TEX
78

TEX
82

TEX
89

Back to the basics

1 Introduction

When Mikael told me that there were enhanced lectures by Don Knuth, pushed online at Stanford, I decided to have a look at it.[1] We're talking of two series: "Advanced TEXarcana" (1981, so between the SAIL[2] and TEX82 version) and "The Internal Details of TEX82" (during early TEX82 development). After all, it's a nice distraction when one is supposed to be in documentation mode. It is also a good retrospective on the period around the 80's when TEX was actively developed, tested on real documents, and when more people became interested in using it, evolved. So I sat down (or walked around), listened, and took some notes.[3]

When watching these videos one has to keep in mind that the version of TEX and its default format plain TEX were not yet finalized, even if there's mentioning of the books being released any time (next year). Rather soon one starts to notice that there are references to primitives and macros that predate the standard TEX engine and plain format. For instance instead of 'plain' there is 'basic'. Some examples and concepts assume a different TEX than one knows today and it's interesting to where all we know and use now came from. These lectures are part of the development cycle of many years and also a clear demonstration of how software development took place: in an academic setting without commercial pressure. With user feedback, discussions, and no problems going back to the drawing board. It all resulted in TEX as we know it today, and although there have been extensions (enhancements) the principles remain the same.[4]

Before we continue, let's guess about the timeline here; please correct me if I'm wrong.

- In the late 70's Don Knuth started working on what became TEX. The reasons for this (quality) are explained in various places but of course making sure that his books look great was the main objective.
- The program was written in SAIL, running on a Digital Equipment infrastructure, but later Pascal was used. There are references to other hardware setups (at other universities) and in one talk SUN workstations are mentioned which illustrates the move from terminals to more integrated high resolution setups.

[1] He got the link to the upgraded videos from Barbara Beeton who was present at one or more of these lectures.

[2] SAIL refers to the "Stanford Artificial Intelligende Laboratory", and references to this acronym refer to the mainframe at that place and its operating system variant. On Wikipedia search for `WAITS` as it also refers to early days media usage.

[3] This wrapup is part of the musing series but rendered independently because of its length.

[4] The Lua project has a similar vibe which is among the reasons that we've chosen it as the extension language for LuaTEX.

- The program was of course initially meant for producing books, and with that in mind it started out as a production tool. Much was hard-coded but some was implemented in a macro package called 'basic'. In these years we see the language evolve to be more flexible: there came additional programming related features and hard coded assumptions became configurable.
- At some point a reimplementation was started, this time in Pascal, generated from web files. A more detailed documentation of how things work conceptually are part of the process and stepwise building up the program paradigm. The mature web suite is part of the outcomes of the TEX and MetaFont project.[5]
- The arcana presentations took place around that time, so they cover the prototype(s) as well as the early TEX82 versions. Between 1983 and 1989 TEX became more and more stable and eventually it was declared finished and frozen, except for bugs periodically being fixed.
- Then TEX82 started being used and a period of development, documentation and experimenting started. Various students were involved in subsystems.
- The presentations about the internal details happen in the middle of this evolution, or maybe a better term is transition. There is still talk of a basic format, some primitives will later change, the book we mention here dates from that time, the books mentioned being written are the multi-column set about TEX, MetaFont and the fonts.[6]
- From 1982 to 1989 the versions evolve and in the tape dumps of the by then retired SAIL machines we see less and less changes. We also see macro packages pop up and evolve. The fonts mature as well.

The above is more or less noticeable in the presentations. For instance, the many references to "loading basic" made me curious to what this prehistoric TEX format actually looked like, if only to figure out what references to `\jpar` actually mean (given the context one can guess here). The macros, actually a relative small set compared to plain, and tiny compared to for instance ConTEXt MkXL, can be found in "TEX and MetaFont, New Directions in Typesetting", published by the now defunct Digital Equipment, the company that made the main frame and mini computers that TEX was developed on and that also are mentioned (and used) in the presentations.

Watching all that comes with some nostalgic feelings, because I grew up with those machines too (DEC 10 and 20, as well as VAX), did quite a bit of Pascal and Modula2 programming, used the line printers, slow tube terminals, went from 300 to 1200 baud

[5] Because Pascal became less popular and was not available on all platforms, at least not in compatible ways, Knuth later switched to C and therefore cweb. The web2c conversion introduced later converts Pascal code into C which then compiles into the programs distributed in for instance TEXLive.

[6] This first book, discussing digital typography, basic TEX and infant MetaFont, has header lines wrapped in a frame. In those days it was actually quite common (in educational documents) to put rules around things, maybe just because it could be done. It might have been a left-over from the typewriter days when little was possible and with upcoming systems suddenly one could draw rules. The later TEXbook series looks way better! But still, quite often extensively framed tables show up in documents produced by TEX. Maybe because the core TEX engine is pretty much limited to glyphs and rules out of the box.

modems, and even played with these GIGI terminals. Being totally unaware of something TeX running on these machines I even wrote some programs that made pages from ascii input, including generating tables of contents, doing some itemization, move around some space for glued in images, and page numbering.

So, a lot of bells ring when seeing the black and white videos, filmed just before color video entered academia, eventually to be replaced by the Internet. And yes, the first personal computers also showed up, but again, no TeX. Anyways, looking back it is amazing that something TeX took off so well, especially given what users nowadays take for granted with respect to performance, editing and previewing, and are sometimes willing to complain about.

But, let's move beyond the sentimental reflections and have a look at this basic format because that kind of pictures the landscape. This is not a tutorial so no details will be explained but I'll revive some of what can be read in this first TeX manual because it's not something you can pick up in a bookshop. The book actually is a nice read, with plenty of humor sprinkled in.

2 Some observations

We will use the basic format to explain a bit what this first version was about but before we come to that it might be fun to filter out some distinctive differences with today's TeX. Let's start with a quote:[7]

> "Those of you who wish to define control sequences should know that TeX has further rules about them, namely that many different spellings of the same control sequence may be possible. This fact allows TeX to handle control sequences quite efficiently; and TeX's usefulness is not seriously affected, because new control sequences aren't needed very often. A control sequence of the first kind (i.e., one consisting of letters only) may involve both upper case and lower case letters, but the distinction between cases is ignored after the first letter. Thus `\TEX` could also be typed "`\TEx`" or "`\TeX`" or "`\Tex`" — these four have the same meaning and the same effect. But "`\tex`" would not be the same, because there is a case distinction on the first letter. (Typing "`\gamma`" results in γ, but "`\Gamma`" or "`\GAMMA`" results in Γ.)"

In one presentation Knuth explains that Pascal has some limitations on the length of an identifier: eight, and therefore he played safe by limiting them to seven unique initial characters in his code. Maybe that inspired him to come up with the feature mentioned here. The question is why it was done this way. I assume that efficiency here refers to hashing and resolving control sequences because normalizing takes runtime too. Once stored in the format (or memory) the control sequence is a token and therefore just a number, so any length or case property is gone. It is one of the reasons why TeX is

[7] There is a list of named characters in the manual but they are not defined in the basic format, so I guess that there is an extra file that defines them.

fast and efficient! However, keep in mind that tokens as concept were not in the first implementation, there it was about sequences of characters (which then is slower).

But more interesting is the assumption that not that many new control sequences are needed. Tell that to a 2025 TEX user or macro package writer: it just wouldn't work out today. Of course this became clear pretty soon so this feature was dropped in the follow up. It was mentioned in a dangerous bend section that when a reader dared reading that, lead to a next bend:

> "Another rule takes over when there are seven or more letters after the escape: all letters after the seventh are replaced by "x", and then groups of eight letters are removed if necessary until at most 14 letters are left. Thus `\underline` is the same as `\underlixx`; and it is also the same as `\underlinedsymbols` or any other control sequence that starts with `\u` followed by n or N, then d or D, then e or E, then r or R, then l or L, then i or I, then 2 or 10 or 18 or 26 or ⋯ letters. But `\underline` is not the same as `\underlines`, because these two control sequences don't have the same length modulo 8."

I'm sure you "get this" in one read, but try to explain that to a confused user who defined a ton of macros after having managed to bump TEX's memory to the extremes of those days.[8] But, as usual, Don's wit kicks in when he makes us go into the next dangerous bend:

> "... Thus the total number of distinct control sequences available is exactly
>
> $$128 + 52 \bullet 26 + 52 \bullet 26^2 + 52 \bullet 26^3 + 52 \bullet 26^4 + 52 \bullet 26^5 + 8 \bullet 52 \bullet 26^6 = 129151507704$$
>
> that should be enough ..."

In the web file we can read this (and best keep this in mind when we progress in this wrap-up):

> "The present implementation has a long ancestry, beginning in the summer of 1977, when Michael F. Plass and Frank M. Liang designed and coded a prototype based on some specifications that the author had made in May of that year. This original protoTEX included macro definitions and elementary manipulations on boxes and glue, but it did not have line-breaking, page-breaking, mathematical formulas, alignment routines, error recovery, or the present semantic nest; furthermore, it used character lists instead of token lists, so that a control sequence like `\halign` was represented by a list of seven characters."

Watch the fact that there was no token list (as we know it now) yet but a sequence of characters. So storing a macro name was costly. Of course a list of records with two

[8] Just imagine explaining `\big`, `\Big`, `\bigg` and `\Bigg` versus the impossible `\Big`, `\BIg` and `\BIG` when only the first capital is distinctive.

numbers (the token and the pointer to the next token) also takes space. This might be the reason why we have the collapsing of names mentioned above: it had to fit into memory (and format files, which are kind of memory dumps) efficient.

> "A complete version of TEX was designed and coded by the author in late 1977 and early 1978; that program, like its prototype, was written in the SAIL language, for which an excellent debugging system was available. Preliminary plans to convert the SAIL code into a form somewhat like the present "web" were developed by Luis Trabb Pardo and the author at the beginning of 1979, and a complete implementation was created by Ignacio A. Zabala in 1979 and 1980."

