the mentioned article I already have explained what factors play a role and the macro package is one of them. It is hard to say to what extent inefficient macros or a complex layout influence the runtime, but my experience is that it is pretty hard to get speeds as low as 1 page per second. On an average complex document like the LUATEX manual (lots of verbatim and tables, but nothing else demanding apart from color being used and a unique METAPOST graphic per page) I get at least a comfortable 20 pages per second.

I can imagine that for a TEX user who sees other programs on a computer do complex things fast, the performance of TEX is puzzling. But, where for instance rendering videos can benefit from specific features of (video) processors, multiple cores, or just aggressive optimization by compilers of (nested) loops and manipulation of arrays of bytes, this is not the case for TEX. This program processes all in sequence, there is not much repetition that can be optimized, it cannot exploit the processor in special ways and the compiler can not do that many optimizations.

I can't answer why a LATEX run is slower than a CONTEXT run. Actually, one persistent story has always been that CONTEXT was slow in comparison. But maybe it helps to know a bit what happens deep down in TEX and how macro code can play a role in performance. When doing that I will simplify things a bit.

11.2 Text and nodes

The TEX machinery takes input and turns that into some representation that can be turned into a visual representation ending up as PDF. So say that we have this:

```
hello
```

In a regular programming language this is a string with five characters. When the string is manipulated it is basically still a sequence of bytes in memory. In TEX, if this is meant as text, at some point the internal representation is a so called node list:

```
[h] -> [e] -> [l] -> [l] -> [o]
```

In traditional TEX these are actually character nodes. They have a few properties, like what font the character is from and what the character code is (0 up to 255). At some point TEX will turn that list into a glyph list. Say that we have this:

```
efficient
```

This will eventually become seven nodes:

```
[e] -> [ffi] -> [c] -> [i] -> [e] -> [n] -> [t]
```

The ffi ligature is a glyph node which actually also keeps information about this one character being made from three.

In LUATEX it is different, and this is one of the reasons for it being slower. We stick to the first example:

```
[h] <-> [e] <-> [l] <-> [l] <-> [o]
```

So, instead of pointing to the next node, we also point back to the previous: we have a double linked list. This means that all over the program we need to maintain these extra links too. They are not used by TEX itself, but handy at the LUA end. But, instead of only having the font as property there is much more. The TEX program can deal with multiple languages at the same time and this relates to hyphenation. In traditional TEX there are language nodes that indicate a switch to another language. But in LUATEX that property is kept with each glyph node. Actually, even specific language properties like the hyphen min, hyphen max and the choice if uppercase should be hyphenated are kept with these nodes. Spaces are turned into glue nodes, and these nodes are also larger than in regular TEX engines.

So, in LUATEX, when a character goes from the input into a node, a more complex data structure has to be set up and the larger data structure also takes more memory. That in turn means that caching (close to the CPU) gets influenced. Add to that the fact that we operate on 32 bit character values, which also comes with higher memory demands.

We mentioned that a traditional engine goes from one state of node list into another (the ligature building). Actually this is an integrated process: a lot happens on the

fly. If something is put into a \hbox no hyphenation takes place, only ligature building and kerning. When a paragraph is typeset, hyphenation happens on demand, in places where it makes sense.

In LuaTeX these stages are split. A node list is *always* hyphenated. This step as well as ligature building and kerning are *three* separate steps. So, there's always more hyphenation going on than in a traditional TeX engine: we get more discretionary nodes and again these take more memory than before; also the more nodes we have, the more it will impact performance down the line. The reason for this is that each step can be intercepted and replaced by a Lua driven one. In practice, with modern OpenType fonts that is what happens: these are dealt with (or at least managed in) Lua. For Japanese for sure the built-in ligature and kerning doesn't apply: the work is delegated and this comes at a price. Japanese needs no hyphenation but instead characters are treated with respect to their neighbors and glue nodes are injected when needed. This is something that Lua code is used for so here performance is determined by how well the plugged in code behaves. It can be inefficient but it can also be so clever that it just takes a bit of time to complete.

I didn't mention another property of nodes: attributes. Each node that has some meaning in the node list (glyphs, kerns, glue, penalties, discretionary, . . . , these terms should ring bells for a TeX user) have a pointer to an attribute list. Often these are the same for neighboring nodes, but they can be different. If a macro package sets 10 attributes, then there will be lists of ten attributes nodes (plus some overhead) active. When values change, copies are made with the change applied. Grouping even complicates this a little more. This has an impact on performance. Not only need these lists be managed, when they are consulted at the Lua end (as they are meant as communication with that bit of the engine) these lists are interpreted. It all adds up to more runtime. There is nothing like that in traditional TeX, but there some more macro juggling to achieve the same effects can cause a performance hit.

11.3 Macros and tokens

When you define a macro like this:

```
\def\MyMacro#1{\hbox{here: #1!}}
```

the TeX engine will parse this as follows (we keep it simple):

`\def`	primitive token
`\MyMacro`	user macro pointing to:
`#1`	argument list of length 1 and no delimiters
`{`	openbrace token
`\hbox`	hbox primitive token
`h`	letter token h
`e`	letter token e
`r`	letter token r

e	letter token e
:	other token :
	space token
#1	reference to argument
!	other token !
}	close brace token

The \def is eventually lost, and the meaning of the macro is stored as a linked list of tokens that get bound to the user macro \MyMacro. The details about how this list is stored internally can differ a bit per engine but the idea remains. If you compare tokens of a traditional TEX engine with LUATEX, the main difference is in the size: those in LUATEX take more memory and again that impacts performance.

11.4 Processing

Now, for a moment we step aside and look at a regular programming language, like PASCAL, the language TEX is written in, or C that is used for LUATEX. The high level definitions, using the syntax of the language, gets compiled into low level machine code: a sequence of instructions for the CPU. When doing so the compiler can try to optimize the code. When the program is executed all the CPU has to do is fetch the instructions, and execute them, which in turn can lead to fetching data from memory. Successive versions of CPU's have become more clever in handling this, predicting what might happen, (pre) fetching data from memory etc.

When you look at scripting languages, again a high level syntax is used but after interpretation it becomes compact so called bytecode: a sequence of instructions for a virtual machine that itself is a compiled program. The virtual machine fetches the bytes and acts upon them. It also deals with managing memory and variables. There is not much optimization going on there, certainly not when the language permits dynamically changing function calls and such. Here performance is not only influenced by the virtual machine but also by the quality of the original code (the scripts). In LUATEX we're talking LUA here, a scripting language that is actually considered to be pretty fast.

Sometimes bytecode can be compiled Just In Time into low level machine code but for LUATEX that doesn't work out well. Much LUA code is executed only once or a few times so it simply doesn't pay off. Apart from that there are other limitations with this (in itself impressive) technology so I will not go into more detail.

So how does TEX work? It is important to realize that we have a mix of input and macros. The engine interprets that on the fly. A character enters the input and TEX has to look at it in the perspective of what it what it expects. It is just a character? Is it part of a control sequence that started (normally) with a backslash? Does it have a special meaning, like triggering math mode? When a macro is defined, it gets stored as a linked list of tokens and when it gets called the engine has to expand that meaning. In the process some actions themselves kind of generate input. When that happens a new level of input is entered and further expansion takes place. Sometimes TEX looks ahead

and when not satisfied, pushes something back into the input which again introduces a new level. A lot can happen when a macro gets expanded. If you want to see this, just add `\tracingall` at the top of your file: you will be surprised! You will not see how often tokens get pushed and popped but you can see how much got expanded and how often local changes get restored. By the way, here is something to think about:

```
\count4=123
\advance \count4 by 123
```

If this is in your running text, the scanner sees `\count` and then triggers the code that handles it. That code expects a register number, here that is the `4`. Then it checks if there is an optional `=` which means that it has to look ahead. In the second line it checks for the optional keyword `by`. This optional scanning has a side effect: when the next token is *not* an equal or keyword, it has to push back what it just read (we enter a new input level) and go forward. It then scans a number. That number ends with a space or `\relax` or something not being a number. Again, some push back onto the input can happen. In fact, say that instead of `4` we have a macro indicating the register number, intermediate expansion takes place. So, even these simple lines already involve a lot of action! Now, say that we have this

```
% \newcounter \scratchcounter % done once
\scratchcounter 123
\scratchcounter =123
\advance\scratchcounter by 123
\advance\scratchcounter 123
```

Can you predict what is more efficient? If this operation doesn't happen frequently, performance wise there is no real difference between the variants with and without `=` and with and without `b`. This is because TEX is pretty fast in tokenizing its input and interpreting its already stored token lists that have these commands. But given what we said before, when you talk of millions of such assignments, adding the equal sign and `by` *could* actually be faster because there is no pushing back onto the input stack involved. It probably makes no sense to take this into account when writing macros but just keep in mind that performance is in the details.

Actually, contrary to what you might expect, `\scratchcounter` is not even a counter in ConTEXt, and in LuaMetaTEXwe can also do this:

```
% \newinteger\scratchcounter % done once
\scratchcounter 123
\scratchcounter =123
\advanceby\scratchcounter 123
```

Which means that because this counter is defined as so called "constant integer" it avoids some indirectness (to a counter register) and because `\advanceby` doesn't scan for a keyword the code above runs faster anyway.

This model of expansion is very different from compiled code or bytecode. To some extent you can consider a list of tokens that make up a macro to be bytecode, but instead of a sequence of bytes it is a linked list. That itself has a penalty in performance. Depending on how macros expand, the engine can be hopping all over the token memory following that list. That means that quite likely the data that gets accessed is not in your CPU cache and as a result performance cannot benefit from it apart of course from the expanding machinery itself, but that one is not a simple loop messing around with variables: it accesses code all over the place! Text gets hyphenated, fonts get applied, material gets boxed, paragraphs constructed, pages built. We're not moving a blob of bits around (as in a video) but we're constantly manipulating small amounts of memory scattered around memory space.

Now, where a traditional TEX engine works on 8 bit characters and smaller tokens, the 32 bit LUATEX works on larger chunks. Although macro names are stored as single symbolic units, there are moments when its real (serialized to characters) name is used, for instance when with `\csname`. When that happens, the singular token becomes a list, so for instance the (stored) token `\foo` becomes a temporary three token list (actually four if you also count the initial reference token). Those tree tokens become three characters in a string that then is used in the hash lookup. There are plenty cases where such temporary string variables are allocated and filled. Compare:

```
\def\foo{\hello}
```

Here the macro `\foo` has just a one token reference to `\hello` because that's how a macro reference gets stored. But in

```
\def\foo{\csname hello\endcsname}
```

we have two plus five tokens to access what effectively is `\hello`. Each character token has to be converted to a byte into the assembled string. Now it must be said that in practice this is still pretty fast but when we have longer names and especially when we have UTF8 characters in there it can come at a price. It really depends on how your macro package works and sometimes you just pay the price of progress. Buying a faster machine is then the solution because often we're not talking of extreme performance loss here. And modern CPU's can juggle bytes quite efficiently. Actually, when we go to 64 bit architectures, LUATEX's data structures fit quite well to that. As a side note: when you run a 32 bit binary on a 64 bit architecture there can even be a price being paid for that when you use LUATEX. Just move on!

11.5 Management

Before we can even reach the point that some content becomes typeset, much can happen: the engine has to start up. It is quite common that a macro package uses a memory dump so that macros are not to be parsed each run. In traditional engines hyphenation patterns are stored in the memory dump as well. And some macro packages can put fonts in it. All kind of details, like upper- and lowercase codes can get stored too.

In LuaTeX fonts and patterns are normally kept out of the dump. That dump itself is much larger already because we have 32 bit characters instead of 8 bit so more memory is used. There are also new concepts, like catcode tables that take space. Math is 32 bit too, so more codes related to math are stored. Actually the format is so much larger that LuaTeX compresses it. Anyway, it has an impact on startup time. It is not that much, but when you measure differences on a one page document the overhead in getting LuaTeX up and running will definitely impact the measurement.

The same is true for the backend. A traditional engine uses (normally) Type1 fonts and LuaTeX relies on OpenType. So, the backend has to do more work. The impact is normally only visible when the document is finalized. There can be a slightly larger hickup after the last page. So, when you measure one page performance, it again pollutes the page per second performance.

11.6 Summary

So, to come back to the observation that LuaTeX is slower than pdfTeX. At least for ConTeXt we can safely conclude that indeed pdfTeX is faster when we talk about a standard English document, with TeX ASCII input, where we can do with traditional small fonts, with only some kerning and simple ligatures. But as soon as we deal with for instance XML, have different languages and scripts, have more demanding layouts, use color and images, and maybe even features that we were not aware of and therefore didn't require in former times the LuaTeX engine (and for ConTeXt it's LuaMetaTeX follow up) performs way better than pdfTeX. And how about support for hyper links, protrusion and expansion, tagging for the sake of accessibility, new layers of abstraction, etc. The impact on performance can differ a lot per engine (and probably also per macro package). So, there is no simple answer and explanation for the fact that the observed slow LaTeX run on Japanese text, apart from that we can say: look at the whole picture: we have more complex tokens, nodes, scripts and languages, fonts, macros, demands on the machinery, etc. Maybe it is just the price you are paying for that.

All those TEX's 12

Hans Hagen
Hasselt NL
February 2020

This is about TEX, the program that is used as part of the large suite of resources that make up what we call a 'TEX distribution', which is used to typeset documents. There are many flavors of this program and all end with `tex`. But not everything in a distribution that ends with these three characters is a typesetting program. For instance, `latex` launches the a macro package LATEX, code that feeds the program `tex` to do something useful. Other formats are Plain (no `tex` appended) or CONTEXT (`tex` in the middle. Just take a look at the binary path of the TEX distribution to get an idea. When you see `pdftex` it is the program, when you see `pdflatex` it is the macro package LATEX using the PDFTEX program. You won't find this for CONTEXT as we don't use that model of mixing program names and macro package names.

Here I will discuss the programs, not the macro packages that use them. When you look at a complete TEXLIVE installation, you will see many TEX binaries. (I will use the verbatim names to indicate that we're talking of programs). Of course there is the original `tex`. Then there is its also official extended version `etex`, which is mostly known for adding some more primitives and more registers. There can be `aleph`, which is a stable variant of `omega` meant for handling more complex scripts. When PDF became popular the `pdftex` program popped up: this was the first TEX engine that has a backend built in. Before that you always had to run an additional program to convert the native DVI output of TEX into for instance POSTSCRIPT. Much later, `xetex` showed up, that, like OMEGA, dealt with more complex scripts, but using recent font technologies. Eventually we saw `luatex` enter the landscape, an engine that opened up the internals with the LUA script subsystem; it was basically a follow up on `pdftex` and `aleph`.

The previous paragraph mentions a lot of variants and there are plenty more. For CJK and especially Japanese there are `ptex`, `eptex`, `uptex`, `euptex`. Parallel to `luatex` we have `luajittex` and `luahbtex`. As a follow up on the (presumed stable) `luatex` the CONTEXT community now develops `luametatex`. A not yet mentioned side track is NTS (New TEX system), a rewrite of good old TEX in JAVA, which in the end didn't take off and was never really used.

There are even more TEX's and they came and went. There was `enctex` which added encoding support, there were `emtex` and `hugeemtex` that didn't add functionality but made more possible by removing some limits on memory and such; these were quite important. Then there were vendors of TEX systems that came up with variants (some had extra capabilities), like `microtex`, `pctex`, `yandytex` and `vtex` but they never became part of the public effort.

For sure there are more, and I know this because not so long ago, when I cleaned up some of my archives, I found `eetex` (extended ε-TEX), and suddenly remembered that Taco Hoekwater and I indeed had experimented with some extensions that we had in mind but that never made it into ε-TEX. I had completely forgotten about it, probably because we moved on to LUATEX. It is the reason why I wrap this up here.

In parallel there have been some developments in the graphic counterparts. Knuts `metafont` program got a LUA enhanced cousin `mflua` while `metapost` (aka `mpost` or `mp`) became a library that is embedded in LUATEX (and gets a follow up in LUAMETATEX). I will not discuss these here.

If we look back at all this, we need to keep in mind that originally TEX was made by Don Knuth for typesetting his books. These are in English (although over time due to references he needed to handle different scripts than Latin, be it just snippets and not whole paragraphs). Much development of successors was the result of demands with respect to scripts other than Latin and languages other than English. Given the fact that (at least in my country) English seems to become more dominant (kids use it, universities switch to it) one can wonder if at some point the traditional engine can just serve us as well.

The original `tex` program was actually extended once: support for mixed usage of multiple languages became possible. But apart from that, the standard program has been pretty stable in terms of functionality. Of course, the parts that made the extension interface have seen changes but that was foreseeable. For instance, the file system hooks into the KPSE library and one can execute programs via the `\write` command. Virtual font technology was also an extension but that didn't require a change in the program but involved postprocessing the DVI files.

The first major 'upgrade' was ε-TEX. For quite a while extensions were discussed but at some point the first version became available. For me, once PDFTEX incorporated these extensions, it became the default. So what did it bring? First of all we got more than 256 registers (counters, dimensions, etc.). Then there are some extra primitives, for instance `\protected` that permits the definition of unexpandable macros (although before that one could simulate it at the cost of some overhead) and convenient ways to test the existence of a macro with `\ifdefined` and `\ifcsname`. Although not strictly needed, one could use `\dimexpr` for expressions. A probably seldom used extension was the (paragraph bound) right to left typesetting. That actually is a less large extension than one might imagine: we just signal where the direction changes and the backend deals with the reverse flushing. It was mostly about convenience.

The OMEGA project (later followed up by ALEPH) didn't provide the additional programming related primitives but made the use of wide fonts possible. It did extend the number of registers, just by bumping the limits. As a consequence it was much more demanding with respect to memory. The first time I heard of ε-TEX and OMEGA was at the 1995 euroTEX meeting organized by the NTG and I was sort of surprised by the sometimes emotional clash between the supporters of these two variants. Actually it was the first time I became aware of TEX politics in general, but that is another story.

It was also the time that I realized that practical discussions could be obscured by nitpicking about speaking the right terminology (token, node, primitive, expansion, gut, stomach, etc.) and that one could best keep silent about some issues.

The PDFTEX follow up had quite some impact: as mentioned it had a backend built in, but it also permitted hyperlinks and such by means of additional primitives. It added a couple more, for instance for generating random numbers. But it actually was a research project: the frontend was extended with so called character protrusion (which lets glyphs hang into the margin) and expansion (a way to make the output look better by scaling shapes horizontally). Both these extensions were integrated in the paragraph builder and are thereby extending core code. Adding some primitives to the macro processor is one thing, adapting a very fundamental property of the typesetting machinery is something else. Users could get excited: TEX renders a text even better (of course hardly anyone notices this, even TEX users, as experiments proved).

In the end OMEGA never took off, probably because there was never a really stable version and because at some time XETEX showed up. This variant was first only available on Apple computers because it depends on third party libraries. Later, ports to other systems showed up. Using libraries is not specific for XETEX. For instance PDFTEX uses them for embedding images. But, as that is actually a (backend) extension it is not critical. Using libraries in the frontend is more tricky as it adds a dependency and the whole idea about TEX was that is is independent. The fact that after a while XETEX switched libraries is an indication of this dependency. But, if a user can live with that, it's okay. The same is true for (possibly changing) fonts provided by the operating system. Not all users care too strongly about long term compatibility. In fact, most users work on a document, and once finished store some PDF copy some place and then move on and forget about it.