It's here the transition from basic to plain happened. The old book discusses the basic format, but the final TEX reference talks plain.

> "The TEX82 program, which was written by the author during the latter part of 1981 and the early part of 1982, also incorporates ideas from the 1979 implementation of TEX in MESA that was written by Leonidas Guibas, Robert Sedgewick, and Douglas Wyatt at the Xerox Palo Alto Research Center. Several hundred refinements were introduced into TEX82 based on the experiences gained with the original implementations, so that essentially every part of the system has been substantially improved. After the appearance of "Version 0" in September 1982, this program benefited greatly from the comments of many other people, notably David R. Fuchs and Howard W. Trickey. A final revision in September 1989 extended the input character set to eight-bit codes and introduced the ability to hyphenate words from different languages, based on some ideas of Michael J. Ferguson."

By 1989 the program was pretty stable, and we see little changes in for instance plain TEX in the archives. When listening to the talks, especially from session 9 onward, where files are discussed, you will notice that on the one hand there is this 127 characters 'limitation' imposed, very likely a side effect of the Computer Modern fonts having at most that many glyphs, although on the other hand the tfm and dvi formats are capable of more. So, going 256 was no big deal here, and it's the European user groups that took the opportunity to use the extended repertoire for enhanced versions of the fonts, which in turn resulted in different encoding schemes, that itself in turn had consequences for e.g. hyphenation patterns.

Another interesting remark in the book is the following.

> "When a space comes after a control sequence (of either kind), it is ignored by TEX; i.e., it is not considered to be a 'real' space belonging to the manuscript being typeset. Thus, the example above could have been typed as `George P\' olya and Gabor Szeg\" o.` TEX will treat both examples the same way; it always discards spaces after control sequences."

This is something that changed over time, these hundreds of refinements, which shows that this multi-year project was driven by perfection, user feedback and practical thinking. A simple example shows what happens today:

```
\def\`{!} \` \` \`
\def\f{!} \f \f \f
```

It will give you: ! ! ! !!!, so the explanation in the book (and presentations) of how spaces are handled when a control sequence is read nowadays adapts to the assigned catcodes. You have to keep in mind that a control sequence normally was only made from letters, the single character ones that use a non-letter were primitives.

This quote definitely demonstrates how the constraints of those days were dealt with, and just as watching an old SciFi movie with tubes as displays makes one wonder why if flat panels were not predicted, it's clear that foreseeing computers to become 1000 or more times faster and memory going gigabytes was not easy.

> "When you do use the `\:` instruction to change fonts, here are the rules you need to know. TEX can handle up to 32 different fonts in any particular job (counting different sizes of the same style). These 32 fonts are distinguished by the least significant five bits of the 7-bit ascii character code you type following "`\:`"; if you don't understand what this means, use the following code names for your fonts: *table omitted*. You never refer to a font by its number, always by its code. Code `A` is treated the same as `a`, etc.; but a wise typist will consistently use the same codes in any particular manuscript, because later TEXs may allow more than 32 fonts."

Indeed, today's TEX implementations go way over 32 fonts, use multi-byte names, ship macro packages that provide complex font setups, and what more. This of course already had changed in the final version of TEX.

> "Groups within groups will happen only in rather complicated situations, but in such cases it is extremely important that you don't leave out a { or } lest TEX gets hopelessly confused."

It is these (dangerous bend) comments that makes reading this old document worthwhile! Although nesting groups 300 deep is not something that happens often, but processing this document in ConTEXt brings us up to 11 levels, while for instance the LuaMetaTEX manual tops at 25. Maybe in the end macro packages indeed became too complex.

The way a horizontal list becomes a paragraph of lines remained conceptually mostly the same. The formulas changed a bit and some more features were added. In a way this is also how macro packages later evolved: driven by demand. In one presentation, when explaining some exercises and answering questions, Knuth mentions that solutions to problems can interfere with each other in a larger setup and indeed this is where writing a large macro package puts some stress on the authors: integration. The par builder is an example of where (maybe conflicting) demands meet as we found out when we extended the builder in LuaMetaTEX. We felt better after hearing him express in one talk that some parts of the par builder took some iterations to become perfect and that the first prototype version had a (what could be considered) a bug that no one had noticed as on the average the results were fine (so today one would just say it was a feature later to be improved). It's no surprise that when we extended the code we had to do some hard thinking. There's a reason why the original (reference) source has some warnings against changes.

The first version of TEX already had the basic algorithms but more was hard coded. There is a `\tolerance` like parameter but as far as I can see no `\pretolerance` and `\emergencystretch` but there are a few integer parameters that drive the process. Justification is basically limited to flush right with a parameter that determines the raggedness. There are no `\leftskip` and `\rightskip` yet and likely for a good reason: typesetting books is the target. He demonstrates that by messing around with glue in special ways one can achieve left and right ragged effects. In one talk Knuth answers a question about stretch between characters in a word with the remark that this is not what typographers want. If you look at pdfTEX or macro packages on top of LuaTEX you might encounter additional inter-character kerning, something that sometimes makes sense in titling, but so far the urge to add inter-character glue had been suppressed.[9]

An interesting difference with final TEX is that spaces after the right brace that ends the body of a definition (`\def`) are ignored. That for sure got rid of what we call spurious spaces but the final TEX is more consistent in no longer doing that. I think that dealing with spaces and line endings has always been somewhat tricky so it is no wonder this evolved.

There are of course dimensions, and they have the usual units. However, there is no `ex` yet, nor `sp`. The mentioned repertoire is: `pt`, `pc`, `in`, `cm`, `mm`, `dd` and `em`. Do you see what more is missing?[10] At the start there were no dimension registers, only some counters, but when registers showed up it also came with some changes in primitives that expect dimensions.

Space factors are discussed in a way that suggests that they are hard coded which is likely due to the fact that the focus is on Latin scripts with well known punctuation code points. Later that became configurable but the approach remained the same.

In fact quite a bit is still hard coded like the way hyphenation hooks into the par builder. However, in appendix H the book suggests that there are built-in rules while the web file shows the ability to load pattern files efficiently. So here we are in some middle ground between old and new I guess; both identify themselves in the web files as TEX82 anyway.[11]

A substantial bit of the book and talks is about error message and interacting with the system, which is understandable given the systems used at that time. We have ConTEXt

9 Implementing this is actually rather easy but as it won't get used it only adds overhead and we don't feel the need to prove that it can be done.

10 There are in places references to `sp` so it must have been there at some point.

11 When you look at (or listen to) the way hyphenation patterns are made and applied, and when you realize that we're at the Artificial Intelligence lab, you can actually consider this to be an an example of machine learning. After all we have lookups driven by weights stored in a compact form.

configured to just quit at an error, report the location, and users can then fix the issue and run again. Hitting a button in the editor triggering a few seconds run makes more sense. But I do remember the times when it was better to note down the issue and hit return to see if TEX could catch up.[12]

Actually, one nowadays needs ways to inform the user about issues because reading the 'transcript' is often not done. In this context it was interesting to hear on one presentation Don mentioning that a `.tex` file produces a `.dvi` and `.err` file. I made a mental note to mention this 'interesting' and somewhat depressing suffix here, but then was amused by the fact that in a next presentation that had become `.log` because of user feedback. Today the problem is that the (progress, diagnostic, warning) messages scroll by so fast that users probably don't notice them. Do they go to the log file?[13]

I tried to locate where the single character primitives were defined and ended up at the interesting section that describes the magic look up trickery of inter-math element spacing. As the comments mention `\thinmskip` (later to become `\thinmuskip`) and such, it looks like the shortcuts are already gone, although `\quad` is still hard coded. So that manual predates the web version that has the comment "TEX version 0.25 as it existed when I gave twelve lectures on the internal details of TEX82 in July 1982", the videos that is. Although 'basic' still fits the talks, and that is what Knuth refers to, progress was made real time in 1982!

3 The basic format

One thing that you will notice, also in the presentations, is that in addition to ascii, which is mentioned explicitly, Don loved to use some special symbols, like `←` for an assignment instead of an `=`. In plain TEX that is gone although the file starts with some mappings of Don's favorite keyboard, using the pre-space range of characters. Watch the use of octal here:

```
\chcode'173←1 \chcode'176←2 \chcode'44←3 \chcode'26←4
\chcode'45←5 \chcode'43←6 \chcode'136←7 \chcode 1←8
```

The comment sign is the same so we need a macro to typeset it:

```
\def\%{\char'45 } % Note, the space after 45 is needed! (e.g.\%0)
```

Commands often have short names although nowadays one will not see these:

[12] In this perspective I like to notice that when we moved from pdfTEX to LuaTEX (and later LuaMetaTEX) processing the MetaFun manual with thousands of graphics, color, etc. went down from 15 minutes to below 20 seconds. Compiling a 250 page book from xml input (also combined with multiple MetaPost graphics per page) that could take 4 runs and accumulated to 45 minutes, went below a minute. Typesetting a 300 page complex educational math book with some three thousand formulas takes 6 seconds on a 2025 Chromebook. Nowadays processing the TEX book takes less than a second so imagine what it took in those days. It anyway demonstrates that the efficient original implementation still pays off.

[13] It is why in ConTEXt we explore the possibilities to visual feedback via a connected device.

```
\def\lft#1{#1\hfill}
\def\ctr#1{\hfill#1\hfill}
\def\rt#1{\hfill#1}
```

Here we see a feature not present in TeX, the size keyword, which stands for the current horizontal size. Another interesting observation is the extreme value of `1000cm`, something not possible today. But at least it's metric.

```
\def\rjustline#1{\hbox to size{ % newline therefore space
    \hskip0pt plus1000cm minus1000cm #1}}
\def\ctrline#1<\hbox to size{\hskip0pt plus1000cm minus1000cm
    #1\hskip0pt plus1000cm minus1000cm}}
```

There is not yet a repertoire of `\tracing...` parameters but a single reference to an internal parameter zero. In one talk you see an (octal) value assigned to it, we're talking bits and bytes here. The first integer variable reflects the tolerance and number eight does something with raggedness. An example in the book uses a value of 1000 as example but we can't test what that does.

```
\def\trace{\chpar0←} \def\jpar{\chpar1←} \def\ragged{\chpar8←}
```

although we won't go into much detail, it is nice to know how it all started out. The 'tolerance' is set by a multiplier:

> "The number 200 used to determine feasibility can be changed to 100n for any integer $n > 1$ by typing "`\jpar <number>`", where n is the specified number. A large value of n will cause TeX to run more slowly, but it makes more line breaks feasible in cases where lines are so narrow that $n = 2$ finds no solutions."

Watch how we can make a paragraph more ragged by changing a parameter:

> "The instruction `\ragged <number>` specifies a degree of "raggedness" for the right-hand margins. If this number is r, the line width changes towards its natural width by the ratio $r/(100 - r)$. Thus, `\ragged 0` (the normal setting) gives no raggedness; `\ragged 100` causes the width of each line to be midway between `\hsize` and its natural width; and `\ragged 1000000` almost completely suppresses any stretching or shrinking of the glue. Some people like to use this "ragged right margin" feature in order to make the output look less formal, as if it hadn't actually been typeset by an inhuman computer. (Some people also think that "ragged right" typesetting saves money. On traditional typesetting equipment, this was true, but computer typesetting has changed the situation completely: the most expensive part of the computation is now the breaking of lines, while the setting of glue costs almost nothing.)"

Currently this is done by a combination of tolerance and left and/or right skips, in upto three passes (in LuaMetaTeX we can specify a sequence of passes so that can be seen as a 'many years later' follow up then).

Of course there are some math definitions, watch the way characters are defined: by value. The `\limitsswitch` is what later became `\limits`.

```
\def\log{\mathop{\char'154\char'157\char'147}\limitswitch}
\def\lg{\mathop{\char'154\char'147}\limitswitch}
\def\ln{\mathop{\char'154\char'156}\limitswitch}
\def\lim{\mathop{\char'154\char'151\char'155}}
\def\limsup{\mathop{\char'154\char'151\char'155
    \,\char'163\char'165\char'160}}
\def\liminf{\mathop{\char'154\char'151\char'155
    \,\char'151\char'156\char'146}}
\def\sin{\mathop{\char'163\char'151\char'156}\limitswitch}
\def\cos{\mathop{\char'143\char'157\char'163}\limitswitch}
\def\tan{\mathop{\char'164\char-141\char'156}\limitswitch}
\def\cot{\mathop{\char'143\char'157\char'164}\limitswitch}
\def\sec{\mathop{\char'163\char'145\char'143}\limitswitch}
\def\csc{\mathop{\char'143\char'163\char'143}\limitswitch}
\def\max{\mathop{\char'155\char'141\char'170}}
\def\min{\mathop{\char'155\char'151\char'156}}
\def\sup{\mathop{\char'163\char'165\char'160}}
\def\inf{\mathop{\char'151\char'156\char'146}}
\def\det<\mathop{\char'144\char'145\char'164}}
\def\exp{\mathop{\char'145\char'170\char'160}\limitswitch}
\def\Pr{\mathop{\char'120\char'162}}
\def\gcd{\mathop{\char'147\char'143\char'144}}
\def\choose{\comb()}
\def\leftset{\mathopen{\{\,}}
\def\rightset{\mathclose{\,\}}}
\def\modop{\<\,\mathbin{\char'155\char'157\char'144>\penalty900\<\,}
\def\mod#1{\penalty0\;(\char'155\char'157\char'144\,\,#1)}
\def\eqv{\mathrel\char'421 }
\def\neqv{\mathrel{\not\eqv}}
```

I think that talking in octal was quite popular in those days, just like hexadecimal is in our times. In later versions of basic we see this:

```
\def\sin{\mathop{\char s\char i\char n}\limitswitch}
```

In plain we eventually got to this:[14]

```
\def\sin{\mathop{\rm sin}\nolimits}
```

This definitions calls for `\quad` which is not defined, so it's still a primitive.

[14] In LuaMetaTEX we have more built-in classes and one can add even more, so there we use a dedicated function class so that we can tune spacing better. So there is a different way to define functions, also because we can set more properties.

```
\def\qquad{\quad\quad}
```

The `\≥` is likely some spacing directive and uses a symbol that even today is not on keyboards:

```
\def\ldots{{.\≥.\≥.}}
\def\cdots{{\char'401\≥\char'401\≥\char'401}}
\def\ldotss{{.\≥.\≥.\≥}}
\def\cdotss{\cdots\≥}
\def\ldotsm{{\≥.\≥.\≥.\≥}}
\def\vdots{\vbox{\baselineskip 4pt\vskip 6pt
    \hbox{.}\hbox{.}\hbox{.}}}
```

These definitions, related to alignments, are not that different from what they later became. Again we have rather large plus and minus values. The short `\dispstyle` later became more verbose, so it is not a typo. Does anyone today use these pile commands, a two-line helper (meant for splitting a long math line in display mode), chops and/or sposes? The `\spose` definitely is invalid on later TEX engines as they have put a limit on the size of dimensions.

The `\!` is a hard coded primitive that basically ends a line without adding a space. To quote the manual:

> “First, a (carriage-return) always counts as a space, even when it follows a hyphen. If you want to end a line with a (carriage-return) but no space, you can do this by typing the control sequence “`\!`” just before the (carriage-return).”

In various places the complications of spacing in the input is discussed and even today this is something to pay attention to. The `\!` is one of the primitives, the future `\ignorespaces`.

```
\def\eqalign#1{\vcenter{\halign{\hfill$\dispstyle{##}$\!
      ⊗$\dispstyle{\null##}$\hfill\cr#1}}}
\def\eqalignno#1{\vbox{\tabskip0pt plus1000pt minus1000pt
    \halign to size{\hfill$\dispstyle{##}$\tabskip 0pt
      ⊗$\dispstyle{\null##}$\hfill
      \tabskip0pt plus1000pt minus1000pt
      ⊗$\hfill##$\tabskip 0pt\cr#1}}}
\def\cpile#1{\vcenter{\halign{$\hfill##\hfill$\cr#1}}}
\def\lpile#1{\vcenter{\halign{$##\hfill$\cr#1}}}
\def\rpile#1{\vcenter{\halign{$\hfill##$\cr#1}}}
\def\null{\hbox{}}
\def\twoline#1#2#3{\halign{\hbox to size{##}\cr$\quad\dispstyle
    {#1}$\hfill\cr\noalign{\penalty1000\vskip#2}
    \hfill$\dispstyle{#3}\quad$\cr}}
\def\chop to#1pt#2{\hbox{\lower#1pt\null\vbox{\hbox{\lower99pt
    \hbox{\raise99pt\hbox{$\dispstyle{#2}$}}}\vskip-99pt}}}
```

```
\def\spose#1{\hbox to 0pt{#1\hskip0pt minus10000000pt}}
```

We now arrive at fonts. Given memory constraints the number of fonts that can be used is small. The names start with `\:` and the reference is a character. These are the precursors of computer modern with predefined sizes, no scaling.

```
\:@←cmathx
\:a←mr10 \:d←mr7 \:f←mr5
\:g←mi10 \:j←mi7 \:l←mi5
\:n←ms10
\:q←mb10
\:u←msy10 \:x←msy7 \:z←msy5
\:?←mti10
```

In later versions of the basic file (dating from after the book) we see that a `c` has been being prepended to the font names. The 10 → 7 → 5 steps didn't change over time. These are familiar shortcuts and we kept them in ConTEXt, although with a different macro body:

```
\def\rm{\:a} \def\sl{\:n} \def\bf{\:q} \def\it{\:?}
```

There are a few names here that were replaced: `\topbaseline` became `\topskip` and the three display skips became four more verbose ones.

```
\parindent 20pt \maxdepth 2pt \topbaseline 10pt
\parskip 0pt plus 1 pt \baselineskip 12pt \lineskip 1pt
\dispskip 12pt plus 3pt minus 9pt
\dispaskip 0pt plus 3pt \dispbskip 7pt plus 3pt minus 4pt
```

Here we see a font switch to an extensible font:

```
\def\biglp{\mathopen{\vcenter{\hbox{\:@\char'0}}}}
\def\bigrp{\mathclose{\vcenter{\hbox{\:@\char'1}}}}
\def\bigglp{\mathopen{\vcenter{\hbox{\:@\char'22}}}}
\def\biggrp{\mathclose{\vcenter{\hbox{\:@\char'23}}}}
\def\bigggl p{\mathopen{\vcenter<\hbox<\:@\char'40}}}}
\def\bigggrp{\mathclose{\vcenter{\hbox{\:@\char'41}}}}
```

Without reading documentation one can guess what this does: we set the three families:

```
\mathrm adf \mathit gjl \mathsy uxz \mathex @
```

Eventually TEX got the `\textfont` (and script) definition primitives which gave macro packages the opportunity to use these as macro names.

We're nearly done, here is the basic output routine. It assumes a meaningful `\page`. A page gets the top inserts, followed by the skip, then the content, then the other skip and finally the bottom inserts. Of course that became different in TEX86 when we got a more generalized approach to inserts.