It must be noted that where ε-TEX has some limited right to left support, OMEGA supports more. That has some more impact on all kinds of calculations in the machinery because when one goes vertical the width is swapped with the height/depth and therefore the progression is calculated differently.

Naturally, in order to deal with scripts other than Latin, XETEX did add some primitives. I must admit that I never looked into those, as CONTEXT only added support for wide fonts. Maybe these extensions were natural for LATEX, but I never saw a reason to adapt the CONTEXT machinery to it, also because some PDFTEX features were lacking in XETEX that CONTEXT assumed to be present (for the kind of usage it is meant for). But we can safely say that the impact of XETEX was that the TEX community became aware that there were new font technologies that were taking over the existing ones used till now. One thing that is worth noticing is that XETEX is still pretty much a traditional TEX engine: it does for instance OPENTYPE math in a traditional TEX way. This is understandable as one realizes that the OPENTYPE math standard was kind of fuzzy for quite a while. A consequence is that for instance the OPENTYPE math fonts produced by the GUST foundation are a kind of hybrid. Later versions adopted some more PDFTEX features like expansion and protrusion.

I skip the Japanese TEX engines because they serve a very specific audience and provide features for scripts that don't hyphenate but use specific spacing and line breaks by injecting glues and penalties. One should keep in mind that before UNICODE all kinds of encodings were used for these scripts and the 256 limitations of traditional TEX were not suited for that. Add to that demands for vertical typesetting and it will be clear that a specialized engine makes sense. It actually fits perfectly in the original idea that one could extend TEX for any purpose. It is a typical example of where one can argue that users should switch to for instance XETEX or LUATEX but these were not available and therefore there is no reason to ditch a good working system just because some new (yet unproven) alternative shows up a while later.

We now arrive at LUATEX. It started as an experiment in 2005 where a LUA interpreter was added to PDFTEX. One could pipe data into the TEX machinery and query some properties, like the values of registers. At some point the project sped up because Idris Hamid got involved. He was one of the few CONTEXT users who used OMEGA (which it actually did support to some extent) but he was not satisfied with the results. His oriental TEX project helped pushing the LUATEX project forward. The idea was that by opening up the internals of TEX we could do things with fonts and paragraph building that were not possible before. The alternative, XETEX was not suitable for him as it was too bound to what the libraries provides (rendering then depends on what library gets used and what is possible at what time). But, dealing with scripts and fonts is just one aspect of LUATEX. For instance more primitives were added and the math machinery got an additional OPENTYPE code path. Memory constraints were lifted and all became UNICODE internally. Each stage in the typesetting process can be intercepted, overloaded, extended.

Where the ε-TEX and OMEGA extensions were the result of many years of discussion, the PDFTEX, XETEX and LUATEX originate in practical demands. Very small development teams that made fast decisions made that possible.

Let's give some more examples of extensions in LUATEX. Because PDFTEX is the starting point there is protrusion and expansion, but these mechanisms have been promoted to core functionality. The same is true for embedding images and content reuse: these are now core features. This makes it possible to implement them more naturally and efficiently. All the backend related functionality (literal PDF, hyperlinks, etc) is now collected in a few extension primitives and the code is better isolated. This took a bit of effort but is in my opinion better. Support for directions comes from OMEGA and after consulting with its authors it was decided that only four made sense. Here we also promoted the directionality to core features instead of extensions. Because we wanted to serve OMEGA users too extended TFM fonts can be read, not that there are many of them, which fits nicely into the whole machinery going 32 instead of 8 bits. Instead of the ε-TEX register model, where register numbers larger than 255 were implemented differently, we adopted the OMEGA model of just bumping 256 to 65536 (and of course, 16K would have been sufficient too but the additional memory it uses can be neglected compared to what other programs use and/or what resources users carry on their machines).

The modus operandi for extending TEX is to take the original literate WEB sources and define change files. The PDFTEX program already deviated from that by using a monolithic source. But still PASCAL is used for the body of core code. It gets translated to C before being compiled. In the LUATEX project Taco Hoekwater took that converted code and laid the foundation for what became the original LUATEX code base.

Some extensions relate to the fact that we have LUA and have access to TEX's internal node lists for manipulations. An example is the concept of attributes. By setting an attribute to a value, the current nodes (glyphs, kerns, glue, penalties, boxes, etc) get these as properties and one can query them at the LUA end. This basically permits variables to travel with nodes and act accordingly. One can for instance implement color support this way. Instead of injecting literal or special nodes that themselves can interfere we now can have information that does not interfere at all (apart from maybe some performance hit). I think that conceptually this is pretty nice.

At the LUA one has access to the TEX internals but one can also use specific token scanners to fetch information from the input streams. In principle one can create new primitives this way. It is always a chicken-egg question what works better but the possibility is there. There are many such conceptual additions in LUATEX, which for sure makes it the most 'aggressive' extension of TEX so far. One reason for these experiments and extensions is that LUA is such a nice and suitable language for this purpose.

Of course a fundamental part of LUATEX is the embedded METAPOST library. For sure the fact that CONTEXT integrates METAPOST has been the main reason for that.

The CONTEXT macro package is well adapted to LUATEX and the fact that its users are always willing to update made the development of LUATEX possible. However, we are now in a stage that other macro packages use it so LUATEX has entered a state where nothing more gets added. The LATEX macro package now also supports LUATEX, although it uses a variant that falls back on a library to deal with fonts (like XETEX does).

With LUATEX being frozen (of course bugs will be fixed), further exploration and development is now moved to LUAMETATEX, again in the perspective of CONTEXT. I will not go into details apart from saying that is is a lightweight version of LUATEX. More is delegated to LUA, which already happened in CONTEXT anyway, but also some extra primitives were added, mostly to enable writing nicer looking code. However, a major aspect is that this program uses a lean and mean code base, is supposed to compile out of the box, and that sources will be an integral part of the CONTEXT code base, so that users are always in sync.

So, to summarize: we started with `tex` and moved on to `etex` and `pdftex`. At some point `omega` and `xetex` filled the UNICODE and script gaps, but it now looks like `luatex` is becoming popular. Although `luatex` is the reference implementation, LATEX exclusively uses `luahbtex`, while CONTEXT has a version that targets at `luametatex`. In parallel, the `[e][u][p]tex` engines fill the specific needs for Japanese users. In most cases, good old `tex` and less old `etex` are just shortcuts to `pdftex` which is compatible

but has the PDF backend on board. That 8 bit engine is not only faster than the more recent engines, but also suits quite well for a large audience, simply because for articles, thesis, etc. (written in a Latin script, most often English) it fits the bill well.

I deliberately didn't mention names and years as well as detailed pros and cons. A user should have the freedom to choose what suits best. I'm not sure how well TEX would have evolved or will evolve in these days of polarized views on operating systems, changing popularity of languages, many (also open source) projects being set up to eventually be monetized. We live in a time where so called influencers play a role, where experience and age often matters less than being fancy or able to target audiences. Where something called a standard today is replaced quickly by a new one tomorrow. Where stability and long term usage of a program is only a valid argument for a few. Where one can read claims that one should use this or that because it is todays fashion instead of the older thing that was the actually the only way to achieve something at all a while ago. Where a presence on facebook, twitter, instagram, whatsapp, stack exchange is also an indication of being around at all. Where hits, likes, badges, bounties all play a role in competing and self promotion. Where today's standards are tomorrow's drawbacks. Where even in the TEX community politics seem to creep in. Maybe you can best not tell what is your favorite TEX engine because what is hip today makes you look out of place tomorrow.

Hidden treasures 13

Hans Hagen
Hasselt 2020
February 2020

At CONTEXT meetings we always find our moments to reflect on the interesting things that relate to TEX that we have run into. Among those we discussed were some of the historic treasures one can run into when one looks at source files. I will show examples from several domains in the ecosystem and we hereby invite the reader to come up with other interesting observations, not so much in order to criticize the fantastic open source efforts related to TEX, but just to indicate how decades of development and usage are reflected in the code base and usage, if only to make it part of the history of computing.

I start with the plain TEX format. At the top of that file we run into this:

```
% The following changes define internal codes as recommended
% in Appendix C of The TeXbook:

\mathcode`\^^@="2201 % \cdot
\mathcode`\^^A="3223 % \downarrow
\mathcode`\^^B="010B % \alpha
\mathcode`\^^C="010C % \beta
\mathcode`\^^D="225E % \land
\mathcode`\^^E="023A % \lnot
\mathcode`\^^F="3232 % \in
...
\mathcode`\^^Y="3221 % \rightarrow
\mathcode`\^^Z="8000 % \ne
\mathcode`\^^[="2205 % \diamond
\mathcode`\^^\="3214 % \le
\mathcode`\^^]="3215 % \ge
\mathcode`\^^^="3211 % \equiv
\mathcode`\^^_="225F % \lor
```

This means that when you manage to key in one of these recommended character codes that in ASCII sits below the space slot, you will get some math symbol, given that you are in math mode. Now, if you also consider that the plain TEX format is pretty compact and that no bytes are wasted,[8] you might wonder what these lines do there. The answer is simple: there were keyboards out there that had these symbols. But, by the time TEX became popular, the dominance of the IBM keyboard let those memories fade away.

[8] Such definitions don't take additional space in the format file.

This is just Don's personal touch I guess. Of course the question remains if the sources of TAOCP contain these characters.

There is another interesting hack in the plain TEX file, one that actually, when I first looked at the file, didn't immediately made sense to me.

```
\font\preloaded=cmti9
\font\preloaded=cmti8
\font\preloaded=cmti7

\let\preloaded=\undefined
```

What happens here is that a bunch of fonts get defined and they all use the same name. Then eventually that name gets nilled. The reason that these definitions are there is that when TEX dumps a format file, the information that comes from those fonts is embedded to (dimensions, ligatures, kerns, parameters and math related) data. It is an indication that in those days it was more efficient to have them preloaded (that is why they use that name) than loading them at runtime. The fonts are loaded but you can only access them when you define them again! Of course nowadays that makes less sense, especially because storage is fast and operating systems do a nice job at caching files in memory so that successive runs have font files available already.

Talking of fonts, one of the things a new TEX user will notice and also one of the things users love to brag about is ligatures. If you run the `tftopl` program on a file like `cmr10.tfm` you will get a verbose representation of the font. Here are some lines:

```
(LABEL C f)  (LIG C i O 14) (LIG C f O 13) (LIG C l O 15)
(LABEL O 13) (LIG C i O 16) (LIG C l O 17)
(LABEL C `)  (LIG C ` C \)
(LABEL C ')  (LIG C ' C ")
(LABEL C –)  (LIG C - C {)
(LABEL C {)  (LIG C - C |)
(LABEL C !)  (LIG C ` C <)
(LABEL C ?)  (LIG C ` C >)
```

The `C` is followed by an ASCII representation and the `)` by the position in the font `O` (a number) or `C` (a character). So, consider the first two lines to be a puzzle: they define the fi, ff, fl ligatures as well as the ffi and ffl ones. Do you see how ligatures are chained?

But anyway, what do these other lines do there? It looks like ``` `` ``` becomes the character in the backslash slot and `''` the one in the double quote. Keep in mind that TEX treats the backslash special and when you want it, it will be taken from elsewhere. But still, these two ligatures look familiar: they point to slots that have the left and right double quotes.[9] They are not really ligatures but abuse the ligature mechanism to achieve

[9] CONTEXT never assumed this and encourages users to use the quotation macros. Those ``` ``quotes'' ``` look horrible in a source anyway.

a similar effect. The last four lines are the most interesting: these are ligatures that (probably) no TEX user ever uses or encounters. They are again something from the past. Also, changes are low that you mistakenly enter these sequences and the follow up Latin Modern fonts don't have them anyway.

Actually, if you look at the METAFONTand METAPOST sources you can find lines like these (here we took from `mp.w` in the LUATEX repository):

```
@ @<Put each...@>=
mp_primitive (mp, "=:", mp_lig_kern_token, 0);
@:=:_}{\.{=:} primitive@>;
mp_primitive (mp, "=:|", mp_lig_kern_token, 1);
@:=:/_}{\.{=:\char'174} primitive@>;
mp_primitive (mp, "=:|>", mp_lig_kern_token, 5);
@:=:/>_}{\.{=:\char'174>} primitive@>;
mp_primitive (mp, "|=:", mp_lig_kern_token, 2);
@:=:/_}{\.{\char'174=:} primitive@>;
mp_primitive (mp, "|=:>", mp_lig_kern_token, 6);
@:=:/>_}{\.{\char'174=:>} primitive@>;
mp_primitive (mp, "|=:|", mp_lig_kern_token, 3);
@:=:/_}{\.{\char'174=:\char'174} primitive@>;
mp_primitive (mp, "|=:|>", mp_lig_kern_token, 7);
@:=:/>_}{\.{\char'174=:\char'174>} primitive@>;
mp_primitive (mp, "|=:|>>", mp_lig_kern_token, 11);
@:=:/>_}{\.{\char'174=:\char'174>>} primitive@>;
```

I won't explain what happens there (as I would have to reread the relevant sections of TEX The Program) but the magic is in the special sequences: `=: =:| =:|> |=: |=:> |=:| |=:|> |=:|>>`. Similar sequences are used in some font related files. I bet that most METAPOST users never entered these as they relate to defining ligatures for fonts. Most users know that combining a `f` and `i` gives a `fi` but there are other ways to combine too. One can praise today's capabilities of OPENTYPE ligature building but TEX was not stupid either! But these options were never really used and this treasure will stay hidden. Actually, to come back to a previous remark about abusing the ligature mechanism: OPENTYPE fonts are just as sloppy as TEX with the quotes: there a ligature is just a name for a multiple-to-one mapping which is not always the same as a ligature.

But there are even more surprises with fonts. When Alan Braslau and I redid the bibliography subsystem of CONTEXT with help from LUA, I wrote a converter in that language. I actually did that the way I normally do: look at a file (in this case a BIBTEX file) and write a parser from scratch. However, at some point we wondered how exactly strings got concatenated so I decided to locate the source and look at it there. When I scrolled down I noticed a peculiar section:

```
@^character set dependencies@>
@^system dependencies@>
Now we initialize the system-dependent |char_width| array, for which
```

```
|space| is the only |white_space| character given a nonzero printing
width.  The widths here are taken from Stanford's June~'87
$cmr10$~font and represent hundredths of a point (rounded), but since
they're used only for relative comparisons, the units have no meaning.

@d ss_width = 500        {character |@'31|'s width in the $cmr10$ font}
@d ae_width = 722        {character |@'32|'s width in the $cmr10$ font}
@d oe_width = 778        {character |@'33|'s width in the $cmr10$ font}
@d upper_ae_width = 903  {character |@'35|'s width in the $cmr10$ font}
@d upper_oe_width = 1014 {character |@'36|'s width in the $cmr10$ font}

@<Set initial values of key variables@>=
for i:=0 to @'177 do char_width[i] := 0;
@#
char_width[@'40] := 278;
char_width[@'41] := 278;
char_width[@'42] := 500;
char_width[@'43] := 833;
char_width[@'44] := 500;
char_width[@'45] := 833;
```

Do you see what happens here? There are hard coded font metrics in there! As far as I can tell, these are used in order to guess the width of the margin for references. Of course that won't work well in practice, simply because fonts differ. But given that the majority of documents that need references are using Computer Modern fonts, it actually might work well, especially with Plain TEX because that is also hardwired for 10pt fonts. Personally I'd go for a multipass analysis (or maybe would have had BIBTEX produce a list of those labels for the purpose of analysis but for sure at that time any extra pass was costly in terms of performance). That code stays around of course. It makes for some nice deduction by historians in the future.

I bet that one can also find weird or unexpected code in CONTEXT, and definitely on the machines of TEX users all around the world. For instance, now that most people use UTF8 all those encoding related hacks have become history. On the other hand, as history tends to cycle, bitmap symbolic fonts suddenly can look modern in a time when emoji are often bitmaps. We should guard our treasures.

Don't use TEX! 14

Occasionally I run into a web post that involves LUATEX and it is sometimes surprising what nonsense one can read. Now, I normally don't care that much but there are cases where I feel that a comment is justified. Partly because I'm one of the developers, but also because I'm involved in user groups.

In this particular case the title of a (small) blog post was *"Why I do not like luaTeX!"* and the website announced itself ambitiously as *'DIGITAL TYPOGRAPHY NEWS'*. Normally I assume that in such a case it is a general site about typesetting and that the author has not much experience or insight in the already ancient TEX typesetting system. However, the URL is:

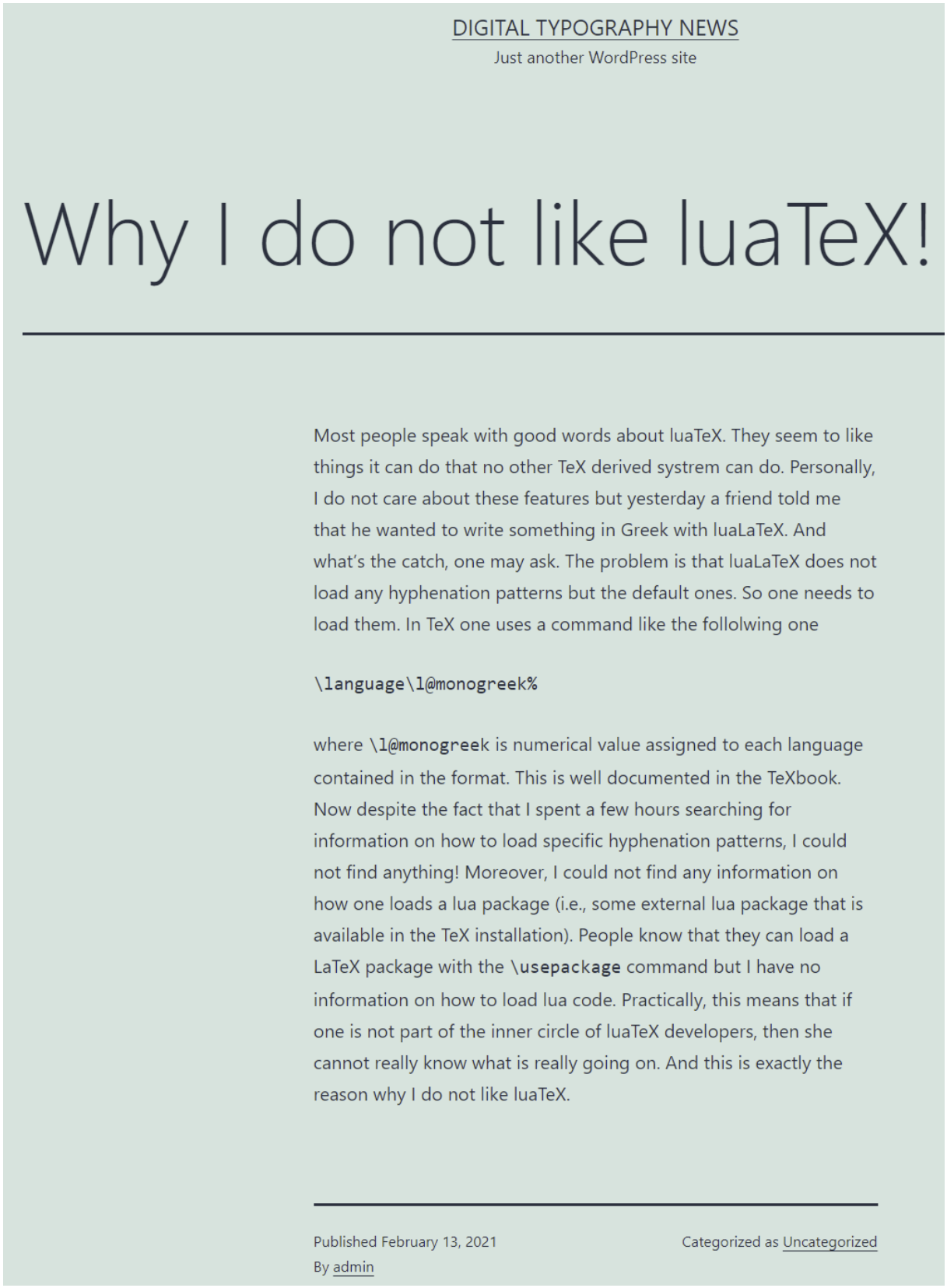
`eutypon.gr/e-blog/index.php/2021/02/13/why-i-do-not-like-luatex/`

which happens to be the Greek User Groups portal. So why do I feel the need to reflect on this? Why do I even care? The answer is simple: because user groups should inform their (potential) users correctly. Another reason is that I'm involved in the program that is disliked, and yet another one is that there is a suggestion that language support is bad in LUATEX, while actually hyphenation patterns are very well maintained by Mojca and Arthur who are also actively involved in the community around the mentioned engine.

Let's start with the title. For sure one might not like a specific program, but when it involves one of the mainstream TEX engines, it should at least be clear that it's a personal opinion. Because no name is mentioned, we can assume that this is the opinion of the Greek user group as a whole. The text starts with *"Most people speak with good words about luaTeX."* and the *'most'* in that sentence sort of puts the author in a small group, which should trigger using a bit more careful title. Now I know a couple of users who use LUATEX (with CONTEXT) for typesetting Greek, and we can assume that they are among the people who speak those good words: typesetting Greek just works.

More good news is that *"They seem to like things it can do that no other TeX derived systrem can do."* This might invite potential users to take a closer look at the system, especially because we already know that most people are positive. In 2021 one should keep in mind that, although the LUATEX engine is around for more than a decade, the level of support can differ per macro package which is why PDFTEX is still the most widespread used TEX variant: much TEX usage relates to writing (scientific articles) in English so one doesn't really need an UNICODE engine. I always say: don't change a good working workflow if you have no reason; use what makes you feel comfortable. Only use LUATEX if you have a reason. There is plenty of good and positive advice to be given.

With *"Personally, I do not care about these features but yesterday a friend told me that he wanted to write something in Greek with luaLaTeX."* the author steps over his or her personal rejection of the engine and enters the help-a-friend mode. *"And what's the catch, one may*



DIGITAL TYPOGRAPHY NEWS

Just another WordPress site

Why I do not like luaTeX!

Most people speak with good words about luaTeX. They seem to like things it can do that no other TeX derived systrem can do. Personally, I do not care about these features but yesterday a friend told me that he wanted to write something in Greek with luaLaTeX. And what's the catch, one may ask. The problem is that luaLaTeX does not load any hyphenation patterns but the default ones. So one needs to load them. In TeX one uses a command like the followlwing one

```
\language\l@monogreek%
```

where `\l@monogreek` is numerical value assigned to each language contained in the format. This is well documented in the TeXbook. Now despite the fact that I spent a few hours searching for information on how to load specific hyphenation patterns, I could not find anything! Moreover, I could not find any information on how one loads a lua package (i.e., some external lua package that is available in the TeX installation). People know that they can load a LaTeX package with the `\usepackage` command but I have no information on how to load lua code. Practically, this means that if one is not part of the inner circle of luaTeX developers, then she cannot really know what is really going on. And this is exactly the reason why I do not like luaTeX.

Published February 13, 2021
By admin

Categorized as Uncategorized

eutypon.gr/e-blog/index.php/2021/02/13/why-i-do-not-like-luatex/

ask. The problem is that luaLaTeX does not load any hyphenation patterns but the default ones. So one needs to load them." I'm not sure why this is a catch. It actually is a feature. One drawback of the traditional TEX engines is that one needs to preload the hyphenation patterns. Before memory was bumped, that often meant creating format files for a subset of languages, and when memory became plenty it meant preloading dozens of patterns by default. The good news is that in all these cases the macro package takes care of that. In the case of LUATEX no patterns need to be preloaded so it might even be that LATEX doesn't have any preloaded but, not being a user, I didn't check that.

This all makes the next sentence puzzling: *"In TeX one uses a command like the fololwing one : `\language\l@monogreek`, where `\l@monogreek` is numerical value assigned to each language contained in the format."* Now, I'm no expert on LATEX but I'm pretty sure that the `@` sign is not a letter by default. I'm also pretty sure that there is some high level interface to enable a language, and in the case of LUATEX being used that mechanism will load the patterns runtime. I bet it will also deal with making sure other language specific properties are set. Therefore the *"This is well documented in the TeXbook."* is somewhat weird: original TEX only had one language and later versions could deal with more, but plain TEX has no `\l@monogreek` command. It doesn't sound like the best advice to me.

Just to be sure, I unpacked all the archives in the most recent TEXLIVE DVD and grepped for that command in `tex` and `sty` files and surprise: in the LATEX specific style file `/tex/xelatex/xgreek/xgreek.sty` there is a line `\language\l@monogreek\else\HyphenRules{monogreek}\fi` which to me looks way to low level for common users to figure out, let alone that it's a file for XETEX so bound to a specific engine. Further grepping for `{greek}` gave hits for LATEX's babel an there are Greek files under the `polyglossia` directory so I bet that Arthur (who once told me he was reponsible for languages) deals with Greek there. Even I, as a CONTEXT user who never use LATEX and only know some things by rumor (like the fact that there is something like polyglossia at all) could help a new user with some suggestions of where to look, just by googling for a solution. But explicitly using the `\language` primitive is not one of them. Okay, in CONTEXT the `\language[greek]` command does something useful, but we're not talking about that package here, if only because it relates to LUATEX development, which as we will see later is a kind of inner circle.

So, picking up on the blog post, in an attempt to get Greek working in LATEX the author got online but *"Now despite the fact that I spent a few hours searching for information on how to load specific hyphenation patterns, I could not find anything!"* It might have helped to search for `lualatex greek` because that gives plenty of hits. And maybe there are even manuals out there that explain which of the packages in the TEX tree to load in order to get it working. Maybe searching CTAN or TEXLIVE helps too. Maybe other user groups have experts who can help out. No matter what you run into, I don't think that the average user expects to find a recipe for installing and invoking patterns. Just for the record, the LUATEX manual has a whole chapter on language support, but again, users can safely assume that the macro package that they use hides those details. Actually, if users were supposed to load patterns using a unique id, they are likely to end up in the modern Greek versus ancient Greek, as well as Greek mixed with English or other

languages situations. That demands some more in depth knowledge to deal with, in any macro package and with any engine. You can add a bit of UNICODE and UTF-8 or encodings in the mix too. Suggesting to consult the TEXbook is even a bit dangerous because one then also ends up in an eight bit universe where font encodings play a role, while LUATEX is an UNICODE engine that expects UTF and uses OPENTYPE fonts. And, while languages seem to be a problem for the author and his/her friend, fonts seem to be an easy deal. In my experience it's more likely that a user runs into font issues because modern fonts operate on multiple axis: script, language and features.

Maybe the confusion (or at that time accumulated frustration) is best summarized by *"Moreover, I could not find any information on how one loads a lua package (i.e., some external lua package that is available in the TeX installation)."* Well, again I'm sure that one can find some information on LATEX support sites but as I already said: language support is so basic in a macro package that users can use some simple command to enable their favorite one. So, when *"People know that they can load a LaTeX package with the* `\usepackage` *command but I have no information on how to load lua code."* the first part is what matters: LUA files are often part of a package and thereby they get loaded by the package, also because often stand-alone usage makes not much sense.

It is absolutely no problem if someone doesn't like (or maybe even hates) LUATEX, but it's a different matter when we end up in disinformation, and even worse in comments that smell like conspiracy: there is an inner circle of LUATEX developers and *"Practically, this means that if one is not part of the inner circle of luaTeX developers, then she cannot really know what is really going on."* Really? Is this how user groups educate their users? There are manuals written, plenty of articles published, active mailing lists, presentations given, and there is support on platforms like Stack Exchange. And most of that (the development of LUATEX included) is done by volunteers in their spare time, for free. Of course the groups of core developers are small but that is true for any development. History (in the TEX community) has demonstrated that this is the only way to make progress at all, simply because there are too many different views on matters, and also because the time of volunteers is limited. It is the end result what counts and when that is properly embedded in the community all is fine. So we have some different engines like TEX, PDFTEX, LUATEX, etc., different macro packages, specialized engines like those dealing with large CJK fonts, all serving a different audience from the same ecosystem. Are these all secretive inner circles with bad intentions to confuse users?

The blog post ends with *"And this is exactly the reason why I do not like luaTeX."* to which I can only comment that I already long ago decided not to waste any time on users who in their comments sound like they were forced to use a TEX system (and seem to dislike it, so probably are better off with Microsoft Word, but nevertheless like to bark against some specific TEX tree), who complain about manuals not realizing that their own contributions might be rather minimalistic, maybe even counter productive, or possibly of not much use to potential users anyway. I also ignore those who love to brag about the many bugs, any small number suits that criterium, without ever mentioning how bugged their own stuff is, etc. If your ego grows by disregarding something you don't even use, it's fine for me.

So why do I bother writing this? Because I think it is a very bad move and signal of a user group to mix personal dislike, whatever the reason is, with informing and educating users. If a group is that frustrated with developments, it should resolve itself. On the other hand, it fits well in how todays communication works: everyone is a specialist, which get confirmed by the fact that many publish (also on topics they should stay away from) on the web without fact checking, and where likes and page hits are interpreted as a confirmation of one's expertise. Even for the TEX community there seems to be no escaping from this.

The objectives of TEX user groups shift, simply because users can find information and help online instead of at meetings and in journals. The physical TEX distributions get replaced by fast downloads but they are definitely under control of able packagers. Maybe a new task of user groups is to act as guardian against disinformation. Of course one then has to run into these nonsense blogs (or comments on forums) and such but that can partly be solved by a mechanism where readers can report this. A user group can then try to make its own information better. However, we have a problem when user groups themselves are the source of disinformation. I see no easy way out of this. We can only hope that such a port drowns in the ocean of information that is already out there to confuse users. In the end a good and able TEX friend is all you need to get going, right? The blog post leaves it open if the Greek text ever got typeset well. If not, there's always CONTEXT to consider, but then one eventually ends up with LUAMETA-TEX which might work on the author as another *"rode lap op een stier"* as we say in Dutch.

Speeding up TEX 15

15.1 Introduction

Recently a couple of cordless phones that I use gave up as soon as I used them for a minute or so. The first time that happened I figured that after all these years the batteries had gone bad and after some testing I decided to replace them. I got some of these high end batteries that discharge slowly and store a lot of power. Within a year they were dead too. Then I went for the more regular and cheaper ones, again with a lot of capacity. And yes, these also gave up, that is: only in the phones that were hardly used. The batteries lasted longer in phones that were discharged by usage daily.

When I went out for new batteries I was asked if I needed them for cordless phones and, surprise, was given special ones that actually stored less but were guaranteed to work for at least 6 years. The package explicitly mentioned use in cordless phones. So here performance doesn't come with the most high end ones, based on specifications that impress.

This is also true for computers that are used to process TEX documents. More cores amount to much accumulated processing power but for a single core TEX process, a few fast cores are more relevant than plenty slower ones that run in parallel. More memory helps but compared to other processes TEX actually doesn't use that much memory. And disk speed matters but less so when the operating system caches files. What does play a role are cpu caches because TEX is very memory intense and processing is not concentrated in a few functions. But a large cache shared among many (busy) cores makes for a less impressive performance.

So what matters really? In the next sections we will explore a few points of view. It's not some advertisement for a specific engine, but much more about putting it into perspective (as one can run into ridiculous arguments on the web). It is not only the hardware and software that matters but also how one uses it.

15.2 The engine

There are various ways to compare engines and each has its own characteristics. The PDFTEX engine is closest to the original. It directly produces the output which can give it an edge. It is eight bit and therefore uses small fonts and internally all that is related to fonts and characters is also small. This means that there is little overhead in typesetting a paragraph: hyphenation, ligature building and kerning are interwoven and perform well.

The XETEX engine supports wide fonts and UNICODE and therefore can be seen as 32 bit. I never looked into the code so I can't tell how far that goes but performance is

definitely less than PDFTEX. The rendering of text is delegated to a library (there were some changes in that along its development) which is less efficient than the built in PDFTEX route. But it is also more powerful.

The LUATEX engine is mostly 32 bit and delegates non standard font handling to LUA which comes with a performance penalty but also adds a lot of flexibility. Also, the fact that one can call out to LUA in many places makes that one can not really blame the engine for performance hits. The fact that hyphenation, ligature building and kerning is split comes at a small price too. We have larger nodes so compared to PDFTEX more memory is used and accessed. Some mechanisms are actually more efficient, like font expansion and protrusion.

The LUAMETATEX engine lacks a font loader (but it does have the traditional renderer on board) and it has no backend. So even more is delegated to LUA, which in turn makes this the slowest of the lot. And, again more data travels with nodes. In some modes of operation much more calculations take place. However, because it has an enriched macro processor, additional primitives, and plenty deep down '*improvements*' it can perform better than LUATEX (and even LUAJITTEX, the LUATEX version with a faster but limited LUA virtual machine). And as with LUATEX, there are usage patterns that make it faster than PDFTEX.

So, in general the order of performance is PDFTEX, XETEX, LUAJITTEX (kind of obsolete), LUATEX, LUAMETATEX. But then, how come that CONTEXT users never complain about performance? The reasons is simple: performance is quite okay and as it is relative to what one does, a user will accept a drop in performance when more has to be done. When we moved on from LUATEX to LUAMETATEX there definitely was a drop in performance, simply because of the LUA backend. Because upgrading happened in small (but continuous) steps, right from the start the new engine was good enough to be used in production which is why most users switched to LMTX as soon as became clear that this is where the progress is made.

There were no real complaints about the upto 15% initial performance drop which indicates that for most users it doesn't matter that much. As the engine evolved we could gain some back and now LUAMETATEX ends up between PDFTEX and LUATEX and in many modern scenarios even comes out first. The fact that in the meantime we can be much faster than LUATEX did get noticed (when asked). However, as development takes years updating a machine in the meantime puts discussions about performance in a different (causality) perspective anyway.

15.3 The coding

Performance can increase when native engine features are used instead of complex macros that have to work around limitations. It can also decrease when new features are used that add complex functionality. And when an engine extends existing functionality that is likely to come at a price. So where LUAMETATEX provides a more rich

programming environment, it also had a more complex par builder, page builder, insert, mark and adjust handling, plenty of extra character, rule and box features and all of that definitely adds some overhead. Quite often a gain in simplicity (nicer and more efficient macros) compensate the more complex features. That is because on the average the engine doesn't do that much (tens of thousands of the same) complex macro expansion and also doesn't demand that much complex low level typesetting. A gain here is often compensated by a loss there. This is one reason why during the years LuaMetaTeX could sustain a decent performance. Personally I don't accept a drop in performance easily which is why in practice most mechanism, even when extended, probably perform better but I'm not going to prove that observation.

One important reason why ConTeXt LMTX with LuaMetaTeX is faster than its ancestors is that we got rid of some intermediate programming layers. Most users have never seen the auxiliary macros or implementation details but plenty were used in MkII and MkIV. Of course we kept them because often they are nicer than many lines of primitive code, but only a few (and less in the future) are used in the core. Examples are multi step macros (that pick up arguments) that became single step and complex if tests that became inline native tests. Because ConTeXt always had a high level of abstraction consistency of the interface also makes that we don't need many helpers. When some features (like for instance box manipulation) got extended one could expect a performance hit due to more extensive optional keyword scanning in the engine but that was compensated by improved scanners. The same is true for scanning numbers and dimensions. So, more functionality doesn't always come at a price.

To summarize this: although the engine went a bit more *'cisc'* than `risc` the macro package went more *'risc'*. It reminds me a bit of the end of the previous century when there was much talk of fourth generation languages, something on top of the normal languages. In the end it were scripting languages that became the fashion while traditional languages like C remained relatively stable and unchanged for implementing them (and more). A similar observation can be made for ConTeXt itself. Whenever some new feature gets added to an existing mechanism I try to not cripple performance and thanks to the way ConTeXt is set up it works out okay.

Let's look at an example. In MkII we can compare two *'strings'* with the macro `doifelse`. Its definition is as follows:

```
\long\def\doifelse#1#2%
  {\let\donottest\dontprocesstest
   \edef\!!stringa{#1}%
   \edef\!!stringb{#2}%
   \let\donottest\doprocesstest
   \ifx\!!stringa\!!stringb
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

This macro takes two arguments that gets expanded inside two helpers that we then compare with a primitive \ifx. Depending on the outcome we expand one of the two following arguments but first we get rid of the interfering \else and \fi. The pushing and popping of \donottest takes care of protection of unwanted expansion in an \edef. Many functional macros are what we call protected: then expand in two steps depending on the embedded \donottest macro. Think of (simplified):

```
\def\realfoo{something is done here}
\def\usedfoo{\donottest\realfoo}
```

Normally \donottest is doing nothing so \realfoo gets expanded but there are cases where we (for instance) \let it be \string which then serializes the macro. This is something that happens when writing to the multi pass data file. It can also be used for overloading, for instance in the backend or when converting something. This protection against expansion has always been a ConTeXt feature, which in turn made it pretty robust in multi pass scenarios, but it definitely came with performance penalty.

When pdfTeX got the ε-TeX extensions we could use the \protected prefix to replace this trickery. That means that MkII will use a different definition of \doifelse when that primitive is known:

```
\long\def\doifelse#1#2%
  {\edef\!!stringa{#1}%
   \edef\!!stringb{#2}%
   \ifx\!!stringa\!!stringb
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

This works okay because we now do this:

```
\protected\def\usedfoo{something is done here}
```

The \doifelse helper itself is not protected in MkII (non ε-TeX mode) It would be a performance hit. I won't bore the reader with the tricks needed to do the opposite, that is: expand a protected macro. It is seldom needed anyway.

The MkIV definition used with LuaTeX is not much different, only the \long prefix is missing. That one is needed when one wants #1 and/or #2 to be tolerant with respect to embedded \par equivalents. In LuaMetaTeX we can disable that check and in ConTeXt all macros are thereby \long. Users won't notice because in ConTeXt most macros were always defined the long way; we also suppress \outer errors.

```
\protected\def\doifelse#1#2%
  {\edef\m_syst_string_one{#1}%
   \edef\m_syst_string_two{#2}%
   \ifx\m_syst_string_one\m_syst_string_two
```

```
    \expandafter\firstoftwoarguments
  \else
    \expandafter\secondoftwoarguments
  \fi}
```

Implementation wise a macro, once scanned and stored, carries the long property in its command code so that has overhead. However because LuaTEX is compatible we cannot make all normal macros long by default when \suppresslongerror is used. Therefore checking for an argument running into a \par is still checked but the message is suppressed based on the setting of the mentioned parameter. Performance wise, not using \long comes a the cost of checking a parameter which means an additional memory access and comparison. Unless we otherwise gain something in the engine it comes at a cost. In LuaMetaTEX the \long and \outer prefixes are ignored. Even better, protected macros are also implemented a bit more efficiently.