The baseline skip is quite extreme here, as if it's meant for proofing. Also watch the page counter: the `\count` command is not what we're accustomed to: it expands, like `\the\count` and `\number\count` do today. Some videos refer to `\the` so when you watch them and hear a reference to 'sail' or an older TEX, we're talking this one, otherwise a successor. The `\setcount` does the assignment.

```
\output{\baselineskip20pt\page\ctrline'{\:a\count0}\advcount0}

\setcount0 1
```

Well, why not end the basic file with:

```
\rm
\null\vskip-12pt % allow glue at top of first page
```

This format is small and for a real book more is needed. The assumption was that there is only a very basic setup with then an additional book specific style. One could also copy basic and use a patched version. But because those involved were programmers, or at least aware of what computers could mean for them, it is no surprise that larger macro packages started showing up. When looking in more detail at what is provided, in the next sections we will see that some parameters that went away, others came along, and a couple of concepts changed. That made writing more general purpose macro packages easier.

4 Sail

The first version was written in SAIL, and when we fetch the sources from Don Knuths website we can identify what become primitives and how they are grouped. Contrary to the next sections, where we can parse the `.web` files for `primitive` and use that as trigger for (runtime) filtering, here we show some more manually filtered sections. If you're not familiar with how TEX is coded, you have to take what is said below for granted and just try to get the idea. Seeing the variables and definitions involved can give an impression of how it all relates to for instance primitives and functionality that you run into.

In TEX we need to distinguish primitives and the way that is done is by relating them with two numbers: `cmd` and `chr`. The first 12 commands are single character commands. For instance a backslash `chr` is an escape `cmd`. When `\something` is encountered in the source, normally the backslash will trigger reading letters and when done lookup the name assembled from them. Unless one made an error, that name (it could be a primitive or user defined macro) is looked up and resolved to a `cmd` and `chr` where the latter can be a number representing a value or a pointer to a token list. We will not go into details here.

An interesting revelation in the presentations is that when you start up TEX in interaction mode, and no escape character has been setup, the first character entered will be defined as such, something that of course gets unnoticed when you start with a command, because then the backslash becomes the escape character.

```
escape      0 # escape delimiter (\ in TEX manual);
lbrace      1 # begin block symbol ({);
rbrace      2 # end block symbol (});
mathbr      3 # math break ($);
tabmrk      4 # tab mark ();
carret      5 # carriage return and comment mark (%);
comment carret is also used as the command code for \cr;
macprm      6 # macro parameter (#);
supmrk      7 # superscript (^);
submrk      8 # subscript (↓);
ignore      9 # chars to ignore;
spacer     10 # chars treated as blank space;
letter     11 # chars treated as letters;
otherchar  12 # none of the above character types;
```

These were basically what later became catcode categories, here we have 12, a modern TEX has 16. It might be a bit out of place, but here is how ConTEXt defines these constants in MkIX using LuaMetaTEX:[15]

```
\permanent\immutable\integerdef\escapecatcode        0
\permanent\immutable\integerdef\begingroupcatcode    1
\permanent\immutable\integerdef\endgroupcatcode      2
\permanent\immutable\integerdef\mathshiftcatcode     3
\permanent\immutable\integerdef\alignmentcatcode     4
\permanent\immutable\integerdef\endoflinecatcode     5
\permanent\immutable\integerdef\parametercatcode     6
\permanent\immutable\integerdef\superscriptcatcode   7
\permanent\immutable\integerdef\subscriptcatcode     8
\permanent\immutable\integerdef\ignorecatcode        9
\permanent\immutable\integerdef\spacecatcode        10
\permanent\immutable\integerdef\lettercatcode       11
\permanent\immutable\integerdef\othercatcode        12
\permanent\immutable\integerdef\activecatcode       13
\permanent\immutable\integerdef\commentcatcode      14
\permanent\immutable\integerdef\invalidcatcode      15
```

Watch the change in terminology: *block symbol* is now *group*, *math break* became *math shift*, and some other minor renames. But we also got active characters, a configurable comment and, just to complete the hexadecimal range, an 'invalid' category code.

[15] For the record: the prefixes freezes these definitions (permanent can be overruled by changing an overload protection flag, the immutable property inhibits even that.)

We will go over the rest of the range and comment when we see something interesting and in most cases the comment explains what we encounter.

```
parend      13 # end of paragraph;
match       14 # macro parameter matching;
outpar  ignore # output a macro parameter;
endv        15 # end of vlist in halign or valign template;
call        16 # call a user-defined macro;
```

In the presentations Don frequently refers to extensions. In current TEX examples of extensions are opening and closing files, reading and writing to them, and specials. They are nowadays called whatsits but were always part of the game and lucky us: they made TEX survive decades to come because it could adapt. For instance pdfTEX uses extensions a lot, for instance for hyperlinks.[16]

```
xt  17 # extensions to basic TEX (\x)
```

Here we see dedicated assignment classes for glue and (indeed) reals. In later versions we talk about (16.16) dimensions and only use 'real' for a field in the box that stored the to be applied glue factor: the only officially system specific difference across platforms.

```
assignglue  18 # user-defined glue
font        19 # user-defined current font
assignreal  20 # user-defined length
```

We see that the output routine is an independent command:

```
def     21 # macro definition (\def,\gdef)
output  22 # output routine definition (\output)
innput  23 # required input file (\input)
```

There are all kind of parameters but less than we will later get. They don't have primitive names, and in the basic format we saw `\jpar` and `\trace` being defined as shortcuts for numeric references:

```
setpar  24 # set TEX control parameter (\trace,\jpar)
```

In the presentations we also see some tracing that later changed, and debugging is mentioned too, although it's seen as an emergency measure.

```
stop  25 # end of input (\end)
ddt   26 # emergency debugging (\ddt)
```

[16] The original extensions didn't really assume that for instance whatsit nodes have dimensions, because if that were the case one has to also patch the various places where dimensions kick in, like in the par builder. In pdfTEX (and therefore LuaTEX) some whatsits do have dimensions so there indeed some mechanisms have to be aware of that. In LuaMetaTEX we treat whatsits as invisible because, after all, one can wrap into a box to communicate dimensions.

This `chartype` later became `catcode` but what is this `ascii`?

```
ascii   27 # code for possibly untypeable character (\char)
chcode  28 # change chartype table (\chcode)
```

The `\char` primitives behave different in text and math mode and later we got a split between `\char` and `\mathchar`. The concept of families is there:

```
fntfam  29 # declare font family (\mathrm,etc.)
```

Here we see a difference with later versions. We have an explicit set and advance primitives and a serializer. In modern TEX `\count` got a different meaning (setter and getter in one) and `\advance` got shared.

```
setcount  30 # set current page number (\setcount)
advcount  31 # increase current page number (\advcount)
count     32 # insert current page number (\count)
```

We're sure that there have been good reasons to change `\ifeven` into `\ifodd` even when that introduced an incompatibility, but I guess that given the pace of development, that was the least of ones worries. The name 'delimiter' for `\else` makes me smile. Watch how we have two conditional command classes. There are no `\iftrue` and `\iffalse` so here is a state check example given:

```
\def\firsttime{T}
... \if T \firsttime{\gdef\firstime{F}}\else{...}\fi ...
```

The braces are mandate and also prevent look-ahead expansion as they end the test. I couldn't find `\ifT`, sorry.

```
ifeven    33 # conditional on count even (\ifeven)
ifT       34 # conditional on character T (\ifT)
elsecode  35 # delimiter for conditionals (\else)
```

We still have a split between setter and getter but instead of `\save` we now use `\setbox`; we also can make copies. It is worth noticing that we have a `\shipout` called `\page` and dedicates justification commands (`\hbox` and `\vbox`):

```
box    36 # saved box (\box,\page) or justification (\hjust,\vjust)
hmove  37 # horizontal motion of box (\moveleft,\moveright)
vmove  38 # vertical motion of box (\raise,\lower)
save   39 # save a box (\save)
```

So these were always available:

```
leaders  40 # define leaders (\leaders)
```

These are also familiar:

```
halign   41 # horizontal table alignment (\halign)
```

```
valign   42 # vertical table alignment (\valign)
noalign  43 # insertion into halign or valign (\noalign)
vskip    44 # vertical glue (\vskip,\vfill)
hskip    45 # horizontal glue (\hskip,\hfill)
vrule    46 # vertical rule (\vrule)
hrule    47 # horizontal rule (\hrule)
```

However, inserts were hard coded. I think that in these days graphics were still glued in so basically only footnotes were in demand.

```
topbotins  48 # inserted vlist (\topinsert or \botinsert)
```

In the book style example in the basic TEXbook we see this definition:

```
\def\footnote#1#2{\botinsert{\hrule width5pc \vskip3pt
  \baselineskip9pt\hbox par size{\eightpoint#1#2}}}
```

So we use the bottom insert, which eventually will be separated by a `\botskip` and it will have a rule on top (question: what will happen with multiple notes? We didn't mention specifying boxes yet but here is the possible syntax:

```
\hbox {} \hbox to size {} \hbox to <dimen> {} \hbox expand <dimen> {}
\hbox par size {} \hbox par <dimen> {}
\vbox {} \vbox to size {} \vbox to <dimen> {} \vbox expand <dimen> {}
```

This `par` is used in the footnote definition, we basically get a `\vbox` disguised as `\hbox`. Later these box definitions became:

```
\hbox {} \hbox to <dimen> {} \hbox spread <dimen> {}
\vbox {} \vbox to <dimen> {} \vbox spread <dimen> {}
```

At that point we also got `\vsplit`, multiple inserts, inserts that could be split over pages, properties like skips bound to inserts, etc.