In the end the definition of \doifelse in LMTX looks a bit different:

```
\permanent\protected\def\doifelse#1#2%
  {\iftok{#1}{#2}%
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

The \permanent prefix flags this macro as such. Depending on the value of \overloadmode a redefinition is permitted, comes with a warning or results in a fatal error. Of course this comes at a price when we define macros or values of quantities but this is rather well compensated by all kind of improvements in handling macros: defining, expansion, saving and restoring, etc.

More interesting is the use of \iftok here. It saves us defining two helper macros. Of course the content still needs to be expanded before comparison but we no longer have various macro management overhead. In scenarios where we don't need to jump over the \else or \fi we can use this test in place which saves passing two arguments and grabbing one argument later on. Actually, grabbing is also different, compare:

```
          \def\firstoftwoarguments #1#2{#1} % MkII and MkIV
\permanent\def\firstoftwoarguments #1#-{#1} % MkXL aka LMTX

          \def\secondoftwoarguments#1#2{#1} % MkII and MkIV
\permanent\def\secondoftwoarguments#-#1{#1} % MkXL aka LMTX
```

In the case of LuaMetaTEX the #- makes that we don't even bother to store the argument as it is ignored. Where #0 does the same it also increments the argument counter which is why here even the second arguments has number 1. Now, if this more efficient? Sure, but how often does it really happen? The engine still needs to scan (which comes at a cost) but we save on temporary token list storage. Because TEX is so fast

already, measuring only shows differences when one has many (and here a real lot) iterations. However, all these small bits add up which is what we've seen in 2022 in ConTeXt: it is the reason why we are now faster than MkIV with LuaTeX, even with more functionality in the engine.

I can probably write hundreds of pages in explaining what was added, changed, made more flexible and what side effects it had/has on performance but I bet no one is really interested in that. In fact, the previous exploration is just a side effect of a question that triggered it, so maybe future questions will trigger more explanations. It anyhow demonstrates what I meant when I said that LuaMetaTeX is meant to be leaner and meaner. Of course the code base and binary is smaller but that also gets compensated by more functionality. It also means that we can make the ConTeXt code base nicer because for me a good looking source (which of course is subjective) is pretty important.

15.4 Compatibility

There are non ConTeXt users who seem to love to stress that successive versions of ConTeXt are incompatible. Other claims are that it is developed in a commercial setting. While it is true that there are changes and it is also true that ConTeXt is used in commercial settings, it is not that different from other open source projects. The majority of the code is written without compensation and it is offered without advertisements or request for support. It is true that when we can render better, it will be done. But the user interfaces only change when there is a reason and there are few cases where some functionality became obsolete, think of input and font encodings. Most such changes directly relate to the engine: in pdfTeX and MkII we emulate utf-8 wile in LuaTeX is comes natively. In pdfTeX eight bit (Type1) fonts are used while LuaTeX adds support for OpenType. Other macro packages support that by additional packages while ConTeXt has it integrated. That is why the system evolves over time.

Just a users adapt to (yearly) operating system interfaces, mobile phones, all kinds of hardware, cars, clothing, media and so on, the ConTeXt users have no problem adapting to an evolving TeX ecosystem. I guess claims about changes (being a disadvantage) can only point to a lack of development elsewhere. The main reason for mentioning this is that when ConTeXt users move on to newer engines, the older ones are seldom used. So, few users compare a LMTX run with one using pdfTeX or LuaTeX. They naturally expect LuaMetaTeX to perform well and maybe even to perform better over time. They just don't complain. And unless one hacks (overloads) system macros compatibility is not really an issue. What can be an issue is that updates and adaptations to a newer engine come with bugs but those are solved.

So, the fact that we compare incompatible engines with likely different low level macro implementations of otherwise stable features of a macro package makes comparison hard. For instance, maybe there are speedups possible in frozen MkII, although it is unlikely, which makes that it might even perform better than reported. In a similar fashion, the fact that OpenType is more demanding for sure makes that LuaTeX rendering is slower than pdfTeX. It anyhow makes a discussion about performance within

and between macro packages even more ridiculous. Just don't buy those claims and/or ask on the ConTeXt mailing list for clarification.

15.5 The job

So, say that we now have an efficient and powerful engine and a matching macro package. Does that make all jobs faster? For sure, the ones that I use as benchmark run much smoother. The 360 page LuaMetaTeX manual runs in less than 8.4 seconds on a Dell Precision laptop with (mobile) Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz, 2TB fast Samsung pro SSD, and 48 GB of memory, running Windows 10. The MetaFun manual with many more pages and thousands of MetaPost graphics needs a bit more than 12 seconds. So you don't hear me complain. This chapter takes 7.5 seconds plus 0.5 is for the runner, not enough time to get coffee.

Nowadays I tend to measure performance in terms of pages per second, because in the end that is what users experience. For me more important are the gains for my colleague who processes documents of 400 pages from hundreds of small XML files with multiple graphics per page. Given different output variants a lot of processing takes place, so there a gain from 20 pages per second to 25 pages per second is welcome. Anyway, here are a few measurements of a *simple* test suite per January 7, 2023. We use this as test text:

```
\def\Knuth{%%
Thus, I came to the conclusion that the designer of a new system
must not only be the implementer and first large||scale user; the
designer should also write the first user manual.
\par
The separation of any of these four components would have hurt
\TeX\ significantly. If I had not participated fully in all these
activities, literally hundreds of improvements would never have
been made, because I would never have thought of them or perceived
why they were important.
\par
But a system cannot be successful if it is too strongly influenced
by a single person. Once the initial design is complete and fairly
robust, the real test begins as people with many different
viewpoints undertake their own experiments.
}
```

Now keep in mind that these are simple examples. On more complex documents the LuaMetaTeX engine with LMTX is relatively faster: think XML, plenty MetaPost, complex tables, advanced math, dozens of fonts in combination with the new compact font mode.

The tests themselves are simple: we switch fonts (because fonts bring overhead), we add some color (because we use different methods), we process some graphics (to show what embedding MetaPost brings), we do some tables (because that can be stressful).

Each sample is run 50, 500 or 1000 times, and each set is run a couple of times so that we compensate for caching and fluctuating system load. The tests are more about signaling a trend than about absolute numbers. For what it's worth, I used a LUA script to run the samples.

When you run an experiment that measures performance, keep in mind that performance not only depends on the engine, but also on for instance logging. When I run the CONTEXT test suite it takes 1250 seconds if the console takes the full screen on a 2560 by 1600 display and 30 seconds more on a 3840 by 2160 display and it even depends on how large the font is set. On the 1920 by 1200 monitor I get to 1230. Of course these times change when we add more to the test suite so it's always a momentary measurement.

Similar differences can be observed when running in an editor. A good test is making a CONTEXT format: 2.2 seconds goes down to below 1.8 when the output is piped to a file. On a decent 2023 desktop those times are probably half but I don't have one at hand.

sample 1, number of runs: 2

```
\starttext
    \dorecurse {%s} {
        \Knuth
        \par
    }
\stoptext
```

engine	50	500	1000
pdftex	0.63	0.83	1.07
luatex	0.95	1.86	2.94
luametatex	0.61	1.49	2.48

sample 2, number of runs: 2

```
\starttext
    \dorecurse {%s} {
        \tf \Knuth \bf \Knuth
        \it \Knuth \bs \Knuth
        \par
    }
\stoptext
```

engine	50	500	1000
pdftex	0.70	1.73	2.80
luatex	1.37	5.37	9.92
luametatex	1.04	5.06	9.73

sample 3, number of runs: 2

```
\starttext
    \dorecurse {%s} {
        \tf \Knuth \it knuth \bf \Knuth \bs knuth
        \it \Knuth \tf knuth \bs \Knuth \bf knuth
        \par
    }
\stoptext
```

engine	50	500	1000
pdftex	0.71	1.81	2.98
luatex	1.41	5.84	10.77
luametatex	1.05	5.71	10.60

sample 4, number of runs: 2

```
\setupcolors[state=start]
\starttext
    \dorecurse {%s} {
        {\red \tf \Knuth \green \it knuth}
        {\red \bf \Knuth \green \bs knuth}
        {\red \it \Knuth \green \tf knuth}
        {\red \bs \Knuth \green \bf knuth}
        \par
    }
\stoptext
```

engine	50	500	1000
pdftex	0.73	1.91	3.64
luatex	1.39	5.82	12.58
luametatex	1.07	5.57	11.85

sample 5, number of runs: 2

```
\starttext
    \dorecurse {%s} {
        \null \page
    }
\stoptext
```

engine	50	500	1000
pdftex	0.62	1.12	1.68
luatex	0.90	1.39	1.98
luametatex	0.58	0.99	1.46

sample 6, number of runs: 2

```
\starttext
    \dorecurse {%s} {
        %% nothing
    }
\stoptext
```

engine	50	500	1000
pdftex	0.55	0.54	0.56
luatex	0.79	0.81	0.82
luametatex	0.54	0.52	0.53

sample 7, number of runs: 2

```
\starttext
    \dontleavehmode
    \dorecurse {%s} {
        \framed[width=1cm,height=1cm,offset=2mm]{x}
    }
\stoptext
```

engine	50	500	1000
pdftex	0.58	0.65	0.71
luatex	0.84	0.96	1.08
luametatex	0.54	0.62	0.72

sample 8, number of runs: 2

```
\starttext
    \dontleavehmode
    \dorecurse {%s} {
        \framed
          [width=1cm,height=1cm,offset=2mm,
           foregroundstyle=bold,foregroundcolor=red,
           background=color,backgroundcolor=green]
          {x}
    }
\stoptext
```

engine	50	500	1000
pdftex	0.59	0.70	0.83
luatex	0.87	1.00	1.17
luametatex	0.55	0.66	0.78

sample 9, number of runs: 2

```
\starttext
    \ifdefined\permanent\else\def\BC{\NC\bf}\fi
    \dontleavehmode
    \dorecurse {%s} {
        \starttabulate[|||||]
            \NC test \BC test \NC test \NC test \NC \NR
            \NC test \BC test \NC test \NC test \NC \NR
            \NC test \BC test \NC test \NC test \NC \NR
            \NC test \BC test \NC test \NC test \NC \NR
        \stoptabulate
    }
\stoptext
```

engine	50	500	1000
pdftex	0.62	1.15	1.71
luatex	0.94	1.84	2.86
luametatex	0.60	1.19	1.88

sample 10, number of runs: 2