The, also migrating, marks look familiar and two command categories are used. Here `\firstmark` is not mentioned:

```
topbotmark  49 # insert mark (\topmark,\botmark)
mark        50 # define a mark (\mark)
```

For sure we have:

```
penalty  51 # specify badness of break (\penalty)
```

And:

```
noindent  52 # begin nonindented paragraph (\noindent)
```

Eh ... we now have triggers like `\outputpenalty` and the concept of push back in the output routine. The old TEX engine has an explicit `\eject` that can be invoked in horizontal mode (middle of a paragraph) or vertical mode. In new TEX this became a plain macro. Another interesting primitive is `\page` that inserts the collected content, box number 255 in new TEX.

```
eject  53 # eject page here (\eject)
```

The next command deals with the hyphen related primitives. The `\*` primitive is for math where it will insert a × with a possible line break after it. This is an interesting observation: this native feature was removed in the final version of TEX, but at some point in LuaMetaTEX discretionaries came back, for repeated operators (at the end of a line and the start of a next line) as well as specific (three part) breakpoints.

```
discr  54 # discretionary hyphen (\-,\*)
```

Interesting, as later we only had an accent placement, seldom seen in the running text, possibly used in macros, if at all, because at some point TEX went eight bit.

```
accent     55 # attach accent to character (\+)
newaccent  56 # define nonstandard accent (\accent)
```

Only one side of the (centered) equation is covered:

```
eqno  57 # insert equation number (\eqno)
```

Hm, apart from that space, we don't have it like this any longer:

```
mathonly       58 # character or token allowed in mathmode only
exspace        59 # explicit space (\ )
nonmathletter  60 # letter except in mmode
```

In modern TEX these became fences:

```
leftright  61 # variable delimiter (\left, \right)
              # comment there is no code 62 today
```

These are atoms (a nucleus with a superscript or subscript):

```
mathinput  63 # component of math formula (\mathop,\mathbin, etc.)
```

We still have this modifier, a primitive that adapts the last added math atom (if it makes sense):[17]

```
limsw  64 # modify limit conventions (\limitswitch)
```

Math again, but the `\comb` is gone. We have the `\..withdelim` variants now:

```
above  65 # numerator-denominator separator(\above,\atop,\over,\comb)
```

[17] The node list is forward linked only in traditional TEX so this only works on what is called the tail of the current (math) list. In LuaTEX we are dual linked so in principle one can look further back but there is no real reason to do this. It is one of the few cases where the engine looks back anyway.

Watch how for instance the text style is referred as:

```
mathstyle  66 # style or space specification (\dispstyle,\,,etc.)
```

We still have that, it's a typical TEX concept, not present in modern fonts:

```
italcorr  67 # italic correction (\/)
```

Yes, another adjust. In traditional TEX this is a math mode command, but in LuaMetaTEX we made it valid in text mode too.

```
vcenter  68 # vjust centered on axis (\vcenter)
```

This next command is peculiar because it is not a parameter as the other integer ones but it has it's own command category. There is in this list of commands no mentioning of more advanced paragraph properties, not even `\everypar` because that will show up in TEX82.

```
hangindent  69 # specifies hanging indentation (\hangindent)
```

So where is `\hangafter`? It actually is the reason why we have a dedicated command class because a special scanner is needed:

```
\hangafter 20pt for   1
\hangafter 20pt after 2
```

A negative dimension hangs right, the `for` and `after` options determine the number of lines.

Next we see some of the parameters mapped to memory. We challenge you to translate these to modern TEX's parameters:

```
define tracing = eqtb[hashsize+268] # controls diagnostics
define jpar     = eqtb[hashsize+269] # controls justification
define hpen     = eqtb[hashsize+270] # hyphenation penalty
define penpen  = eqtb[hashsize+271] # penultimate penalty
define wpen     = eqtb[hashsize+272] # widow-line penalty
define bpen     = eqtb[hashsize+273] # broken-line penalty
define mbpen    = eqtb[hashsize+274] # binary-op-break penalty
define mrpen    = eqtb[hashsize+275] # relation-break penalty
define ragged  = eqtb[hashsize+276] # raggedness
```

The par builder can be configured a bit but some criteria are hard coded. Watch how we don't have a club penalty here. The widow penalty is used for that as well as display math.

Because skips are more than just a single integer, they are actually nodes, so here we need to refer to node memory:

```
define lineskiploc      = locs[0]
define baselineskiploc = locs[1]
define parskiploc      = locs[2]
define dispskiploc     = locs[3]
define topskiploc      = locs[4]
define botskiploc      = locs[5]
define tabskiploc      = locs[6]
define dispaskiploc    = locs[7]
define dispbskiploc    = locs[8]
```

Just to show you that the math concepts are there:

```
define dispstyle         = 0
define textstyle         = 1
define scriptstyle       = 2
define scriptscriptstyle = 3
```

Nowadays we still have these noads, intermediate nodes, here are the atoms:

```
define boxnoad   = 0
define opnoad    = 1
define binnoad   = 2
define relnoad   = 3
define opennoad  = 4
define closenoad = 5
define punctnoad = 6
```

Do you recognize the fraction?

```
define sqrtnoad   =  7
define overnoad   =  8
define undernoad  =  9
define accentnoad = 10
define abovenoad  = 11
```

The square root has no degree but other roots have. Eventually Unicode engines using OpenType math fonts got a variant that handles the degree option but here it still has to be done manually via a macro that overlays the degree.

Nowadays we have a fence noad to which e-TEX added a middle variant (subtype):

```
define leftnoad  = 12
define rightnoad = 13
```

The simple noad and style are there too but we can assume that the way the noads map onto memory changed a bit over time (I didn't check it).

```
define nodenoad  = 14
define stylenoad = 15
```

These look more hard coded than in the final version of TEX, where we can define and set glue and muglue registers.

```
define thinspace      =  8
define thickspace     =  9
define quadspace      = 10
define negthinspace   = 11
define negthickspace  = 12
define negopspace     = 13
define userspace      = 14
define nospace        =  6
define opspace        =  7
define thspace        = 15
define negthspace     = 16
```

This was just an impression and likely one that has errors. We don't have (access to) many details but even if we're wrong in aspects it is clear using this SAIL prototype gave plenty of input to the project so when the rewrite to Pascal took place, mixed with web documentation, these concepts evolved. Let's now move to the next iteration but before we do that we zoom in on the transition between the prototype and what became the final version.

5 The intermezzo

The book has an appendix `<X>` titled "Recent extensions to TEX" that describes additions to the engine done right before the mentioned book was published. These were likely driven by usage, macro packages showing up, and users demanding more from macros and manipulations. In the videos Knuth makes clear that TEX82 will converge to a stable version. He announces the TEX book, the MetaFont book and a book about the fonts. He also mentions that, while TEX underwent rather fundamental changes and upgrades, MetaFont will be less different. But when you look at the MetaFont section in the book, it's hard to see the similarities. Of course I look at it from the perspective of MetaPost being used for graphics other that glyph shapes, but if we look at the bitmap generation bits, the language, variables, path construction, macros ... one really has to rewrite the older code, but most ideas behind it remain. References to built-in variables, hard coded pens, control point features, subroutines and calls, it looks somewhat less meta than today.

But when it comes to TEX, in a way you can recognize the upcoming TEX82 in that because when extensions like that get added, later changes are natural. So, let's reflect a bit on what we can read about it. Take for instance how we stepwise came to `\advance`. There are ten counters and you can do the following

```
\setcount 4 = 10  \advcount 4  \advcount 4  \count 4
```

And because `\count` expands you get a typeset 12 here because `\advcount` increments by one. In the appendix an extension is introduced:

```
\setcount 4 = 10  \advcount 4 by 2  \count 4
```

This comes in handy when we need to increment by 50 which otherwise would demand a recursive loop. However, in TEX82 the `\advcount` got companions like `\advdimen` and code was shared between, made possible by (internally) assigning a property that signals what is dealt with. But, soon an incompatible change was introduced:

```
\count 4 = 10  \advance\count 4 by 2  \the\count 4
```

From that time on each register uses the same `\advance` and the now unexpandable register references have to use `\the` as serializer. This transition was rather natural and made it easier to program more complex solutions.

We didn't mention yet that the conditional primitives were also syntactically different. It started with:

```
do \ifeven \count4 {this}\else{that}
```

so mandate braces, a mandate (redundant?) `\else` and no `\fi`. That later became:

```
do   \ifodd \count4 this\else that\fi % if it has to be clear
do th\ifodd \count4   is\else   at\fi % if you want to impress
```

When the appendix introduces the `\ifpos` test, the next example is given:

```
\def\neg#1{\setcount#1-\count#1}
\def\ifzero#1#2\else#3{\ifpos#1{#3}\else{\neg#1
    \ifpos#1{\neg#1 #3}\else{\neg#1 #2}}}
```

Apart from `\neg` later being a math related command, this looks like a lot of work for a test on a number being positive. It is not fully expandable (but one can wonder if that mattered much in those early days), it negates the counter because it can only test for positive and then has to convert the counter back to its original value. How much more convenient will this become:

```
\ifnum#1>0 ... \else ... \fi % replaces \ifpos
\ifnum#1=0 ... \else ... \fi % replaces \ifzero
\ifcase#1  ... \else ... \fi % alternative
```

It is revealing to read about the development of (conditionals in) early TEX, in particular regarding recent choices I made in ConTEXt when extending the repertoire. As a LuaMetaTEX extension `\orelse` is one of my favorites because it makes for less nesting of conditionals. These types of simple constructions are the core of the language and of immediate use as soon as a user needs to go a bit low level. As part of the soul of the TEX language we are happy to expose our users to them.

Let's just mention that `\ifvmode`, `\ifhmode`, `\ifmmode` were added as well as `\lineskiplimit` and `\mathsurround` and more would follow.

Yet another extension holds some promise for the future TEX82: a user dimensions (there are no dimen registers yet). The `\varunit` variable can be set to some value and the `vu` dimension can then use that. In order to be able to work with the dimensions of boxes there were also `wd`, `ht` and `dp` units each to be followed by some box number.

The fact that one could say `12.3vu` or `2wd3` actually makes me feel less guilty for adding the `dk` unit (as a demonstration of extension), the `ts` and `es` units (in order to celebrate the fact that teenagers are willing to accompany parents to TEX conferences) and `eu` for what actually is a bit like the var unit, in this case a multiplier for `es`, and therefore European thumb based (to counter the inch). But aside from these semi-serious extensions, in LuaMetaTEX we also have installable units, so we could do this if we like to:

```
\newdimen\varunit \varunit 10pt
\associateunit\varunit{(`v-`a)*26+(`u-`a)}
\varunit 10pt \dimen0=10vu \the\dimen0
```

It happens that we already have `uu` as 'user unit' but if I'd known that there had been a `vu` . . . who knows. We could discuss it at an upcoming ConTEXt meeting.

But there is another guilt diminishing feature: this box dimension unit. It demands not just a keyword check but also picking up a number. Later we got `\wd` etc. that are a but more natural in accessing these properties, but the fact that we have a kind of a subexpression makes that adding plenty more such specific burdens on the parser seems quite okay. So where in the above snippet you will notice that we scan an expression, you can be sure that more is possible between these braces and it's not that unnatural to TEX after all.

In today's TEX there are ways to set up characters to be of a certain math class. In the intermezzo between SAIL TEX and TEX82 that could be done via the what was then the catcode setter: `\chcode`, using values beyond the catcode range. Later math got its own setters for that. At the same time the `\char` primitive got the option to pass a letter as alternative for a number. That was later changed, because we got `` ` `` as numeric prefix to characters, like `` `a ``. I suppose some users had to change their format files and styles a few times. It might also have been one of the reasons why later TEX got frozen, not only because of the rendering but also because of setting it up to do just that.[18]

The somewhat weird, large `plus` and `minus` values in some basic macros are so large that later they became invalid. In the intermezzo period we see `\hfil`, `\hfilneg`, `\hss` and their vertical counterparts replace them: no more `plus 100000pt` and such are needed. The argument is actually that it saves memory, which is due to the fact that instead of variables (of glue nodes) one has references to fixed nodes. Of course one can wonder if in practice it really saved much memory. A similar optimization, namely sharing space glue by referencing glue nodes fits in here too but isn't mentioned.

[18] There is this active character class in math mode so in some sense we still have a special relationship between character codes and math classes.

The introduction of `\xdef` was needed because there was no `\global` prefix yet. Introducing the `\lowercase` and `\uppercase` primitives made sense but later usage demonstrated that as they operate on their argument and not the node list it is not always trivial to use them in situations where expansion can interfere.

A final remark in the appendix mentions that control sequences are from then on remembered in full, that is: the seven character limit is gone! It probably also made for less code. That said, we now move on to the first TEX82 implementation that figures in the presentations.

6 Old TEX82

When we talk about 'old TEX', we mean the early '82 version. The sources can be downloaded from Don Knuths website but they are just that: sources. There is nothing you can run so your mind is the computer. For that reason we will only look at the commands, and comment a bit on those. We are of course talking of a working program but when you compare it with what we will call 'new TEX82' you will notice some differences that indicate that the latter is more mature and that decisions were made that really made sense for the program to succeed and prosper. We start with the primitives and present the list as filtered from the source(s). Bear in mind that we could have missed something. The idea is that you go over the list and try to identify commands that are gone or changed. The first column mentions the primitive, the second column the so called command category it fits in. You can think a bit of the primitives being interpreted by a virtual machine that interprets an operator and operand.

It is a long list but we like you to try to locate the primitives that are not in today's TEX, that have been renamed, or have become different.

\lineskip	assign_glue	line_skip_code
\baselineskip	assign_glue	baseline_skip_code
\parskip	assign_glue	par_skip_code
\dispskip	assign_glue	disp_skip_code
\dispaskip	assign_glue	disp_a_skip_code
\dispbskip	assign_glue	disp_b_skip_code
\leftskip	assign_glue	left_skip_code
\rightskip	assign_glue	right_skip_code
\topskip	assign_glue	top_skip_code
\splittopskip	assign_glue	split_top_skip_code
\tabskip	assign_glue	tab_skip_code
\spaceskip	assign_glue	space_skip_code
\xspaceskip	assign_glue	xspace_skip_code
\parfillskip	assign_glue	par_fill_skip_code
\thinmskip	assign_glue	thin_mskip_code
\medmskip	assign_glue	med_mskip_code
\thickmskip	assign_glue	thick_mskip_code
\output	assign_toks	output_routine_loc
\everypar	assign_toks	every_par_loc
\pretolerance	assign_int	pretolerance_code
\tolerance	assign_int	tolerance_code

\linepenalty	assign_int	line_penalty_code
\hyphenpenalty	assign_int	hyphen_penalty_code
\exhyphenpenalty	assign_int	ex_hyphen_penalty_code
\widowpenalty	assign_int	widow_penalty_code
\displaywidowpenalty	assign_int	display_widow_penalty_code
\brokenpenalty	assign_int	broken_penalty_code
\binoppenalty	assign_int	bin_op_penalty_code
\relpenalty	assign_int	rel_penalty_code
\predisplaypenalty	assign_int	pre_display_penalty_code
\postdisplaypenalty	assign_int	post_display_penalty_code
\interlinepenalty	assign_int	inter_line_penalty_code
\doublehyphendemerits	assign_int	double_hyphen_demerits_code
\finalhyphendemerits	assign_int	final_hyphen_demerits_code
\adjdemerits	assign_int	adj_demerits_code
\mag	assign_int	mag_code
\delimiterfactor	assign_int	delimiter_factor_code
\looseness	assign_int	looseness_code
\time	assign_int	time_code
\day	assign_int	day_code
\month	assign_int	month_code
\year	assign_int	year_code
\showboxbreadth	assign_int	show_box_breadth_code
\showboxdepth	assign_int	show_box_depth_code
\hbadness	assign_int	hbadness_code
\vbadness	assign_int	vbadness_code
\pause	assign_int	pause_code
\tracingonline	assign_int	tracing_online_code
\tracingmacros	assign_int	tracing_macros_code
\tracingstats	assign_int	tracing_stats_code
\tracingoutput	assign_int	tracing_output_code
\tracinglostchars	assign_int	tracing_lost_chars_code
\tracingcommands	assign_int	tracing_commands_code
\uchyph	assign_int	uc_hyph_code
\outputpenalty	assign_int	output_penalty_code
\hangafter	assign_int	hang_after_code
\parindent	assign_dimen	par_indent_code
\mathsurround	assign_dimen	math_surround_code
\varunit	assign_dimen	var_unit_code
\lineskiplimit	assign_dimen	line_skip_limit_code
\hsize	assign_dimen	hsize_code
\vsize	assign_dimen	vsize_code
\maxdepth	assign_dimen	max_depth_code
\splitmaxdepth	assign_dimen	split_max_depth_code
\hfuzz	assign_dimen	hfuzz_code
\vfuzz	assign_dimen	vfuzz_code
\delimiterlimit	assign_dimen	delimiter_limit_code
\nulldelimiterspace	assign_dimen	null_delimiter_space_code
\scriptspace	assign_dimen	script_space_code
\predisplaysize	assign_dimen	pre_display_size_code
\displaywidth	assign_dimen	display_width_code
\displayindent	assign_dimen	display_indent_code
\overfullrule	assign_dimen	overfull_rule_code
\relax	relax	0
\let	let	0

\char	char_num	0
\mathchar	math_char_num	0
\mark	mark	0
\input	input	0
\penalty	break_penalty	0
\font	def_font	0
\:	set_font	0
\fam	set_family	0
\number	number	0
\setbox	set_box	0
\unbox	unbox	0
\unskip	unskip	0
\lastskip	last_skip	0
\halign	halign	0
\valign	valign	0
\noalign	no_align	0
\vrule	vrule	0
\hrule	hrule	0
\insert	insert	0
\vadjust	vadjust	0
\ignorespace	ignore_space	0
\parshape	set_shape	0
\/	ital_corr	0
\accent	accent	0
\mathaccent	math_accent	0
\texinfo	assign_tex_info	0
\delimiter	delim_num	0
\limitswitch	limit_switch	0
\nonscript	non_script	0
\vcenter	vcenter	0
\case	case_branch	0
\else	else_code	0
\omit	omit	0
\groupbegin	group_begin	0
\groupend	group_end	0
\	ex_space	0
\radical	radical	0
\par	par_end	0
\topmark	top_bot_mark	top_mark_code
\firstmark	top_bot_mark	first_mark_code
\botmark	top_bot_mark	bot_mark_code
\splitfirstmark	top_bot_mark	split_first_mark_code
\splitbotmark	top_bot_mark	split_bot_mark_code
\count	register	int_val
\dimen	register	dimen_val
\skip	register	glue_val
\hangindent	hang_indent	hanging_indent_code
\the	the	0
\minus	the	1
\spacefactor	set_aux	hmode
\prevdepth	set_aux	vmode
\span	tab_mark	span_code
\cr	car_ret	cr_code
\end	stop	0

\dump	stop	1
\hskip	hskip	skip_code
\hfil	hskip	fil_code
\hfill	hskip	fill_code
\hss	hskip	ss_code
\hfilneg	hskip	fil_neg_code
\vskip	vskip	skip_code
\vfil	vskip	fil_code
\vfill	vskip	fill_code
\vss	vskip	ss_code
\vfilneg	vskip	fil_neg_code
\mskip	mskip	mskip_code
\kern	kern	normal
\mkern	mkern	mu_glue
\moveleft	hmove	1
\moveright	hmove	0
\raise	vmove	1
\lower	vmove	0
\box	make_box	box_code
\copy	make_box	copy_code
\lastbox	make_box	last_box_code
\vsplit	make_box	vsplit_code
\vtop	make_box	vtop_code
\vbox	make_box	vtop_code+vmode
\hbox	make_box	vtop_code+hmode
\shipout	leader_ship	a_leaders-1
\leaders	leader_ship	a_leaders
\cleaders	leader_ship	c_leaders
\xleaders	leader_ship	x_leaders
\indent	start_par	1
\noindent	start_par	0
\-	discretionary	1
\discretionary	discretionary	0
\eqno	eq_no	0
\leqno	eq_no	1
\mathord	math_comp	ord_noad
\mathop	math_comp	op_noad
\mathbin	math_comp	bin_noad
\mathrel	math_comp	rel_noad
\mathopen	math_comp	open_noad
\mathclose	math_comp	close_noad
\mathpunct	math_comp	punct_noad
\underline	math_comp	under_noad
\overline	math_comp	over_noad
\displaystyle	math_style	display_style
\textstyle	math_style	text_style
\scriptstyle	math_style	script_style
\scriptscriptstyle	math_style	script_script_style
\above	above	above_code
\over	above	over_code
\atop	above	atop_code
\xabovex	above	xx_code+above_code
\xoverx	above	xx_code+over_code
\xatopx	above	xx_code+atop_code