```
\starttext
    \dontleavehmode
    \dorecurse {%s} {
        \startMPcode
            fill fullcircle scaled 1cm withcolor red ;
            fill fullsquare scaled 1cm withcolor green ;
        \stopMPcode
        \space
    }
\stoptext
```

engine	50	500	1000
pdftex	5.73	50.98	102.10
luatex	0.93	1.07	1.30
luametatex	0.57	0.71	0.86

15.6 Final words

Whenever I run into (or get send) remarks of (especially non CONTEXT) users suggesting that LUATEX is much slower than PDFTEX or that LUAMETATEX seems much faster than LUATEX, one really has to keep in mind that this is not always true. Among the questions to be asked are *"What engine do you use?"*, *"Which macro package do you use?"*,

"How well is your style set up?", *"How complex is the document?"*, *"Is your own additional code efficient?"*, *"Do you use engine and macro package features the right way?"* and of course *"What do you compare with?"*, *"What do you expect and why?"*, *"Do you actually know what goes on deep down?"*. An embarrassing one can be *"Do you have an idea what is involved in fulfilling your request given that we use a flexible adaptive macro language?"*. Much probably these questions not get answered properly.

Another thing to make clear is that when someone claims for instance that CONTEXT LMTX is fast because of LUAMETATEX, or that LUAMETATEX is much faster than LUATEX, a healthy suspicion should kick in: does that someone really knows what happens and matters? The previous numbers do show differences for simple cases but we're often not talking of differences that can be used as an excuse for insufficient coding. In the end it is all about the experience: does performance feel in tune with expectations. Which is not to say that I will make CONTEXT and LUAMETATEX faster because after all there are usage scenarios where one has to process tens of thousands of documents with a reasonable amount of time, on regular infrastructure, and of course with as little as possible energy consumption.

If PDFTEX suits your purpose, there is no need to move to LUATEX. As with rechargeable batteries in cordless phones a higher capacity can make things worse. If LUATEX fits the bill, don't dream about using LUAMETATEX instead because it will half runtime because the adaptations needed in the macro package (like adding a backend) might actually slow it down. Moores law doesn't apply to TEX engines and macro packages and you might get disappointed. Accept that the choice you made for a macro package can come with a price.

Quite often it is rather easy to debunk complaints and claims which makes one wonder why claims about perceived or potential are made at all. But then, I'm accustomed to weird remarks and conclusions about CONTEXT as a macro package, or for that matter LUATEX (as it originates in the CONTEXT community) even by people who should know better. Hopefully the above invites to being more careful.

Unicode 16

16.1 Introduction

When working on a TEX macro package for decades one can hardly avoid dealing with math; after all TEX is pretty much about math. When this wonderful typesetting infrastructure was written it was all about quality and how to make your documents look nice. And for sure, Don Knuths documents looks nice, also because he pays a lot of attention to the *"fine points of math typesetting"*.

The constraints of those time (like hardware, compilers, fonts, and for sure also time) made TEX into what it is: eight bit character sets, eight bit fonts, eight bit hyphenation patterns, efficient memory usage and therefore carrying around as little as possible. It all makes sense. But one needs to pay attention.[10]

Math typesetting is actually a sort of separated process in the engine: unprocessed lists go in and after some juggling a list of assembled boxes, glyphs, glues and penalties come out. I will not go into detail about that and only mention that in LUAMETATEX we extended all this to be a bit more flexible and controllable, something that has been driven by the fact that we need to support UNICODE fonts. This is all part of a related effort to move from eight bit '*everything*' to UNICODE '*everywhere*'.

Now, one can say a lot about UNICODE but the main advantage is that it tries to cover '*all*' characters ever encountered, including scripts (used in languages) that are long gone, as well as these little pictures that people like to see on the web: emojis. One can safely say that UNICODE simplifies mixing languages and scripts, and thereby makes TEX macro packages less complex. On the other hand, UNICODE (or more precisely, related wide) fonts makes all kind of features possible and thereby add a complication.

So, how about math? When Don Knuth gave us TEX he also gave us fonts and there are plenty symbols in these fonts. But, as mathematicians seem to love variations on symbols soon more fonts arrived, most noticeably those from the AMS that also added some more alphabets: mathematicians also love to render the shapes of letters differently. In order to access these glyphs names were invented that also sometimes suggested that there was some order in the matter. And, for some reason these names got aliases and soon we had a huge list of often obscure and inconsistent macro names. It didn't take long for a little mess and confusion to creep in.

It has been said that the verbose TEX math ASCII input format is also a way for mathematicians to communicate, just because many use the same tool to render the formulas. Of course that gets obscured when one starts to add additional macros. It gets even

[10] And that is what Mikael Sundqvist and I have been doing a lot since we started upgrading math in CONTEXT in combination with enhancing the math engine in LUAMETATEX. The story here is a byproduct of our explorations and very much a combined effort.

more tricky once we start talking *'standard'* as in *"LaTeX is the standard"*. That has for instance resulted in browsers interpreting TeX like input without using TeX (so how about expansion?). It has also sort of put TeX into the range of possible word processing systems, which in turn leads to these MS WORD versus Google docs versus LaTeX debates that can get rather nasty and unrealistic when it comes to discussing usage and quality. Interestingly, MS WORD now has reasonable math, to some extent modelled after TeX. It has some verbose TeX like (but constrained) input and would do well for probably mostly people who occasionally have to inject some math. There were also attempts by the people at MICROSOFT to normalize the input but we leave that aside now.

However, because we now do have all these symbols and because source code editors make them accessible and show them there is a good chance that users will inject them, if only by cut and paste, so we do have to deal with that. This automatically puts us in the position that we need to deal with different meanings for the same symbol, which in turn might demand different spacing, penalties and such. In the end it is users that drive all this, not publishers; they don't really care and out-source typesetting anyway. We're not aware of any research and development being done and I suppose we would have noticed because after all we're involved in developing LUATEX. It is one of the engines that does OPENTYPE and UNICODE math and no publisher or supplier ever took serious interest in it. From our perspective what users do is visible, everything else is hidden behind corporate curtains. And this is why nowadays we only need to care about users (mainly authors).

Back to typesetting. For a long time all went well: one could typeset documents that looked good. Okay, not all looked good because not everyone paid attention to details, and the more the web evolved the more patching cut'n'paste of bad examples made its way into documents, but let's not start talking quality here. But then came UNICODE and a while later people started talking about accessibility, cutting and pasting and more. In the meantime there had been developments like MATHML and OPENMATH that tried to structure and organize formulas in a more symbolic way.[11]

In the meantime the TeX community had lost the edge on fonts, and OPENTYPE math was invented by MICROSOFT and implemented in MS WORD before a substantial number of TeX users understood what was happening. They had it coming. To a large extend one can say the same about math in UNICODE. Where a Greek capital *'A'* is seen as different from a Latin capital *'A'*, even when they often have the same shape, a math italic variable *'h'* was made synonym to *'Planck constant'*, as if the letters used in math had no meaning at all. We'll see that a wide hat is an extensible character of zero width combining hat accent, which makes for curious handling of the initial character. There is more granularity in some symbols, especially popular symbols like slashes and bars,

[11] It probably went unnoticed that CONTEXT always supported rendering MATHML, and as such had to deal with all the weird aspects (read: way it was used). Although one is not supposed to directly edit MATHML we work with authors who are quite happy to do that simply because they code the documents in XML because there is a need for high quality PDF as well as HTML output and a CONTEXT based workflow can handle the XML well. We're talking of large volumes here (mostly for basically free school math).

than in letters. It is as if the math community didn't care much about how the letters (variables) were communicated and perceived but were picky about the slope of slashes. It seems more of a visual world, which might actually be the reason structured input never really took of. Maybe TEXies just love the mix of characters, commands, spacing directives. Maybe they just love to reposition and space these glyphs to suit all kind of curious non-standard math rendering.

All this makes it pretty hard to communicate meaning, and it is just one of the examples where the TEX community, for as far involved, failed to make a strong case. Our personal opinion is that no one really cared because in the TEX community it is all about rendering. The fact that we use math to communicate only gained attention when accessibility became hot and by then it was too late. Efforts like OPENMATH started ambitious and in the end basically failed. Coding in XML using MATHML isn't much better and one always had to adapt to the latest fashion. Also, once plenty code shows up bugs become features. Browser support came and went and came back. Simplified input using for instance ASCIIMATH started indeed simple but quickly became a (somewhat inconsistent) mess. What we see here is the same as everything web (and computer languages): we can do better, we start some project, then move on, and we end up with half-way abandoned results. The development cycles are short, results have to be achieved fast, there is no time (or interest) for iterating and refactoring. The word *'standard'* and mantra *'everyone should use this'* are quite popular.

So where does that leave us with TEX? Well, with a mess. Decades of various efforts have not brought us a coherent system of organizing symbols and properties, made us end up with inconsistencies, made users revert to hacks, didn't make math easily transferable and complicates rendering. Personally we find it sort of strange that we spend time on for instance tagging and accessibility before we get these math alphabets and shared math specific symbols sorted out. If we cannot make good arguments for that (math being a script on its own with semantics and such) we waste energy and are pulling a dead horse. What puzzles us most is that one would expect mathematicians to be able to come up with strong arguments for a structured approach. But maybe it was simply the fact that TEX math typesetting was pretty much driven by large commercial publishers and those providing services for them: the first category doesn't invest in these matters and even less today, and the second category makes money from sorting out the mess, so why get rid of it. Who knows. For us, it means that any complain about these matters deserves the same answer: the TEX community created this mess, so it has to live with it. And the bad thing is: bugs and work-arounds eventually become features and then one is supposed to conform, even if deep down one knows better. It doesn't help that the community is proud of what it can render and has built itself a reputation that all is good.

So why this criticism? Why not just abandon TEX? The answer is simple: TEX is quite okay and cannot be blamed for where we are now. We need to think of solutions and in that respect the CONTEXT users are lucky! They have always been told not to use this macro package for math because there are other standards and because publishers want LATEX (even if they just let the manuscripts be recoded). That means that we

don't really need to care much about the past. Those who use ConTeXt can benefit from the compatibility we have anyway but also move forward to more structured and consistent math. It is in this perspective that we will discuss some more details next so that eventually we can draw some conclusions. The end goal is to have an additional layer of grouping math symbols that permits consistent high quality rendering in a mixed input environment.

16.2 Molecules

Before we go into details about some characters, we spend some word on the rendering. The building blocks of a formula are atoms and internally the term nucleus is used for what we have without scripts. The simple sequence `1 + x` will result in a linked list of three atoms with three nuclei. In `x^2` the `x` is the nucleus. Atoms can have scripts: prescripts, postscripts and a prime. The majority of Unicode math characters become such atoms (nuclei and scripts) and they get a class property that determines their spacing, but that is not part of the Unicode specification. From the upcoming sections it will be clear that when we classify we don't get that much help from MathML or even the TeX community either.

In addition to these atoms the LuaMetaTeX engine (which builds upon TeX) has what we can call molecules. There are several types: fractions, accents, fences, radicals. This distinction is to some extent present in Unicode: plenty of fraction related slashes, all kind of accents, vertical delimiters that can be made from snippets and act as fences, and a radical symbol. In MathML we see similar constructs but there in practice quite often operators need to be interpreted in a way that can distinguish between atoms and molecules. That is partly a side effect of applications that generate MathML. And as usual with standards pushed upon the world without years of exploration the confusion became part of the norm and will stay.

In the TeX engine over and under delimiters are implemented on top of radicals (using the same noad, the wrapper node for yet unprocessed math) but they have different code paths. Basically we have vertically fenced material and just like fractions have left and right fences as part of the concept (for binominals) the radical has a sort of left fence too. You can also wonder why we need accent noads while we support other delimiters with radicals. This organization mostly relates to subtypes and classes (and likely some limitations of the past) that have related spacing properties, but we can think if a generic structure noad and meaningful subtypes. However, that is not what we get so let's be more precise:

Fractions: these stack two atoms (or molecules) and separate them by a visible or phantom rule, or in LuaMetaTeX by a delimiter. They can have a left and right fence which originates in them also suitable for binominals. You may wonder why we don't use regular fences here. One reason we can think of is that when you fence something, you have an open and close class at the edges while with a fenced fraction the whole still is fraction. In LuaMetaTeX we can tweak classes at the edges but in regular TeX there are fewer classes, so there constructs become ordinary or inner.

Accents: these put something on top of or below an atom (or molecule) and are driven by characters. The accent related commands take an integer (traditional) or three integers (extended) and it is this expected input that drives it. However, they are treated like delimiters. In traditional TEX a delimiter is defined by two characters: the direct unscaled one, and when not found a second one drives the lookup from wider variants and eventually an extensible character. Accents just have the second one, which probably relates to the fact that the text ones that would be the starting point make no sense. It is this '*looking*' for a single code point that makes that accents are not merged with the more general radical command space. Another reason is that accents deal a bit different with spacing and italic correction so even if we could merge, it would be more confusing in the end.

Fences: these come in pairs with optional middle ones. The reason for pairing is that they need to get the same size. That means that before we construct them the atom or molecule that they fence has to be analyzed. It also makes the result a construct of its own, although in LUAMETATEX we can unpack that result so that it can be broken across lines. In practice that was never an issue because in a running text unscaled fences are used (just atoms with open and close classes assigned) but as soon as one goes to multi-line displays formulas things become more hairy. The related commands expect delimiters (the two part character definitions) but in the meantime are also happy with a single one because in the end OPENTYPE math has all in one font.

Radicals: originally this only concerned roots but because they are basically wrappers we also use them for content that gets a delimiter above, below or both. In that sense the term radical can also be interpreted as '*extreme*', more than a carrot looking symbol. The related commands take one or more delimiters (or character) because we support left as well as right delimiters connected by a rule, so in the end radicals evolved into a construct with delimiters of all kind. So, the unique property of radicals is that the fences assume a cooperation between one or more glyphs and a rule. In CONTEXT we support actuarian hooks as radicals that are used for annuity expressions, otherwise the UNICODE symbols is useless and the MATHML construct complex.

So, where accents take numbers as delimiter specification, fences, fractions and radicals take specific math quantities or just letters. This makes that we will not merge these into one scanner and handler even if they all use the same (large) noad to store and carry around their properties. Also, it has some charm to keep the original TEX distinctions. After all, it's not like UNICODE, MATHML or OPENTYPE math fonts have brought some new insights: in the end they all draw from TEX and they way it's done there.

16.3 Symbols

There are plenty of symbols in UNICODE. When we try to get an idea how we ended up with that set we're surprised that not much seems to be known about it. There are references to ISO standards, usage by specific organizations (like those dealing with patents), there are references to lists of publishers. In personal communications with people involved it becomes clear that the criterion that some symbols really has to be

used somewhere doesn't apply to these math symbols. There are bizarre specimens that we cannot locate anywhere. They are often assigned the *'relation'* property which for TEX is a safe bet because binary and relations get similar spacing, but binary makes an exception when it sits at the front. The fact that relation spacing is used can even obscure the fact that some characters have zero width properties; the results just look somewhat bad and one can always blame the font or renderer and adding some thin spacing is accepted behavior. So one can make the argument that because TEX was the main renderer of math, a safe bet was better than a confusing and unproven-by-usage assignment to some category.

In TEX some symbols have multiple names, even when they have the same class. This indicates the wish for meaning at one end but shape at the other, and once a name has been assigned it sticks. It would be interesting to know how mathematicians see formulas: if one puts `\bar`s around a variable does one see *"bar x bar"* or *"the modulus of x"*, and how is translation to audio to be performed?

One important aspect of using any symbol in TEX, or basically any typesetting system that deals with math, is that the spacing depends on the meaning. Now, in the perspective of UNICODE meaning is somewhat diffuse. A Latin capital *'A'* related to *'a'* is not the same as a Greek capital *'A'* that relates to *'α'*. So, from the shape one cannot beforehand deduce what is meant, but when copying it the UNICODE will expose the meaning. This is not the case in math: although many symbols have one meaning only, there are also plenty that can mean different things and the (TEX) math community has not been able to make a strong case for providing different slots. Maybe the reason was that there already was a tradition of using commands that then relate a shape to a class that then results in appropriate spacing. Maybe it is also assumed that an article or book starts by explaining what a specific symbol means in that particular context. But that doesn't help much for copying. It also doesn't help with direct UNICODE input. The way out for this last problem is that in CONTEXT we will add additional properties to characters that then can communicate the class and thereby control the spacing. Although we initially did that at the LUA end we now use the lightweight dictionary feature of the engine: a property, group, slot model. The main reason is that we foresee that at some point we might have to add property based rendering to the engine, and this opens up that possibility. Ever since we started with LUATEX and MKIV we have used the character database (in LUA format) to store most properties so that we have all in one place.

For figuring out the properties we can look at how traditionally symbols got multiple commands associated, how MATHML looks at it, what UNICODE reveals and what we find in fonts. It is a bit of jungle out there so for sure we have to make decisions ourselves. We next turn to that exploration.

16.4 Slashes

The definition on the WIKIPEDIA page [1] of slashes is as follows:

> *"The slash is an oblique slanting line punctuation mark /. Once used to mark periods and commas, the slash is now used to represent exclusive or inclusive or, division and*

fractions, and as a date separator. It is called a solidus in UNICODE, is also known as an oblique stroke, and has several other historical or technical names including oblique and virgule."

The page then has a very detailed description on how slashes are used in text, mathematics, computing, currency, dates, numbering, linguistic transcriptions, line breaks, abbreviations, proofreading, fiction, libraries, addresses, poetry, music, sports, and text messages. It is a pretty good and detailed page which also gives a nice summary of usage in math.

In mathematics, we use the slash (a forward leaning bar) for fractions, division, and quotient of set. Examples of fractions are $^1/_2$ but also % sits in this category.

U+0002F	/	this is the official solidus
U+02044	⁄	the mathematical fraction slash
U+02215	∕	the mathematical division slash
U+02571	╱	a diagonal box drawing line
U+029F8	⧸	the mathematical big solidus
U+0FF0F	／	a full width solidus
U+1F67C	🙼	the very heavy solidus

The STIX fonts have the first five, the rest is not there, so we can safely assume that they are not used in math. That brings us to the question that, say that the other ones are used, how does the user access them? In the editor they often look pretty much the same. For TEXies the answer is easy: you use a command. But as we already mentioned, there we enter a real fuzzy area: these commands either describe a shape or they communicate a meaning, at least, in an ideal world. Sometimes wrapping in a macro helps, like `$\vfrac{1}{2}$`.

In the document that explains UNICODE math there is a section *"Fraction Slash and Other Diagonals"*. Even if we limit ourselves to the forward leaning slashes it looks like we need to include exotic symbols, as the empty set symbol with an left arrow on top: U+29B4 a circle with left pointing arrow on top, that doesn't show up in most math fonts but STIX has it ⦴. We quote:

"U+2044 / FRACTION SLASH is typically used to build up simple skewed fractions in running text. It applies to immediately adjacent sequences of decimal digits, that is, to spans of characters with the General Category property value Nd. For example, 1/2 should be displayed as ½. In ordinary plain text, any character other than a digit delimits the numerator or denominator. So 5 1/2 should be displayed as 5½ since a space follows the 5. In general mathematical use, a more versatile method for layout of fractions is needed (see, for example, Section 2.1 of [UnicodeMath]), however parsers of mathematical texts should be prepared to handle FRACTION SLASH when it is received from other sources. U+27CB MATHEMATICAL RISING DIAGONAL and U+27CD MATHEMATICAL FALLING DIAGONAL are mathematical symbols for specific uses, to be distinguished from the more widely used solidi and reverse solidi operators as well as from nonmathematical diagonals."

In TEX there is no parsing going on: we just get sequences of atoms and the inter atom spacing applies. Curly braced arguments are used to communicate units that needs to be treated a while. As side note: where for some scripts there are special characters that tell where something (state) starts and ends this is not available for math, which makes it impossible to mark a sequence of characters as being something math. The whole repertoire of pre-composed fractions and super- and subscripted UNICODE symbols are not to be used in math.

Most documents that somehow relate to or (partially) originate in TEX can be rather fuzzy, so we can read here:

> *"U+27CB corresponds to the LATEX entity \diagup and U+27CD to \diagdown. Their glyphs are invariably drawn with 45° and 135° slopes, respectively, instead of the more upright slants typical for the solidi operators. The diagonals are also to be distinguished from the two box drawing characters U+2571 and U+2572. While in some fonts those characters may be drawn with 45° and 135° slopes, respectively, they are not intended to be used as mathematical symbols. One usage recorded for U+27CB and U+27CD is in the notation for spaces of double cosets."*

So, it is the angles that math users should translate into meaning which I guess is natural for them. From the above we cannot deduce if we should take them into account in a macro package.

The MATHML specification [3] keeps it abstract and talks about division without mentioning the rendering. In content MATHML we have:

```
divide = element divide { CommonAtt, DefEncAtt, empty}
```

and the suggested rendering (from an example) is a slash.

In the chapter *"Characters, Entities and Fonts"* there is mentioning of:

> *"There is one more case where combining characters turn up naturally in mathematical markup. Some relations have associated negations, such as U+226F [NOT GREATER-THAN] for the negation of U+003E [GREATER-THAN SIGN]. The glyph for U+226F [NOT GREATER-THAN] is usually just that for U+003E [GREATER-THAN SIGN] with a slash through it. Thus it could also be expressed by U+003E-U+0338 making use of the combining slash U+0338 [COMBINING LONG SOLIDUS OVERLAY]. That is true of 25 other characters in common enough mathematical use to merit their own UNICODE code points. In the other direction there are 31 character entity names listed in [Entities] which are to be expressed using U+0338 [COMBINING LONG SOLIDUS OVERLAY]."*

A curious note is this:

> *"For special purposes, one may need a symbol which does not have a UNICODE representation. In these cases one may use the mglyph element for direct access to a glyph as an image, or (in some systems) from a font that uses a non-UNICODE encoding. All*

MATHML token elements accept characters in their content and also accept an `mglyph` *there. Beware, however, that use of* `mglyph` *to access a font is deprecated and the mechanism may not work in all systems. The* `mglyph` *element should always supply a useful alternative representation in its alt attribute."*

At some point we experimented with very precise positioned HTML from TEX (read: CONTEXT) and that worked very well: the rendering was exactly the same as PDF but then suddenly it was no longer possible to access glyphs from fonts. The assumption had become that one should feed text into the font rendering machinery and use OPENTYPE features to access specific shapes, which of course is a fragile approach (the libraries and logic keep evolving, and the most robust access is simply by index, or by glyph name if present, assuming that one uses the font that was meant to be used). So, how the MATHML glyph element is supposed to work out well is not clear. Anyway, as we want nicely typeset math we don't care that much if features present in LUAMETATEX and CONTEXT are unique and cannot be reproduced otherwise.

In `mathclass.txt` [4] which is "not *formally part of the UNICODE Character Database at this time*" we see a classification:

`U+0002F`	binary
`U+02044`	binary
`U+02215`	binary
`U+02571`	not mentioned
`U+029F8`	n-ary or large operator, often takes limits
`U+0FF0F`	not mentioned
`U+1F67C`	not mentioned

So, in the end we can focus on the four that are mentioned, and we will do that with the above in mind as well as what is common in the TEX world. We will look at usage, classification (groups) and classes.

Unfortunately this sort of mess also results in a mess in fonts. For instance when we checked out the difference between `U+002F` and `U+2044` we found that in the fonts produced by the TEXGyre project both have proper dimensions (and look the same), so they can be used stand alone, but also as delimiters. In Cambria the dimensions are okay but only `U+2044` has extensible characters. In CONTEXT we have defined `\slash` to use that slot but when you test Lucida and STIX2 the results are disappointing: In Lucida the width of `U+2044` makes it unusable (it looks bad anyway), and in STIX2 it is a bit wider so in the end it even becomes fuzzy what to recommend as fix: quarter width, half width or full width. Defining `\slash` as any of them gives at some point an issue so in the end we just patch the font in the goodie file: we make them the same and make sure they have extensible characters. After all, chances are slim that this will ever be fixed. In that respect a newer engine doesn't change the problem: we need to handle it in the macro package, but at least that can be done a bit more natural.[12]

[12] In principle, we can support the goodies in the generic font handler, but we think it makes no sense because it also relates to the way math is handled in general and supporting a wide range of different applications can only cripple the code, let along that agreeing on matters can be hard.

16.5 Bars

Again we start with the WIKIPEDIA page, this time the one dedicated to bars [5]. The page starts with mathematics so that suggests that the (initial) author is familiar with usage in that field: if we cut and paste the itemized list we even get TEX math (sort of). Examples of usage are: absolute value, cardinality, conditional probability, determinant, distance, divisibility, function evaluation, length, norm, order, restriction, set-builder notation, the Sheffer stroke in logic, subtraction, but also *"A vertical bar can be used to separate variables from fixed parameters in a function, or in the notation for elliptic integrals"*.

Among the objectives of our exploration are grouping symbols in sets that represent related meanings and usage. Within these groups we can fine tune with classes but that is more geared at rendering. Although currently users enter specific usage of symbols with the same shape (or even UNICODE) with commands we can imagine them entering the *'real'* characters and in that case we need some automatic class assignment based on a group (or set of groups). The WIKIPEDIA page mentions that in physics *"The vertical bar is used in bra–ket notation in quantum physics"*. It then goes on about usage in computing, phonetics and literature. This ordering is different from the slashes, but okay.

The page then makes a distinction between solid and broken bars and there is some interesting history behind that, which relates to typewriters, terminals and printers in the perspective of distinction and indeed we noticed that on our keyboard the broken bar is still used, even if the rendering is solid. The page ends with the UNICODE bars and entities. We mention most:

U+007C	|	a single vertical line
U+00A6	¦	a single broken line
U+2016	‖	a double vertical line (norms)
U+2223	∣	divides
U+2225	∥	parallel lines
U+2502	│	a vertical box drawing line
U+FF5C	?	a fullwidth vertical line

Given the mentioned wide range of usage it will be clear bars that can be confusing and are pretty overloaded. We're not aware of broken bars being used in math, so we ignore these.

The UNICODE math draft talks of *'vertical lines'* and distinguishes two series, delimiters:

U+007C	|	single vertical lines
U+2016	‖	double vertical lines
U+2980	⦀	triple vertical lines

and operators:

U+2223	∣	divides (single line)
U+2225	∥	parallel (double lines)

U+2AF4 ||| binary relation (tripple lines)
U+2AFC ||| s large triplle operator

Watch the triples: these are not (yet) in the WIKIPEDIA summary. Rightfully there is a remark that the official UNICODE descriptions use BAR and LINE but TEXies can't complain about that, can they? After all, they also use these terms mixed.

The delimiters sit at the edges but sometimes also in the middle. The operators are between other elements and the document states that they also should grow. And is it mentioned that spacing depends on usage. The large triple is an n-ary operator but as usual with math symbols the user (reader) has to guess what that actually means.

It is actually unfortunate that the fences have no left, middle and right variant. Even if these render the same it would make life easier and consistency with other fences is also worth something. One wonders how it would have looked if accessibility demands had kicked in earlier. The UNICODE mathclass.txt [4] provides:

U+007C fence (unpaired delimiter)
U+2016 fence (unpaired delimiter)
U+2980 fence (unpaired delimiter)

We assume that the unpaired qualification is actually an indication that usage as what in TEX is called '*middle*' is okay. The operators are classified as:

U+2223 relation
U+2225 relation
U+2AF4 binary
U+2AFC large n-ary

The main problem with bars in TEX is that there is no distinction between a left and right bar which makes it impossible to use them directly as fences. On can consider this to be an omission to UNICODE math because shape rules over meaning. So anyway, this is something that a macro package has to deal with. If needed these can get a class on their own in which case we can define atom spacing rules that deal with them ending up left or right. In UNICODE there are signals that deal with bidirectional text, so we see no reason why there shouldn't be similar provisions for math.

16.6 Hyphens and Dashes

This section applies to text and math as both are riddled with horizontal lines: easy to scratch in wood, chisel in stone or draw on paper symbols. We limit ourselves to the straight ones, but similar observations can be made for curved ones.

WIKIPEDIA distinguishes hyphens, minus, and dashes so there are multiple pages dedicated to this. The page about minus mentions that there are three usages (somewhat rephrased):

- It is used as subtraction operator and therefore a binary operator that indicates the operation of subtraction.
- It can be function whose value for any real or complex argument is the additive inverse of that argument.
- It can serve as a prefix of a numeric constant. When it is placed immediately before an unsigned numeral, the combination names a negative number, the additive inverse of the positive number that the numeral would otherwise name.

The functional variant is how content MATHML sees it: you apply a minus operator to something, singular of multiple. We were surprised to see that there is a distinctive rendering suggested, something we have argued for at several occasions (mostly TEX meetings):

> *"In many contexts, it does not matter whether the second or the third of these usages is intended: −5 is the same number. When it is important to distinguish them, a raised minus sign ⁻ is sometimes used for negative constants, as in elementary education, the programming language apl, and some early graphing calculators."*

Unfortunately that distinction was not recognized by the TEX community at large which (we guess) is why we don't see it in UNICODE, which on the other hand has plenty dashes as we will see soon.

The page mentions usage in indicating blood types and music, which is a nice detail. It also mentions usage in computing, including regular expressions and in physics and chemistry indicating charge. It lists these codes for minus symbols:

U+002D	hyphen minus
U+2212	minus
U+FE63	small hyphen minus
U+FF0D	full width hyphen minus

The page also mentions the commercial minus ⁒ (see also [7]) and division sign ÷ (see also [8]) and we think these should be supported in math mode simply because they can be part of (even simple text style) formulas.

The fact that we use the hyphen as minus and expect it to render as a wider dash like shape is something that related to math mode in TEX speak. In text mode we expect it to be seen as hyphenation related indicator. We won't go into details about automated hyphenation and explicit hyphens in text mode but here are the hyphens as mentioned on the hyphen specific WIKIPEDIA page:

U+002D	hyphen minus
U+00AD	soft hyphen
U+2010	hyphen
U+2011	non breaking hyphen

You might wonder why we mention text variants here and one reason is that we actually might need to provide a catch for the last two: maybe when a user copies these from a

document (when rendered at all) we need to treat them as the simple hyphen minus and just remap them to the math minus when in math mode. Below, we will discuss dashes, and although these are also meant for text, a reason for exploring these can be found in the fact that TEX users like to decorate the content in unexpected ways and lines (or rules) fit into that. The WIKIPEDIA pages go into some details about the hyphens being used in compounds and there can be some confusion about whether to use endashes or hyphens for that. We're pretty sure that typesetting wars have been fought over that. Usage as pre- and suffixes definitely is worth noting (and we use them as such in this sentence).

We leave out all the other usages and see what there is to tell about related symbols. The WIKIPEDIA page about dashes is an extensive one. It starts out with the distinction between figure dash (U+2012: ‒), endash (U+2013: –), emdash (U+2014: —) and horizontal bar (U+2015: ―). Of these a TEXie will for sure recognize the endash and emdash. The hyphen is not a dash but if you look at TEX input that double or triple hyphens get ligatured into en- and emdashes! The only certainty one has is that the endash is often half the width of an emdash. Also, the width of the emdash is often the same as the font size.

One reason why a language subsystem of a TEX macro package is complex is that it has to deal with cultural aspects and the usage as well as spacing around all these dashes can differ. When trying to support that a macro writer soon finds out that one user of language X can tell you the rules are done this way, and a while later you get a mail from another user who claims that in language X the rules are done that way. Word processing and dominance of English probably adds to the confusion. The same is true for quotes, but math doesn't need these, so we skip them. Now wait, you will say: does math use these dashes? Users probably will mix them in but more important is that the width of these dashes also has associated skips: `\enspace` and `\emspace` or `\quad` and these one definitely see users mix into math.

The figure dash has the same width as digits which makes them useful in tables. In the fonts that come with TEX it is the reverse: the digits have the same width and that width matches the endash. There is no habit of using the figuredash, but we might need to change that. After all, we now have the fonts! We do need to deal with the figure dash because users might mix math and text in tables, and although you can find plenty of badly typeset by TEX tables, this is no excuse for using a mix of minus and figure dash in inconsistent ways.

The WIKIPEDIA page mentions the usage of the endash: as connector, as compound hyphen, and as sentence interrupter. Now the one that needs some attention is the second one. In Dutch, we can combine words in many ways and for educational purposes adding a compound dash makes sense. However, because the weight of the hyphen and endash in TEX fonts is rather incompatible, in CONTEXT we use(d) fakes: two overlapping hyphens. Another complication is that one has to wrap that in a discretionary node in order to make the hyphenator happy, but that is now delegated to the engine that can be configured to see certain characters as valid hyphenation points. Although

we support discretionaries in math this doesn't relate to dashes but to pluses and minuses and such. The engine supports explicit discretionaries but can also automatically repeat symbols that are set up as repeatable across lines. We're not sure if users actually use en- and emdashes in math mode, but one can occasionally run into examples (on the web) where special effects are achieved in curious ways.[13]

It is worth pointing out that WIKIPEDIA discusses *"Ranges of values"* and this is something we need to investigate in the perspective of math! Strictly spoken that is a text thing, but . . . Among the many observed and suggested patterns we note that among TEXies using the endash as itemize symbols is also popular.

Usage of the emdash is related to the use of parenthesis or colons, so it is more a kind of punctuation. It can also be used as an interrupt and again it is a candidate for an itemize symbol. There is of course a TEX thing there: lack of text symbols made for a rather mixed usage of math and text symbols in itemizations. For instance a dotted one uses the well visible math dot instead of the often hardly visible text dot that simply was not present in TEX fonts, so our eyes got accustomed to the bolder ones. It is one of the reasons why a TEX macro package load a math font even when no math is used. Over the years in TEX math and text symbols have been mixed in various ways, also a side effect if the limited amount of characters in text fonts and the abundance of them in math mode, even if most are only accessible by name. We need to deal with that historic mix.

The page rightfully mentions that TEX has no horizontal bar, also known as *'quotation dash'*, used for dialogues in some languages. We should make a note then that it might be good to see if we have to reconfigure the sub-sentence presets to match that expectation. The proposed hack MPS: where? for a missing symbol is somewhat curious:

```
x \hbox{---}\kern-.5em--- x
```

Why not `\hbox{---\kern-.5em---}` or just `---\kern-.5em---` to get the same effect? This also assumes that the font collapses these three hyphens into a dash, then it backtracks the symbol width and does a second one.[14] Anyway, where figure dashes are related to minuses we can probably ignore this super minus resembling horizontal bar.[15]

The WIKIPEDIA page ends with a summary of all kind of dashes, including underscores, script specific symbols, accents (like macron), modifiers and curly ones. Here we only

[13] The math stream doesn't go through the font handler although embedded `\hbox`es get that treatment. This means that two hyphens in a row are just two atoms and not get collapsed to an endash.

[14] Here is some food for thought: for this kind of usage one can argue that such a dash should have some stretch. In LUAMETATEX and therefore CONTEXT we can do this: `\uleaders \hbox to 1.5em {---\hskip 0pt minus .5em---} \hskip.125em minus .125em \relax` and get: x —— xx —— xx xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— xx —— x. Boxed material can be stretched and be taken into account when creating paragraphs. It is no big deal to wrap that in a macro, say `\figuredashed`.

[15] We can actually issue a warning when it is used in math mode.

mention the ones that can end up in some source when one cuts and pastes. Doing that can result in missing characters (because not all fonts provides them) or a change in meaning (for as far as the symbols relates to an intention). We show some that fit into this discussion and also mention the UNICODE description:

U+002D	HYPHEN-MINUS	the usual hyphen but also used as minus
U+005F	LOW LINE	aka underscore
U+00AD	SOFT HYPHEN	valid hyphenation point (invisible)
U+2010	HYPHEN	the real hyphen but more work on a keyboard
U+2011	NON-BREAKING HYPHEN	a hard hyphen, disables following hyphenation
U+2012	FIGURE DASH	see discussion above
U+2013	EN DASH	see discussion above
U+2014	EM DASH	see discussion above
U+2015	HORIZONTAL BAR	see discussion above
U+2043	HYPHEN BULLET	used in itemized lists
U+207B	SUPERSCRIPT MINUS	combined with pre-superscripted characters
U+208B	SUBSCRIPT MINUS	combined with pre-subscripted characters
U+2212	MINUS SIGN	the math minus (rendering of hyphen)
U+23AF	HORIZONTAL LINE EXTENSION	build long connected horizontal lines
U+23E4	STRAIGHTNESS	represents line straightness in technical context
U+2500	BOX DRAWINGS LIGHT HORIZONTAL	part of the box-drawing repertoire
U+2796	HEAVY MINUS SIGN	a visual variant with no meaning
U+2E3A	TWO-EM DASH	a visual variant with no meaning
U+2E3B	THREE-EM DASH	a visual variant with no meaning
U+FE58	SMALL EM DASH	a visual variant with no meaning
U+FE63	SMALL HYPHEN-MINUS	a visual variant with no meaning
U+FF0D	FULLWIDTH HYPHEN-MINUS	a visual variant with no meaning

The UNICODE math draft only mentions the hyphen:[16]

> *"Minus sign. U+2212 [or] - [known as] MINUS SIGN is the preferred representation of the unary and binary minus sign rather than the ASCII-derived U+002D [or] - [known as] HYPHEN-MINUS, because minus sign is unambiguous and because it is rendered with a more desirable length, usually longer than a hyphen."*

and elsewhere we can read:

> *"The ASCII hyphen minus U+002D [or] - is a weakly mathematical character that may be used for the subtraction operator, but U+2212 [or] - [known as] MINUS SIGN is preferred for this purpose and looks better."*

[16] When I copy this snippet into the document source there are START OF TEXT symbols at the places where a hyphenation occurs, which is probably a side effect of a bad TOUNICODE entry in the PDF file, but it is kind of interesting in this perspective as definitely a hyphen is rendered.

We are not aware of the concept of weak mathematical characters, so we will not take that property too serious when we try to improve the rendering.

This is basically it. There is no mentioning of classes (after all, traditional TEX has no unary class) so it is assumed that the renderer does the right thing: interpreting the sequence of characters and apply spacing accordingly. There are users who like to see a unary minus being rendered differently, just as the minus that a student is supposed to key in a calculator and while the WIKIPEDIA page mentions this explicitly, it is ignored here. Yes, having two distinctive slots for this would have been great. Maybe it is not seen as relevant enough by the community that would benefit most, but who knows what had happened it the WIKIPEDIA page had been there before!

The minus is mentioned in the somewhat curious section about how shapes should be positioned relative to the baseline, where the position of the minus relates to what in TEX speak is the math axis. There is also some mentioning of non-mathematical use, like:

> *"The concept of mathematical use is deliberately kept broad; therefore the Math property is also given to characters that are used as operators, but are not part of standard mathematical notation, such as U+2052 COMMERCIAL MINUS."*

There should be no confusion with the SET MINUS which renders as a backslash, a (NEG\-ATED) MINUS TILDE or (NEG\-ATED) SIMILAR MINUS SIMILAR that look more like relations. MPS: overfull hbox, and do you intend to hyphenate?

The MATHML document recognizes the minus as being unary or binary. In content MATHML it is easy: when applied to a single atom it is a unary. In presentation MATHML minus is an operator that sits at the front of a row (unary) or in the middle (binary). Keep in mind that we are limited to mn for numbers, mi for alphabetic symbols and mo for operators, not to be confused with TEX's math operators, because in MATHML relations are also operators. One can wonder about a minus in mn elements.

So to summarize: we definitely need to make sure that (whatever renders as) hyphens is dealt with in math as minus. We can wonder what to do with (especially) en- and emdashes and the other horizontal lines that actually might show up as (what we call) middle delimiters in mathematical constructs: if it's there, TEXies will use it! The lack of specific symbols for unary minus has to be compensated at the macro package level.

16.7 Pieces

In UNICODE one can find all kind of constructors, for instance characters that find their origin in those character sets that had lines and corners for drawing on a terminal. It is therefore no surprise that there are also some constructors that relate to math. An example demonstrates this:

```
\def\makeweird#1#2#3#4%
  {\vcenter\bgroup
```

```
    \offinterlineskip
    \hbox{$\scriptscriptstyle\char"#1$}\par
    \hbox{$\scriptscriptstyle\char"#2$}\par
    \hbox{$\scriptscriptstyle\char"#3$}\par
    \hbox{$\scriptscriptstyle\char"#4$}%
  \egroup}

\def\lwA{\mathopen {\makeweird{23A7}{23A8}{23A8}{23A9}}}
\def\rwA{\mathclose{\makeweird{23AB}{23AC}{23AC}{23AD}}}
\def\lwB{\mathopen {\makeweird{23A7}{23AC}{23AC}{23A9}}}
\def\rwB{\mathclose{\makeweird{23AB}{23A8}{23A8}{23AD}}}
\def\lwC{\mathopen {\makeweird{23A7}{23AC}{23A8}{23A9}}}
\def\rwC{\mathclose{\makeweird{23AB}{23A8}{23AC}{23AD}}}

$\lwA x + 4 + \lwB x^2 + 4^2 + \lwC x^3 + 4^3 \rwC \rwB \rwA$
```

This renders as:

$$\lwA x + 4 + \lwB x^2 + 4^2 + \lwC x^3 + 4^3 \rwC \rwB \rwA$$

So, we have official UNICODE characters for constructing large fences. In the UNICODE math documents there is some mentioning of this and interesting is that there are suggested compositions expressed in 2, 3, 5 etc. stacked *'lines'* which makes one wonder how math is perceived (or supposed to be rendered). But what is really weird is that there are plenty of arrows but no snippets defined that can be used to create extended ones. Why vertical snippets and no horizontal ones? This is clearly an omission and the TEX community did take care of this need. So, for horizontal arrows and alike one expects the font to handle it and for fences not?

It is not only fences that have snippets, we also find them for integrals. But for vertical arrows they are lacking: that is completely up to the font. Now, for us that is fine, but again, for consistency they could have been there. It would make it possible to filter bits and pieces from fonts using official slots instead of private ones. So, to some extent we can best assume there is nothing like that and ignore whatever pieces are in UNICODE anyway (like the braces in the example). One can even argue that because of this inconsistency a font designed can as well only use private slots and not provide snippets at all.

So, how do we get out of this situation? Because no one cared getting it in UNICODE, we can do as we like. Of course, we can define arrow fillers as has always been done in TEX, but because in LUAMETATEX we have a bit more in our toolkit, and because we want to support stretch fractions (where the rule is replaced by a horizontal delimiter) it was decided to define a tweak that deals with this: when the basic arrows have no horizontal parts defined, we just assemble them. For those arrows that have a hook or so at the

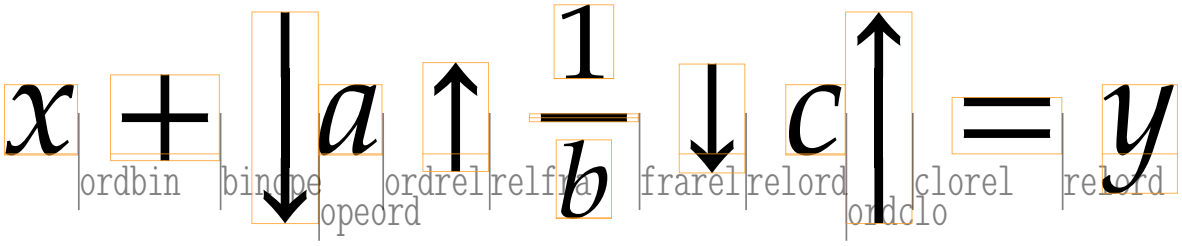
other end, we use the space as extender.[17] If we ever end up with proper snippets un UNICODE then we also need adapted fonts, and then we can get rid of these hacks. That said: because all decent math fonts do have the three pairs or fences (brace, parenthesis, bracket) the vertical snippets are rather useless, unless one wants to construct assembled weird ones. This would be different for horizontal assemblies, because there is more variety in them.

The official name for all related to characters that can stretch is *'delimiter'*. In traditional TEX one can define a command that becomes a character. In that case a family, class and slot is assigned. You can also directly access a character in which case one will assign these properties otherwise (no command is defined). The same is true for these delimiters. However, in traditional TEX the larger character usually comes from a so called extension font and uses family 3). In OPENTYPE fonts we have all in one font so there the large family, class, and slot are not used.

An interesting side effect of the updated math machinery in LUAMETATEX is that we no longer really need delimiter specifications when we use OPENTYPE fonts. This is because in practice the only two classes that really matter are the open and close ones. There are basically two kinds of delimiters: fences and singulars. Fences need open and close and only bars have a dual character. So, when we don't define it as delimiter, the engine can still use that character and take its assigned class when used stand-alone, while in the case of fences these themselves are of class open and close. And, for instance a left brace can get class open because when used stand alone it is an unscaled left fence. In the rare case that one really need a different class we are using commands: some characters can be binary, ordinary or whatever so then commands relate a name to a class-character combination. Actually, in CONTEXT we will switch to using dictionaries and field specific rendering instead, but that is a different story. We can illustrate the arrows with an example:

```
$ x +
    \left\downarrow a \uparrow \frac{1}{b} \downarrow c \right\uparrow
= y $
```

The stand alone arrows are defines with class relation but when used as fences their spacing is driven by the fences themselves.



This means that in CONTEXT LMTX we no longer have delimiter code definitions. Of course the engine has to be able to use math characters of any kind (by commands,

[17] Actually we no longer do that because the engine will center the arrow anyway when it's too short.

direct or as UTF) as delimiters, but that was not that hard to provide. It also simplifies the code we use for fencing as it can be less selective.

Another interesting side effect of once again looking into these stretched characters is that the fraction mechanism that already was extended with skewed fractions, now supports any stretchable character as alternative for a fraction rule.