\left	left_right	left_noad
\right	left_right	right_noad
\if	if_test	if_char_code
\ifnum	if_test	if_int_code
\ifdim	if_test	if_dimen_code
\ifeven	if_test	if_even_code
\ifvmode	if_test	if_vmode_code
\ifhmode	if_test	if_hmode_code
\ifmmode	if_test	if_mmode_code
\ifinner	if_test	if_inner_code
\ifabsent	if_test	if_absent_code
\ifx	if_test	ifx_code
\long	prefix	1
\outer	prefix	2
\global	prefix	4
\def	def	0
\gdef	def	1
\edef	def	2
\xdef	def	3
\chcode	def_code	ch_code_base
\mathcode	def_code	math_code_base
\lccode	def_code	lc_code_base
\uccode	def_code	uc_code_base
\sfcode	def_code	sf_code_base
\delcode	def_code	del_code_base
\textfont	def_family	math_font_base
\scriptfont	def_family	math_font_base+script_size
\scriptscriptfont	def_family	math_font_base+script_script_size
\setcount	set_register	int_val
\setdimen	set_register	dimen_val
\setskip	set_register	glue_val
\advcount	adv_register	int_val
\advdimen	adv_register	dimen_val
\advskip	adv_register	glue_val
\multcount	mult_register	int_val
\multdimen	mult_register	dimen_val
\multskip	mult_register	glue_val
\divcount	div_register	int_val
\divdimen	div_register	dimen_val
\divskip	div_register	glue_val
\hyphenation	hyph_data	0
\patterns	hyph_data	1
\batchmode	set_interaction	batch_mode
\nonstopmode	set_interaction	nonstop_mode
\scrollmode	set_interaction	scroll_mode
\errorstopmode	set_interaction	error_stop_mode
\message	message	0
\errmessage	message	1
\lowercase	case_shift	lc_code_base
\uppercase	case_shift	uc_code_base
\show	xray	show_code
\showbox	xray	show_box_code
\showthe	xray	show_the_code
\showlists	xray	show_lists

\open	extension	open_node
\send	extension	send_node
\close	extension	close_node
\xsend	extension	xsend_node

The list of commands that the primitives are grouped in:

above	6	hmove	2	omit	1
accent	1	hrule	1	par_end	1
adv_register	3	hskip	5	prefix	3
assign_dimen	17	hyph_data	2	radical	1
assign_glue	17	if_test	10	register	3
assign_int	37	ignore_space	1	relax	1
assign_tex_info	1	input	1	set_aux	2
assign_toks	2	insert	1	set_box	1
break_penalty	1	ital_corr	1	set_family	1
car_ret	1	kern	1	set_font	1
case_branch	1	last_skip	1	set_interaction	4
case_shift	2	leader_ship	4	set_register	3
char_num	1	left_right	2	set_shape	1
def	4	let	1	start_par	2
def_code	6	limit_switch	1	stop	2
def_family	3	make_box	7	tab_mark	1
def_font	1	mark	1	the	2
delim_num	1	math_accent	1	top_bot_mark	5
discretionary	2	math_char_num	1	unbox	1
div_register	3	math_comp	9	unskip	1
else_code	1	math_style	4	vadjust	1
eq_no	2	message	2	valign	1
ex_space	1	mkern	1	vcenter	1
extension	4	mskip	1	vmove	2
group_begin	1	mult_register	3	vrule	1
group_end	1	no_align	1	vskip	5
halign	1	non_script	1	xray	4
hang_indent	1	number	1		

So what are the most significant differences with the SAIL version? We see many more primitives and quite some are for setting and getting parameters. These are now organized into integers, dimensions, glue, muglue and token lists.

A noticeable difference with today's TEX is the `set_font` that uses this `\:` shortcut. It's good to know that we still have limited memory. Machines got more powerful but even a DEC 10 (or 20) had to serve more than one user and given that each seemingly had 500 kB running TEX puts some strain on the system (swapping memory and such). It is why there were two versions, one that could dump a format and another that just loaded it and therefore could drop some code in favor of using more memory for processing.

But fonts, although small by today's standards took space so the number was limited. On the other hand, we got away from the short (sometimes cryptic) names and have way more primitives. It feels good.

Watch commands like `\texinfo`, `\groupend` and `\groupbegin`, `\hangindent` with its own command class, `\xoverx`. But also look at the various `\adv...` commands and `\minus`. Conditionals have this `\ifeven` the future negated `\ifodd`; the `\ifabsent` check will at some point become `\ifvoid` and `\case` will migrate to `\ifcase`. Also consider `\varunit` and `\:` as we will see these change later.

We see the concept of mu glue show up, and we see `\thinmskip` pop up. Interestingly that one became `\thinmuskip`, probably because `\mskip` is a more general skipper and therefore we need to avoid confusion. It's in the details. Going from `disp` to `display` also feels natural. We see `\xabovex` and companions, `\chcode`, quite some `\set...`, `\adv...`, `\mult...` and `\div...`. Maybe if there had't been so many we'd have ended up with `\add` and `\subtract` instead of just `\advance`.

Watch how the `\shipout` is treated as a leader. The operands (`chr`) now use more symbolic names but not all of them. Interaction internally uses a variable with four values but each got its primitive. The tracing options were split into independent parameters.

7 New TEX82–89

The current version of TEX has the next list of primitives. There are some differences in names but more significant is that we have a different grouping in commands. As with the previous versions, not all operands have symbolic names.

\lineskip	assign_glue	glue_base+line_skip_code
\baselineskip	assign_glue	glue_base+baseline_skip_code
\parskip	assign_glue	glue_base+par_skip_code
\abovedisplayskip	assign_glue	glue_base+above_display_skip_code
\belowdisplayskip	assign_glue	glue_base+below_display_skip_code
\abovedisplayshortskip	assign_glue	glue_base+above_display_short_skip_code
\belowdisplayshortskip	assign_glue	glue_base+below_display_short_skip_code
\leftskip	assign_glue	glue_base+left_skip_code
\rightskip	assign_glue	glue_base+right_skip_code
\topskip	assign_glue	glue_base+top_skip_code
\splittopskip	assign_glue	glue_base+split_top_skip_code
\tabskip	assign_glue	glue_base+tab_skip_code
\spaceskip	assign_glue	glue_base+space_skip_code
\xspaceskip	assign_glue	glue_base+xspace_skip_code
\parfillskip	assign_glue	glue_base+par_fill_skip_code
\thinmuskip	assign_mu_glue	glue_base+thin_mu_skip_code
\medmuskip	assign_mu_glue	glue_base+med_mu_skip_code
\thickmuskip	assign_mu_glue	glue_base+thick_mu_skip_code
\output	assign_toks	output_routine_loc
\everypar	assign_toks	every_par_loc
\everymath	assign_toks	every_math_loc
\everydisplay	assign_toks	every_display_loc
\everyhbox	assign_toks	every_hbox_loc

\everyvbox	assign_toks	every_vbox_loc
\everyjob	assign_toks	every_job_loc
\everycr	assign_toks	every_cr_loc
\errhelp	assign_toks	err_help_loc
\pretolerance	assign_int	int_base+pretolerance_code
\tolerance	assign_int	int_base+tolerance_code
\linepenalty	assign_int	int_base+line_penalty_code
\hyphenpenalty	assign_int	int_base+hyphen_penalty_code
\exhyphenpenalty	assign_int	int_base+ex_hyphen_penalty_code
\clubpenalty	assign_int	int_base+club_penalty_code
\widowpenalty	assign_int	int_base+widow_penalty_code
\displaywidowpenalty	assign_int	int_base+display_widow_penalty_code
\brokenpenalty	assign_int	int_base+broken_penalty_code
\binoppenalty	assign_int	int_base+bin_op_penalty_code
\relpenalty	assign_int	int_base+rel_penalty_code
\predisplaypenalty	assign_int	int_base+pre_display_penalty_code
\postdisplaypenalty	assign_int	int_base+post_display_penalty_code
\interlinepenalty	assign_int	int_base+inter_line_penalty_code
\doublehyphendemerits	assign_int	int_base+double_hyphen_demerits_code
\finalhyphendemerits	assign_int	int_base+final_hyphen_demerits_code
\adjdemerits	assign_int	int_base+adj_demerits_code
\mag	assign_int	int_base+mag_code
\delimiterfactor	assign_int	int_base+delimiter_factor_code
\looseness	assign_int	int_base+looseness_code
\time	assign_int	int_base+time_code
\day	assign_int	int_base+day_code
\month	assign_int	int_base+month_code
\year	assign_int	int_base+year_code
\showboxbreadth	assign_int	int_base+show_box_breadth_code
\showboxdepth	assign_int	int_base+show_box_depth_code
\hbadness	assign_int	int_base+hbadness_code
\vbadness	assign_int	int_base+vbadness_code
\pausing	assign_int	int_base+pausing_code
\tracingonline	assign_int	int_base+tracing_online_code
\tracingmacros	assign_int	int_base+tracing_macros_code
\tracingstats	assign_int	int_base+tracing_stats_code
\tracingparagraphs	assign_int	int_base+tracing_paragraphs_code
\tracingpages	assign_int	int_base+tracing_pages_code
\tracingoutput	assign_int	int_base+tracing_output_code
\tracinglostchars	assign_int	int_base+tracing_lost_chars_code
\tracingcommands	assign_int	int_base+tracing_commands_code
\tracingrestores	assign_int	int_base+tracing_restores_code
\uchyph	assign_int	int_base+uc_hyph_code
\outputpenalty	assign_int	int_base+output_penalty_code
\maxdeadcycles	assign_int	int_base+max_dead_cycles_code
\hangafter	assign_int	int_base+hang_after_code
\floatingpenalty	assign_int	int_base+floating_penalty_code
\globaldefs	assign_int	int_base+global_defs_code
\fam	assign_int	int_base+cur_fam_code
\escapechar	assign_int	int_base+escape_char_code
\defaulthyphenchar	assign_int	int_base+default_hyphen_char_code
\defaultskewchar	assign_int	int_base+default_skew_char_code
\endlinechar	assign_int	int_base+end_line_char_code
\newlinechar	assign_int	int_base+new_line_char_code