```
$
    p \leftarrowtext {a + b + c + d}{x + y} q
    \quad
    p \frac {a + b + c + d}{x + y} q
$
```

Watch the difference in spacing: here the class of the used delimiter determines the spacing around the (pseudo) fraction:

Again this simplifies some code because normally one ends up with stacking stuff using leaders in between.

16.8 Accents

When we talk about accents, we refer to tiny symbols that anchor themselves onto base characters. We limit ourselves to the ones common in Latin scripts because they are the ones used in math. Accents in UNICODE are somewhat special. In the past, when encoding vectors were limited, accents were entered as part of an input sequence and then anchored by the renderer. Nowadays often pre-composed characters are used. A very cheap way of anchoring is to have accents that just overlay, and in practice centering an accent over a base character works sort of okay. As an example of an accent we will use the hat:

U+005E	x^x m^m	\Hat	$x^x + m^m$
U+02C6	x̂x m̂m	\hat	$x\hat{}x + m\hat{}m$
U+0302	x̂x m̂m	\widehat	$x\hat{}x + m\hat{}m$

Normally the font handler will take care of anchoring U+0302, but it can only be done properly when there are anchors defined for what are called *'marks'*: the official feature description is mark-to-base (or simply mark). The last column in the above table shows math and as we input a raw character we don't get proper anchoring: the zero width makes it overlap.

Now wait, you will say, but why does it actually overlap? The reason is that zero width is not actually zero width here! The glyph has a bounding box that goes into the negative

horizontal direction and therefore, when such a shape gets injected into the output, the rendering in the viewer will move the left edge to the left. But because the TEX engine only handles positive widths and because the width is explicitly part of a character specification anyway[18] we don't progress (advance) which is why the overlapping sort of works for the *x* but less so for the *m*: in math mode we need to use these `\hat` and `\widehat` commands.

The hat and widehat assignments were those of August 2022. In plain TEX we see these definitions:

```
\def\hat     {\mathaccent"705E }
\def\widehat{\mathaccent"0362 }
```

The `\mathaccent` primitive takes an integer that encodes the class, family, and slot in the 8 bit font encoding. Here we see that the hat comes from family 0, the upright math font. The widehat comes from extensible family 3. These two are independently defined. When you want a hat that spans the nucleus, you need to use the widehat. In the math engine spanning actually means that we have a delimiter and normally that means: start with a basic shape, when that is too narrow, go to the extensible font and follow the chain with increasing sizes and when you run out of those apply an extensible recipe. The sequence and extensible are both optional and the important part is that we first look at what is called the small character and then to the large one(s).

However, the `\mathaccent` primitives doesn't take a delimiter! It directly starts following a chain if the given character has it (and then the character itself is of course the first in that chain). And this is where the problems start when we move to OPENTYPE and UNICODE math.

U+005E	Hat	some useless, often ugly large glyph
U+02C6	hat	it has width but no extensibles
U+0302	widehat	it has zero width and extensibles

Now, if we define `\hat` as `U+02C6` we don't get the extensibles, and it basically is what was always done in TEX macro packages following the plain suggestions. If we define `\widehat` we start out with a glyph that has likely zero width[19] And, because OPENTYPE starts with the base glyph and *then* uses a set of variants of eventually a recipe of parts, we suddenly have a different situation with `\mathaccent` than we normally have, where these are decoupled. Therefore, the definition of `\hat` and `\widehat` determines what an OPENTYPE math engine will do, just as in regular TEX, but we might need them to be defined differently.

A solution would be to let `\mathaccent` (or `\Umathaccent`) directly go to the variants, but that is sort of weird. Because a zero width glyph doesn't match the criteria to span a nucleus it is likely to be skipped anyway, although there can be a case where the next in

[18] The height and depth are not: these we derive from the bounding box.

[19] Over the many years that LUATEX evolved this was not guaranteed, for instance when wide (UNICODE) fonts were constructed from traditional eight bit (TEX encoded) fonts.

size overruns the width of the nucleus in which case the zero width one is used which itself is not that nice. We could actually derive the width from the boundingbox, but that would be a bit abnormal, and it makes no sense to burden the font machinery with that exception. Another approach we can follow is to just copy the extensibles from U+0302 to 02C6 and use that one for \hat as well as \widehat and then make \widehat an alias to \hat. After, all, the main reason why we have two commands comes from the fact that \mathaccent doesn't take a delimiter but single character reference (encoded in an integer).

Here is the whole list of accents:

\grave	U+0060	\widegrave	U+0300
\ddot	U+00A8	\wideddot	U+0308
\bar	U+00AF	\widebar	U+0304
\acute	U+00B4	\wideacute	U+0301
\hat	U+02C6	\widehat	U+0302
\check	U+02C7	\widecheck	U+030C
\breve	U+02D8	\widebreve	U+0306
\dot	U+02D9	\widedot	U+0307
\ring	U+02DA	\widering	U+030A
\tilde	U+02DC	\widetilde	U+0303
\dddot	U+20DB	\widedddot	U+20DB

The only accent that is an exception is the last one but is it really used? It anyway makes no real sense to assume that users will ever directly input the UTF characters conforming the last column, so we can just go for the first one and use the extensibles from the second and see where we end up. Neither MATHML nor TEX related specifications seem to cover this well, so we can just do what suits us best.

Because all has to fit into the CONTEXT user interface and because we also want to be backward compatible (command wise), we end up with something:

```
\showglyphs
\im {\widehat{a} + \widehat         {aa}} =
\im {\hat    {a} + \hat             {aa}} =
\im {\hat    {a} + \hat[stretch=yes]{aa}} =
\setupmathaccent[top][stretch=yes]
\im {\hat    {a} + \hat             {aa}}
```

that gives us:

$$\hat{a} + \widehat{aa} = \hat{a} + \hat{a}a = \hat{a} + \widehat{aa} = \hat{a} + \widehat{aa}$$

Now, one problem, is of course that users can enter these modifiers as UTF sequence in the input, just like they do with delimiters. Therefore we do support the following feature (which is under class control so disabled by default):

```
\Umathcode    "02C6   \mathaccentcode 0 "02C6
```

```
\edef         \HiHatA {\Uchar"02C6}
\Umathchardef \HiHatB \mathaccentcode 0 "02C6

$ \Uchar"02C6{x} + \HiHatA{xx} + \HiHatB{xx} = \widehat {xxxx} $
```

You get this:

$\hat{x}\widehat{xx}+\widehat{xx}=\widehat{xxxx}$

The only cheat here is that normally accents come after the accentee, but we can live with that. After all, it's all about convenience.

There is another aspect of accents that we need to mention here. The hat, tilde, and check are often used over not only single letters but also small expressions. So how come that fonts have only very few variants defined? We can imagine that in eight bit fonts the number of available slots plays a role but in OpenType fonts that is not the case. It therefore can be considered an oversight that usage of these wide accents has not be communicated well to the font designers.

$$\hat{a}+\widehat{a+b}+\widehat{a+b+c}+\widehat{a+b+c+d}+\widehat{a+b+c+d+e}+\widehat{a+b+c+d+e+f}$$
$$\tilde{a}+\widetilde{a+b}+\widetilde{a+b+c}+\widetilde{a+b+c+d}+\widetilde{a+b+c+d+e}+\widetilde{a+b+c+d+e+f}$$
$$\check{a}+\widecheck{a+b}+\widecheck{a+b+c}+\widecheck{a+b+c+d}+\widecheck{a+b+c+d+e}+\widecheck{a+b+c+d+e+f}$$

The previous lines demonstrate that we can actually cheat a little for these three top accents: we can just scale the last variant horizontally. It was a few lines patch to LuaMetaTeX to make this automatic and triggered by setting the `extensible` field in a character table to `true` instead of a recipe. The ingredients to get this working were already there, and it works out quite well. The only complication was that the `flac` feature (that provides flat accents for cases where the nucleus is rather high) could interfere, but that was trivial to deal with in the code that does the goodies.[20]

When it comes to these delimiters that have no real solution in the font, we can consider delegating coming up with a glyph to the macro package at the time it is needed, and we can actually do that. However, this is mostly interesting for educational usage, where the amount of delimiters is predictable and limited. About a decade ago some mechanism was added to the MkIV math machinery that support plugins so that we could use MetaFun to generate (most noticeably) square root symbols the way we liked.[21] The main drawback is that mixing this in means matching to a font, and that is not always trivial. But it is this kind of trickery that makes working with TeX fun. That said: what we are discussing here is more fundamental in the sense that we try to come up with generic engine solutions that just rely on the fonts. That way complex math with all reasonable symbols is also served.[22]

[20] When we were testing fonts this got us by surprise when we tested Cambria that has these flat overloads for the tilde and check. Because supports this automatic (hidden from the user) one doesn't look into that direction when testing something.

[21] This was a fun project of Alan and Hans.

[22] These MetaFun plugins are still possible, but we need to adapt some to LMTX which will happen as we go.

Interestingly there are some arrows that act like accents. There are over- and under ones as well as combining (often zero width) accents. Fonts are not always consistent in how these extends (the wide ones). Often the combining accents are smaller and closer to the running text. Traditionally in TEX fonts there are no extensible arrows: they are constructed from arrow heads, minus and equal signs with some negative spacing in between. One can therefore wonder is the smaller combining ones are appreciated by those who want stable math. It definitely means that we have to make choices. Even more interesting is that while UNICODE has some means to construct braces from predictable UNICODE slots. there is no way to do the same with arrows and (indeed) there are fonts out there with shaped arrows that demand different middle and end pieces. In fact, the same is true for rules that are not simple rectangles and radical extensions that are not flat rules either. In all these cases the usage patterns of accents and similar constructs has not really been fed back into the way UNICODE and OPENTYPE fonts support math.[23]

16.9 Bullets

In TEX usage bullets are a it special. Because fonts had a limited number of slots available, bullets in for instance itemized lists traditionally were taken from a math font. The bullet in Computer Modern has a comfortable size and is quite useful for that. Bullets in text fonts often were (are) relatively small so even when they were available they were not really used. The official UNICODE slot for bullet is U+2022 and in this font it shows up as '•'. The WIKIPEDIA page on bullets (typography) mentions:

> *"A variant, the bullet operator (U+2219 ∙ BULLET OPERATOR) is used as a math symbol, akin to the dot operator. Specifically, in logic, x ∙ y means logical conjunction. It is the same as saying "x and y""*

The page also mentions that *"glyphs such as • and ∘"* have *"reversed variants ◘ and ◙"* although we haven't see the reverse once in TEX documents (yet), like these (we use STIX2 to show them):

U+2022	•	BULLET
U+2023	?	TRIANGULAR BULLET
U+2043	⁃	HYPHEN BULLET
U+204C	⁌	LACK LEFTWARDS BULLET
U+204D	⁍	LACK RIGHTWARDS BULLET
U+2219	∙	BULLET OPERATOR (math)
U+25CB	○	WHITE CIRCLE
U+25CF	●	BLACK CIRCLE
U+25D8	◘	INVERSE BULLET
U+25E6	∘	WHITE BULLET

[23] One can argue that this is not what UNICODE is for but if so, then some other bits and pieces also make little sense.

U+29BE ⦾ CIRCLED WHITE BULLET
U+29BF ⦿ CIRCLED BULLET

The reverse ones are not really reverse in STIX2 as they have bigger circles. There are a few more bullets mentioned but probably only because they have the word bullet in their description and they don't really look like bullets. Given the already discussed lack of granularity in some math symbols with multiple usage it is somewhat surprising that we have a math bullet. The weird looking left- and rightward bullets are kind of hard to distinguish. Let's hope that mathematicians don't discover these!

This brings us to the more general way of looking at these bullets because among the popular math symbols used in text are also the triangles and (TEX) math fonts came with. When we have a few commands for circular shapes like `$\bullet\bigcirc\circ$` giving $\bullet\ \bigcirc\ \circ$ we have plenty of (black) triangles.

For instance, we have `\triangledown` and `\bigtriangledown` and these have corresponding UNICODE slots `U+25BD` and `U+25BF` but when you try these in for instance STIX2, Pagella and Cambria you got: $\triangledown + $, $\triangledown + ?$ and $? + ?$, where the question mark indicates a missing character.

It is for that reason that `\triangledown` and `\bigtriangledown` are both defined as using the large one. This test also demonstrated us that we didn't have to waste time looking up what MATHML had to tell about it. A typeset version of that specification was never a visual highlight and missing glyphs only makes that worse. And, when fonts lack shapes no one uses them anyway.

However, it makes sense to think a bit about how to deal with this properly, and we will likely add some checking to the goodie files for it, so that when we do have them, we use them.[24]. But even then, most troublesome is that the size (and even positioning) of these symbols is rather inconsistent across math fonts, but because they are seldom used it doesn't make much sense to compensate for that (read: we just wait till users ask for it).

16.10 Punctuation

There are quite some punctuation symbols in UNICODE but not for math where the main troublemakers are the period, comma, colon and semicolon. The first two can be used as separator in numbers, in which case we don't want any spacing, or they can be part of a (pseudo) sentence in a formula, or they can separate entries in a list (take coordinates).

```
1.1 + 1.2
(1.1, 1.2)
x + 1.1, x + 1.2
```

[24] Most practical is to add this information to the character database which is a bit of work

When used as separator in a sentence, which is more likely in display math than in inline math, the spacing after it can be either regular (as in text) or wide. And the symbol can come from the math font or text (and these can actually look different). In ConTeXt (also pre LMTX) we have some special trickery at work for spacing comma's and periods but we leave that aside now. What should be noted is that out-of-the-box spaces are ignored when math is scanned so we cannot take that surrounding into account when dealing with spacing in the engine.

Although the Unicode specification provides a classification of characters that includes punctuation in practice we need to deal with it ourselves. For instance, by default a period is not considered punctuation but a command and semi colon are, while a colon is a relation!

Take for instance $f.$ (math italic f followed by a period). Italic correction and math glyphs have this special relationship and it also shows up in punctuation. Imagine that we have a sequence of characters, say fx. These are actually two ordinary atoms but in f, we have an ordinary atom followed by a punctuation atom so here spacing is determined by how these classes are set up. But, given the shape if the f we actually don't want italic correction here.

$$fx + f. + f, + f : + f; + a. + a, + a : + a; + x, + x, + x : + x;$$

When you zoom in you can see the subtle spacing differences. We can compensate for the semi colon being a bit higher than the period by applying some kern, something that we can set up in the goodie file.

Actually, if we assume that periods only occur in numbers we can make it punctuation and set it up for digit spacing but then commas etc also get done that way. A variant is to have two punctuation classes (or cheat and put the period in the digit class). No matter what we do, no help can be expected from documents mentioned: it's mostly a visual thing anyway.

Let's end with the visual aspect: in most fonts the two colons `0x003A` and `0x2236` are different: one has more distance between the periods. Which one? Well, that depends on the font! Latin Modern has a cramped `0x2236` while STIX2 has a cramped `0x003A`. Cambria has square dots for the 0x003A and round ones slightly more cramped for `0x2236`. Lucida goes extreme: it has smaller dots far apart for `0x2236`. If the idea is that a reader should get from the shape what it's about one can wonder if texts get read the way the author intended. Of maybe shapes don't matter. Of course a macro package can obscure these inconsistencies by setting the math character code of `0x003A` to `0x2236` but that only obscures the fact that little attention has been paid: what one can consider bugs became features.

16.11 Special ones

There are quite some characters that really depend on a math renderer. Examples are wide accents, fences, and arrows. Some constructs, like fractions use rules and these don't come from UNICODE nor fonts. A mixed case is radicals: there is a UNICODE point and fonts can provide larger variants. Normally one steps up a slightly slanted version but when things get large the radical becomes an extensible and therefore gets an upright shape. The engine is supposed to add a horizontal rule at the right location. Interesting is that there is no provision for a right end cap. The reason probably is that TEX, being the major renderer, has no combined horizontal and vertical extenders and OPENTYPE doesn't have that either. Some properties are driven by the fonts' math parameters which sort of makes the radical rendering a very restricted adventure: it is supposed to be used for roots only, either of not with a degree anchored in the right top area. It looks like that degree is not really to extend much beyond the left edge of the symbol.

In UNICODE there is an actuarian character U+20E7 and support in fonts is not that good. We do support it because we ran into in MATHML. However, it is a hack. The symbol as provided by fonts is rather useless.

```
$ \sqrt {x + 1} + \annuity{x + 1} $
```

Let's see how it renders:

$$\sqrt{x + 1} + \overline{x + 1}\rceil$$

We take the dimensions of a radical as template and when we look at the bare glyphs we see this:

$$\sqrt{\quad}\ \neg$$

Basically we have a right actuarian character like we have a left radical. But In this case the rule will go left instead of right. This is implemented on top of radicals so and driven by \Udelimited that takes two delimiters and doesn't scan for a degree. For two-sided roots (with degree) we have \Urooted. And like normal radicals the delimited one adapts itself to the content:

```
$ \sqrt {x + \frac{1}{x}} + \annuity {x + \frac{1}{x}} $
```

So we get:

$$\sqrt{x + \frac{1}{x}} + \overline{x + \frac{1}{x}}\rceil$$

For the record: in ConTeXt spacing is also driven by the struts and because we use the radicals renderer the gap and distance parameters also apply. It might look spacy, but keep in mind that we want radicals to look similar when we have more of them in line, and we can configure all. We have also enabled the feature that radicals at the same level are normalized in height and depth. Here are some variants:

```
$ \lannuity  {x + \frac{1}{x}} +
  \rannuity  {x + \frac{1}{x}} +
  \lrannuity {x + \frac{1}{x}} $
```

This gives:

$$\overline{\left(x+\frac{1}{x}\right)}+\overline{\left(x+\frac{1}{x}\right)}+\overline{\left(x+\frac{1}{x}\right)}$$

So we can have a mix of left, right and both end radical like symbols that encompass the nucleus. We're not aware of more such characters in Unicode but when they show up we are prepared. Only real usage can result in some parameters being fine-tuned.

16.12 Final words

This text was written in 2022 when we were working on math, extending the goodie files with new tweaks, checking support in fonts and updating manuals. But, as we moved forward, for instance with adapting Type1 support of Antykwa and Iwona to the new possibilities again we had to go back in time and figure out why actually things were done in certain ways. And I have to admit that we had some good laughs and quite some fun on seeing how strange and inconsistent the assumed structured and logical TeX ecosystem deals with math. A wrapup like is is never complete and we can keep adding to it so just consider it to be a momentary impression.

Personally I have to admit that I've always overestimated what happened outside the ConTeXt bubble, especially given the claims made. Consistency in Unicode math is probably not as good as is could have been and the same is true for OpenType math support, but maybe I'm naive in expecting consistency and logic in math related work. The mere fact that Donald Knuth pays a lot of attention to the math in his writing doesn't automatically translate in all TeXies doing the same. I don't claim that ConTeXt is doing better but I do hope that its users keep going for the best outcome.

One final note. In ConTeXt we always tried to keep up with developments and Unicode input as well as using OpenType math fonts are part of that. However, because we're not part of the *'gremia of TeX math and related coding'* it hardly matters what our opinions are with respect to these issues. The best we can do is adapt to whatever shows up, it being bad or good. It is however kind of funny to see (by now rusty) problems that have been noticed already long ago being presented as kind of new. Hopefully staying ahead and/or adapting with specific solutions doesn't'backfire to hard on the

CONTEXT users. If so, we're sorry for that. As long as they can render their documents well, it doesn't matter that much anyway. After all, we can always just blame *'the others involved'*.

16.13 Resources

[1] en.wikipedia.org/wiki/Slash_(punctuation)
[2] www.unicode.org/reports/tr25
[3] www.w3.org/TR/MathML3
[4] www.unicode.org/Public/math/revision-15/MathClass-15.txt
[5] en.wikipedia.org/wiki/Vertical_bar
[6] en.wikipedia.org/wiki/Dash
[7] en.wikipedia.org/wiki/Commercial_minus_sign
[8] en.wikipedia.org/wiki/Division_sign
[9] en.wikipedia.org/wiki/Bullet_(typography)

ConTeXt in TeXLive 2023 17

Starting with TeXLive 2023 the default ConTeXt distribution is LMTX, a follow up on MkIV, running on top of the LuaMetaTeX engine instead of LuaTeX. Already for a long time the MkII version used with pdfTeX, XeTeX and Aleph has been frozen and most users moved on from MkIV to LMTX (a more distinctive tag for what internally is version MkXL).

In principle one can argue that we now have three versions of ConTeXt and there can be the impression that they are very different. However, although MkXL can do more than MkIV which can do more than MkII, the user interface hasn't changed that much and old functionality is available in newer versions. Of course some old features make no sense in newer variants, like eight-bit font encodings in an OpenType font realm and input encodings when one uses utf, although we still support input encodings a.k.a. regimes. When we started using the `Mk*` suffixes the main reason was that we had to distinguish files and the official TeX distribution doesn't permit duplicate file names. Using a distinctive suffix also makes it possible to treat files differently.

suffix	engine	template	arguments	main file
`MkII`	pdfTeX, XeTeX, Aleph			`context.mkii`
`MkIV`	LuaTeX, LuaJITTeX, LuaMetaTeX			`context.mkiv`
`MkVI`	idem		yes	
`MkIX`	idem	yes		
`MkXI`	idem	yes	yes	
`MkXL`	LuaMetaTeX			`context.mkxl`
`MkLX`	idem		yes	

In this table *'template'* files are a mix of TeX and Lua and originate in the early days of MkIV; basically, they are a wink to active server pages. With *'arguments'* we refer to files that accept named macro arguments which means that they need to be preprocessed. That started as a proof of concept but some core files are defined that way. Users will normally just use a `.tex` file.

The Lua files in the code base have the suffix `lua`, or when meant for LuaMetaTeX that uses a newer Lua engine they can have the suffix `lmt`. There can also be `lfg` (font goodies) and `llg` (language goodies) plus byte-compiled files with various suffixes but these are normally not seen by users. We leave it at that.

So, while TeXLive 2022 installed MkII and MkIV, TeXLive 2023 installs MkIV and LMTX. Therefore the most significant upgrade is in the engine that is used by default: LuaMetaTeX instead of LuaTeX. The MkII files are no longer installed so we don't need pdfTeX.

So how did we end up here? Initially the idea was that, because LuaTeX is basically frozen, LuaMetaTeX would be the engine that we conduct experiments with and from which occasionally we could backport code to LuaTeX. However it soon became clear that this would not work out well so backporting is off the table now. Just for the record: the project started years ago so we're not talking about something experimental here. There have been articles in TUGBOAT about what we've been doing over the years.

One of the first decisions I made when starting with LuaMetaTeX was to remove the built-in backend, which then meant also removing the bitmap image inclusion code. That made us get rid of dependencies on external libraries. In fact, a proof-of-concept experimental variant didn't use the built-in backend at all. The font loading code could be removed as well because that was not used in MkIV either. In MkIV we also don't use the KPSE library for managing files so that code could be dropped from the engine tool; it can be loaded as so-called optional library if needed but I'll not discuss that here. If you look at what happens with the LuaTeX code base, you'll notice that updating libraries happens frequently and that is not a burden that we want to impose on users, especially because it also can involve updating build-related files. Another advantage of not using them is that the code base remains small.

A direct consequence of all this was that the build process became much more efficient and less complex. A fast compilation (seconds instead of minutes) meant that more drastic experiments became possible, like most recently an upgrade of the math subsystem. All this, combined with an overhaul of the code base, both the TeX and MetaPost part, meant that backporting was no longer reasonable. Being freed from the constraint that other macro packages might use LuaMetaTeX in turn resulted in more drastic experiments and adding features that had been on our wish list for decades. Another side effect was that we could easily compile native MS WINDOWS binaries and immediately support transitions to ARM-based hardware.

Instead of *"backporting after experimenting"*, a leading motive became *"fundamentally move forward"* while at the same time tightening the relation between ConTeXt and the engine: the engine code became part of the distribution so that users can compile themselves, which fits perfectly in the paradigm (and demands) of distributing all the source code, even that of the engine. There is also less danger that patches on behalf of other usage interferes with stable support for ConTeXt. A specific installation is now more or less long-term stable by design because it no longer depends on binaries and/or libraries being provided for a specific platform and operating system version. Of course installers and TeXLive do provide the binaries, so users aren't forced to worry about it, but they can move along with a system update by recompiling an old, and for their purpose, frozen ConTeXt code base.

An unofficial objective (or challenge) became that the accumulated source stays around 12 MB uncompressed, (compressed a bit over 2 MB) and the binary around 3 MB so that we could use the engine as an efficient LUA runner as well as a launcher stub, thereby removing yet another dependency. That way the official ConTeXt distribution didn't grow much in size. A bonus is that we now use the same setup for all operating systems. It also opened up the possibility of a exceptionally small installation with all bells and

whistles included. Another nice side effect, combined with automatic compilation on the compile farm, makes that we can provide installations that reflect the latest state of affairs: a recent binary combined with the latest ConTeXt. As a result, most users quickly went for LMTX instead of MkIV.

In the code base we avoid dependencies on specific platforms but there are a few cases where the code for MS Windows and Unix differs. However, the functionality should be the same. A good test is that for MS Windows we can compile with mingw (cross-compilation), MSVC (native) and clang (native); that order is also the order of runtime performance. The native MSVC binary is the smallest but users probably don't care. In any case, it is nice to have a fallback plan in place. The code is all in C; the MetaPost code is converted from cweb into C using a Lua script but we also ship the resulting C code. The code base provides a couple of cmake files and comes with a trivial build script.

When I say that there are no libraries used, I mean external libraries. We do use code from elsewhere: adapted `avl` as well as `decnumber` (for the MetaPost library), adapted `hjn` (hyphenation), `miniz` (zip compression), `pplib` (for loading pdf files), `libcerf` (to complement other math library support, but it might be dropped), and `mimalloc` for memory management. However all the code is in the LuaMetaTeX code base and only updated after checking what changed. The most important library originating elsewhere is of course Lua: we use the latest and greatest (currently) 5.4 release. We kept the `socket` library but it might be dropped or replaced at some point. In addition there is a subsystem for dynamically loading libraries; the main reason for that being that I needed `zint` for barcodes, interfaces to sql databases, a bunch of compression libraries, etc. But all that is tagged *optional* and ConTeXt will never depend on it. There are no consequences for compilation either because we don't need the header files. The glue code is very minimalistic and most work gets delegated to Lua.

Initially, because the backend is written in Lua, there was a drop in performance of some 15% but that was stepwise compensated by gains in performance in the engine and additional or improved functionality. The ConTeXt code base is rather optimized so there was little to gain there, apart from using new features. Existing primitive support could also be done a bit more efficiently; it helps if one knows where potential bottlenecks are. Therefore, in the meantime an LMTX run can be quite a bit faster than a MkIV run and it can even outperform a LuaJITTeX run. In practice, the difference between an eight-bit MkII run using the eight-bit pdfTeX engine and a 32-bit LuaMetaTeX run with LMTX can be neglected, definitely on more complex documents. I never get complaints about performance from ConTeXt users, so it might be a minor concern.

So what are the main differences in the installation? If you really want to experience it you should use the standard installation. Currently the small installer is the engine that synchronizes the installation over the net and, assuming a reasonable internet connection, that takes little time. The installation is relatively small, and many of the bytes used are for the documentation. Updates are done by transferring only the changed files. The TeXLive installation is a bit larger because it shares for instance fonts with the main installation and these come with resources used by other macro packages.

Both installations bring MkIV as well as LMTX and therefore provide LuaTeX as well as LuaMetaTeX. However, a MkIV run is now managed by LuaMetaTeX because we use that engine for the runner. The MkII code is no longer in TeXLive but is in the repositories and used to test and compare with pdfTeX. It just works.

The number of binaries and stubs is reduced to a minimum:

file	**symlink**	
`tex/texmf-platform/luametatex`		combined TeX, MetaPost and Lua engine
`tex/texmf-platform/mtxrun`	`luametatex`	script runner, binary
`tex/texmf-platform/context`	`luametatex`	CONTEXT runner, binary
`tex/texmf-platform/mtxrun.lua`		script runner, lua code
`tex/texmf-platform/context.lua`		loader for ConTeXt runner
`tex/texmf-platform/luatex`		the good old ancestor

All of these programs are in the ConTeXt distribution directory `tex/texmf-<platform>/`. In addition, `context` and `mtxrun` are symlinks to the `luametatex` binary, where possible.

So, the `context` command runs `luametatex`, but loads the Lua file with the same name which in turn will locate the ConTeXt management script (`mtx-context`) in the TeX tree and run it. The same is true for `mtxrun`: it is a binary (link) that loads the script in (this time) the same path and then can perform numerous tasks. For instance, identifying the installed fonts so that they can be accessed by name is done with:

```
mtxrun --script font --reload
```

Where in MkII we had stubs for various utility scripts, already in MkIV we went for a generic runner and a bit more keying. It's not like these scripts are used a lot and by avoiding shortcuts there is also little danger for a mixup with the ever-growing list of other scripts in TeXLive or commands that the operating system provides.

The LuaTeX binary is optional and only needed if a user also wants to process MkIV files. There are no shell scripts used for launching. The two main calls used by users are:

```
context foo.tex
context --luatex foo.tex
```

A user has only to make sure that the binaries are in the path specification. When you run from an editor, the next command does the work:

```
mtxrun --autogenerate --script context <filename>
```

with `<filename>` being an editor-specific placeholder. Like other engines, LuaMetaTeX (and ConTeXt) needs a file database and format file, and although it should generate these automatically you can make them with:

```
mtxrun --generate
```

```
context --make
```

The rest of the installation is similar to what we always had and is TDS compliant. The source code of LuaMetaTeX is included in the distribution itself (which nicely fulfills the requirements) but can also be found at:

https://github.com/contextgarden/luametatex

There are also some optional libraries there but ConTeXt works fine without them. The official latest distribution of ConTeXt itself is:

https://github.com/contextgarden/context
https://github.com/contextgarden/context-distribution-fonts

We see users grab fonts from the Internet and play with them. They can install additional fonts in `tex/texmf-fonts/data/<vendor>`. Project-specific files can be collected in `tex/texmf-project/tex/context/user/<project>`. These directories are not touched by installations and can easily be copied or shared between different installations. After adding files to the tree `mtxrun --generate` will update the file database.

In the distribution there are plenty of documents that describe how LuaMetaTeX with LMTX differs from MkIV with LuaTeX: new primitives, macro extensions, more granular math rendering, improved memory management, new (or extended) (rendering) concepts, more MetaPost features; most is covered in one way or another, and much is already applied in the ConTeXt source code. After all, it took a few years before we arrived here so you can expect substantial refactoring of the engine as well as the code base, and therefore eventually there is (and will be) more than in MkIV.

When you compare a ConTeXt installation with what is needed for other macro packages you will notice a few differences. One concerns the way TeX is launched. An engine starts with a blank slate but can be populated with a so-called format file that is basically a memory dump of a preloaded macro package. So, the original way to process a file is to pass a format filename to the engine. In order to avoid that a trick is used: when an engine (or symlink/stub to it) is launched by its format name, the loading happens automatically. So, for instance `pdflatex` is actually an equivalent for starting pdfTeX with the format file `pdflatex.fmt` while `latex` is pdfTeX with another format file (`latex.fmt`) starting up in DVI mode. And, as there are many engines, a specific macro package can have many such combinations of its name and engine.

In ConTeXt we don't do it that way. One reason is that we never distinguished between backends: MkII uses an abstract backend layer and load driver files at runtime (it was one of the reasons why we could support Acrobat as soon as it showed up, because we already supported the now obsolete but quite nice dviwindo viewer). And that model hasn't changed much as we moved on. Because we use a runner, we also don't need to distinguish between engines: all formats have the same name but sit on an engine subpath in the TeX tree. Anyway, this already removes quite some formats. On the other hand, ConTeXt can be run with different language specific user interfaces

which means that instead of just `context.fmt` we have `cont-en.fmt` and possibly more, like `cont-nl.fmt`. So that can increase the number again but by default only the English interface is installed. As a side note: where with MKII we needed to generate METAPOST mem files, with its descendants having MPLIB we load the (actually quite a bit of) METAPOST code at runtime.[25]

In addition to a format file, for the LUATEX and LUAMETATEX engine we also have a (small) LUA loader alongside the format file. All this is handled by the runner, also because we provide extensive command line features, and therefore of no concern to users and package maintainers. However, it does make integrating CONTEXT in for instance TEXLIVE different from other macro packages and thereby puts an extra burden on the TEXLIVE team. Here I want to thank the team for making it possible to move forward this way, in spite of this rather different approach. Hopefully a LUAMETATEX integration is a bit easier in the long run because we no longer have different stubs per platform and at least the binary part now has no dependencies and only has a handful of files.

For those new to CONTEXT or those who want to try it in TEXLIVE 2023 there is not much difference between the versions. However, MKIV is now frozen and new functionality only gets added to LMTX. Of course we could backport some but with most users already having moved on, it makes no sense. Just as we keep MKII around for testing with PDFTEX, we also keep MKIV alive for testing with LUATEX. Maybe in a couple of years MKIV will go the same route as MKII: ending up in the archives as an optional installation.[26]

[25] Occasionally I do experiments with loading the TEX format code at runtime, but at this moment the difference in startup time of about one second (assuming files are cached) is too large and running over networks will be less fun, so the format file will stay. The time involved in loading METAPOST can be brought down but for now I leave it as it is.

[26] This text appeared in TUGBOAT around the 2023 TEXLIVE release. Thanks to Karl Berry for his careful reading and fixing of the text and of course for keeping TEXLIVE alive.

How not to install ConTeXt 18

Installing LuaMetaTeX can be a complex process that requires some technical expertise, but the following steps should give you a general idea of what is involved:

1. First, you need to ensure that you have a recent version of the Lua programming language installed on your system. You can download the latest version of Lua from the official website at http://www.lua.org/download.html.[27]

2. Next, you need to download the latest version of the MetaTeX distribution, which includes the LuaMetaTeX engine, from the official ConTeXt Garden website at https://wiki.contextgarden.net/ConTeXt_Standalone.[28]

3. Once you have downloaded the MetaTeX distribution, extract the files to a directory on your system.[29]

4. You can then run the LuaMetaTeX engine by opening a command prompt or terminal window and navigating to the directory where you extracted the MetaTeX files. From there, you can run the command "luametatex" followed by the name of the TeX file you want to process.[30]

5. To make it easier to use LuaMetaTeX with your favorite text editor, you may also want to install a TeX distribution such as TeX Live or MiKTeX, which includes support for LuaMetaTeX. These distributions typically include a graphical user interface that makes it easier to manage your TeX installation and configure your system for use with LuaMetaTeX.[31]

It's worth noting that the exact steps for installing LuaMetaTeX may vary depending on your operating system and the specific TeX distribution you are using. For more detailed instructions, you may want to consult the official documentation for LuaMetaTeX and the TeX distribution you are using.[32]

by ChatGPT[33] [34]

---

[27] The LUA code needed is part of the source tree that can be downloaded from GitHub or websites.
[28] There is no METATEX, although we sometimes joke about it.
[29] So here one is stuck.
[30] Shouldn't it be compiled first? And even then it needs some format, so one needs `context` and `mtxrun`.
[31] Indeed installing TEXLIVE is easier, as is installing the smaller reference installation which uses LUAMETATEX as its own installer. As far as we know, MIKTEX doesn't include LMTX. And yes, consulting the documentation might be best.
[32] So here is the way out of the proposed mess.
[33] Queried by Mikael Sundqvist.
[34] Which (at least here) is more about well formed sentences than about verified content. We can only hope that new TEX users are able to recognize a fake.

Strange assumptions 19

Below I will collect some of the questions and remarks-turned-questions that keep popping up and start annoying me, especially when they come from people who should know better (being involved in development themselves). I'm always puzzled why these things come up, especially by people who are no user and should not waste time on commenting on CONTEXT.

All these versions, CONTEXT keep changing, so what's next?

Sure, we're now at the third version, MKII, MKIV and LMTX, but there is is some progression in this. The first version evolved from TEX to ε-TEX to PDFTEX (but also could handle XETEX and ALEPH). But in order to get things done better we moved on to LUATEX and because that is a CONTEXT related project it made sense to split the code base which made us end up with a frozen stable MKII and an evolving-with-LUATEX MKIV. Then there was a demand for a stable LUATEX for usage otherwise which in turn lead to the LUAMETATEX project and its related CONTEXT evolution LMTX. So, yes, this macro package keeps changing. And it this bad? Don't other macro packages evolve? And why do users of other packages bother anyway? I never heard a CONTEXT user complain either. By the way, how do other macro packages actually count and distinguish versions?

Why is CONTEXT so slow?

Because I seldom hear complaints from users about performance, why do users of other macro packages find reason to even bother. In MKII we immediately started with a high level keyword driven interface so that came with a price. But quite some effort was put into making it as fast and efficient as possible. Fortunately for CONTEXT users the MKIV version became faster over time, in spite of it using a 32 bit engine (which comes at a price). Even better is that LMTX with LUAMETATEX has gained a lot over MKIV. But then, I guess, other macro packages that use LUATEX are also fast, so maybe the claims that CONTEXT is much slower than other macro packages still hold. I'm not going to check it, and I bet CONTEXT users don't care.

Why does CONTEXT (even) needs a runner.

Indeed, because we don't want users to be bothered with managing runs right from the start it came with a program (MODULA2) and later a script (PERL followed up by RUBY) that checks if an additional run is needed because of some change in the table of contents, references, the index, abbreviations, positioning, etc. Index sorting was done too so there was no further dependency. We though that was actually a good thing. With LUATEX and LUAMETATEX all that became even more integrated because LUA was used. The runner(s) also made it possible to ship additional scripts without the need for potentially clashing applications in the ever growing TEX ecosystem. Interesting is

that ridiculing ConTeXt for script dependency was never complemented by ridiculing other macro packages that nowadays seem to depend on scripts (with some even using LuaTeX which originates in the ConTeXt domain).

Why does ConTeXt organizes files that way?

ConTeXt sticks quite well to the TeX Directory Structure, so what is the problem here?. Yes, we needed some granularity (e.g. for MetaPost) but later that just became normal. And indeed we optionally let users use a flat directory structure for fonts but that's normally in the users own local tree. Oh, and in MkIV and LMTX we use our own file database (actually also in MkII at some point), just because (definitely at that time) it was way faster and we needed more features. The same is true for the font database, UTF encoded hyphenation patterns, and so on. Can it be that we're often just ahead of the pack?

Let's nor forget to complain about the fact that MkIV and LMTX use a cache but so do lots or programs: just think browsers of some scripting language ecosystems. And that was introduced right after we started with MkIV and hasn't changed much at all. Users expect no less. And other macro packages are free not to use the cache (for e.g. fonts).

The authors of ConTeXt don't care about compatibility, do they?

You're joking, right? Surely some features became sort of obsolete when we moved to MkIV, like encodings. But if users like to stick to them, they can. Do you really think that user like us to drop compatibility? Maybe it fits some narrative to spread that story. Of course, we make things better if we can, and the interfaces have always permitted upgrades and extensions. There are definitely cases when (maybe due to user demand) something new gets added that then evolves towards a stable state, so yes, there can be code in flux. But that is natural. Should we just assume that other macro packages don't evolve, never have bugs, don't break anything, never fix broken things immediately? Maybe. And complaining about ConTeXt evolving is none of its non-users business anyway.

Is ConTeXt commercial?

This is one of the strangest questions (or remarks). We use ConTeXt ourself and using it in a job is by definition commercial use. Are all other TeXies only using TeX macro packages in the free time, as hobby? I'm pretty sure that more money is made by competing package users and I'm also sure that most of the time involved in creating ConTeXt (and LuaMetaTeX for that matter) is not covered by income. Using the fact that ConTeXt is developed by a (small) company excuse for lack of development elsewhere is about as lame as it can get. Much development is done without us needed it, but because we like doing it, because of the challenge.

Should I use ConTeXt for math?

Of course, because that's what TeX is good at. It you are forced to use a specific macro package for its math abilities, just do so. If you want to move on or want consistent in-

terfaces, maybe CONTEXT is for you. We don't care. Trust your eyes more than assumed standards or ways of doing math typesetting.

Why is the format file so much larger than for other packages?

The answer is simple: we have an integrated system, so we have plenty macros and with each token taking 8 bytes (data and link) that adds up. And for MKIV and LMTX there also LUA code involved as well as a rather large character database. In LUATEX the format file is compressed (and also zipped) and in LUAMETATEX is it is a bit more compressed but now zipped; still the LMTX format file is smaller than the MKIV one. We let those who complain wonder why that is. We also let users of other macro packages wonder if loading a ton of stuff later on doesn't accumulate to a similar or larger memory footprint. And, as with many critics: make sure to check every few years if that other macro package hasn't catched up and can be criticized the same way.