\language	assign_int	int_base+language_code
\lefthyphenmin	assign_int	int_base+left_hyphen_min_code
\righthyphenmin	assign_int	int_base+right_hyphen_min_code
\holdinginserts	assign_int	int_base+holding_inserts_code
\errorcontextlines	assign_int	int_base+error_context_lines_code
\parindent	assign_dimen	dimen_base+par_indent_code
\mathsurround	assign_dimen	dimen_base+math_surround_code
\lineskiplimit	assign_dimen	dimen_base+line_skip_limit_code
\hsize	assign_dimen	dimen_base+hsize_code
\vsize	assign_dimen	dimen_base+vsize_code
\maxdepth	assign_dimen	dimen_base+max_depth_code
\splitmaxdepth	assign_dimen	dimen_base+split_max_depth_code
\boxmaxdepth	assign_dimen	dimen_base+box_max_depth_code
\hfuzz	assign_dimen	dimen_base+hfuzz_code
\vfuzz	assign_dimen	dimen_base+vfuzz_code
\delimitershortfall	assign_dimen	dimen_base+delimiter_shortfall_code
\nulldelimiterspace	assign_dimen	dimen_base+null_delimiter_space_code
\scriptspace	assign_dimen	dimen_base+script_space_code
\predisplaysize	assign_dimen	dimen_base+pre_display_size_code
\displaywidth	assign_dimen	dimen_base+display_width_code
\displayindent	assign_dimen	dimen_base+display_indent_code
\overfullrule	assign_dimen	dimen_base+overfull_rule_code
\hangindent	assign_dimen	dimen_base+hang_indent_code
\hoffset	assign_dimen	dimen_base+h_offset_code
\voffset	assign_dimen	dimen_base+v_offset_code
\emergencystretch	assign_dimen	dimen_base+emergency_stretch_code
\	ex_space	0
\/	ital_corr	0
\accent	accent	0
\advance	advance	0
\afterassignment	after_assignment	0
\aftergroup	after_group	0
\begingroup	begin_group	0
\char	char_num	0
\csname	cs_name	0
\delimiter	delim_num	0
\divide	divide	0
\endcsname	end_cs_name	0
\endgroup	end_group	0
\expandafter	expand_after	0
\font	def_font	0
\fontdimen	assign_font_dimen	0
\halign	halign	0
\hrule	hrule	0
\ignorespaces	ignore_spaces	0
\insert	insert	0
\mark	mark	0
\mathaccent	math_accent	0
\mathchar	math_char_num	0
\mathchoice	math_choice	0
\multiply	multiply	0
\noalign	no_align	0
\noboundary	no_boundary	0
\noexpand	no_expand	0

\nonscript	non_script	0
\omit	omit	0
\parshape	set_shape	0
\penalty	break_penalty	0
\prevgraf	set_prev_graf	0
\radical	radical	0
\read	read_to_cs	0
\relax	relax	256
\setbox	set_box	0
\the	the	0
\toks	toks_register	0
\vadjust	vadjust	0
\valign	valign	0
\vcenter	vcenter	0
\vrule	vrule	0
\par	par_end	256
\input	input	0
\endinput	input	1
\topmark	top_bot_mark	top_mark_code
\firstmark	top_bot_mark	first_mark_code
\botmark	top_bot_mark	bot_mark_code
\splitfirstmark	top_bot_mark	split_first_mark_code
\splitbotmark	top_bot_mark	split_bot_mark_code
\count	register	int_val
\dimen	register	dimen_val
\skip	register	glue_val
\muskip	register	mu_val
\spacefactor	set_aux	hmode
\prevdepth	set_aux	vmode
\deadcycles	set_page_int	0
\insertpenalties	set_page_int	1
\wd	set_box_dimen	width_offset
\ht	set_box_dimen	height_offset
\dp	set_box_dimen	depth_offset
\lastpenalty	last_item	int_val
\lastkern	last_item	dimen_val
\lastskip	last_item	glue_val
\inputlineno	last_item	input_line_no_code
\badness	last_item	badness_code
\number	convert	number_code
\romannumeral	convert	roman_numeral_code
\string	convert	string_code
\meaning	convert	meaning_code
\fontname	convert	font_name_code
\jobname	convert	job_name_code
\if	if_test	if_char_code
\ifcat	if_test	if_cat_code
\ifnum	if_test	if_int_code
\ifdim	if_test	if_dim_code
\ifodd	if_test	if_odd_code
\ifvmode	if_test	if_vmode_code
\ifhmode	if_test	if_hmode_code
\ifmmode	if_test	if_mmode_code
\ifinner	if_test	if_inner_code

\ifvoid	if_test	if_void_code
\ifhbox	if_test	if_hbox_code
\ifvbox	if_test	if_vbox_code
\ifx	if_test	ifx_code
\ifeof	if_test	if_eof_code
\iftrue	if_test	if_true_code
\iffalse	if_test	if_false_code
\ifcase	if_test	if_case_code
\fi	fi_or_else	fi_code
\or	fi_or_else	or_code
\else	fi_or_else	else_code
\nullfont	set_font	null_font
\span	tab_mark	span_code
\cr	car_ret	cr_code
\crcr	car_ret	cr_cr_code
\pagegoal	set_page_dimen	0
\pagetotal	set_page_dimen	1
\pagestretch	set_page_dimen	2
\pagefilstretch	set_page_dimen	3
\pagefillstretch	set_page_dimen	4
\pagefilllstretch	set_page_dimen	5
\pageshrink	set_page_dimen	6
\pagedepth	set_page_dimen	7
\end	stop	0
\dump	stop	1
\hskip	hskip	skip_code
\hfil	hskip	fil_code
\hfill	hskip	fill_code
\hss	hskip	ss_code
\hfilneg	hskip	fil_neg_code
\vskip	vskip	skip_code
\vfil	vskip	fil_code
\vfill	vskip	fill_code
\vss	vskip	ss_code
\vfilneg	vskip	fil_neg_code
\mskip	mskip	mskip_code
\kern	kern	explicit
\mkern	mkern	mu_glue
\moveleft	hmove	1
\moveright	hmove	0
\raise	vmove	1
\lower	vmove	0
\box	make_box	box_code
\copy	make_box	copy_code
\lastbox	make_box	last_box_code
\vsplit	make_box	vsplit_code
\vtop	make_box	vtop_code
\vbox	make_box	vtop_code+vmode
\hbox	make_box	vtop_code+hmode
\shipout	leader_ship	a_leaders-1
\leaders	leader_ship	a_leaders
\cleaders	leader_ship	c_leaders
\xleaders	leader_ship	x_leaders
\indent	start_par	1

\noindent	start_par	0
\unpenalty	remove_item	penalty_node
\unkern	remove_item	kern_node
\unskip	remove_item	glue_node
\unhbox	un_hbox	box_code
\unhcopy	un_hbox	copy_code
\unvbox	un_vbox	box_code
\unvcopy	un_vbox	copy_code
\-	discretionary	1
\discretionary	discretionary	0
\eqno	eq_no	0
\leqno	eq_no	1
\mathord	math_comp	ord_noad
\mathop	math_comp	op_noad
\mathbin	math_comp	bin_noad
\mathrel	math_comp	rel_noad
\mathopen	math_comp	open_noad
\mathclose	math_comp	close_noad
\mathpunct	math_comp	punct_noad
\mathinner	math_comp	inner_noad
\underline	math_comp	under_noad
\overline	math_comp	over_noad
\displaylimits	limit_switch	normal
\limits	limit_switch	limits
\nolimits	limit_switch	no_limits
\displaystyle	math_style	display_style
\textstyle	math_style	text_style
\scriptstyle	math_style	script_style
\scriptscriptstyle	math_style	script_script_style
\above	above	above_code
\over	above	over_code
\atop	above	atop_code
\abovewithdelims	above	delimited_code+above_code
\overwithdelims	above	delimited_code+over_code
\atopwithdelims	above	delimited_code+atop_code
\left	left_right	left_noad
\right	left_right	right_noad
\long	prefix	1
\outer	prefix	2
\global	prefix	4
\def	def	0
\gdef	def	1
\edef	def	2
\xdef	def	3
\let	let	normal
\futurelet	let	normal+1
\chardef	shorthand_def	char_def_code
\mathchardef	shorthand_def	math_char_def_code
\countdef	shorthand_def	count_def_code
\dimendef	shorthand_def	dimen_def_code
\skipdef	shorthand_def	skip_def_code
\muskipdef	shorthand_def	mu_skip_def_code
\toksdef	shorthand_def	toks_def_code
\catcode	def_code	cat_code_base

\mathcode	def_code	math_code_base
\lccode	def_code	lc_code_base
\uccode	def_code	uc_code_base
\sfcode	def_code	sf_code_base
\delcode	def_code	del_code_base
\textfont	def_family	math_font_base
\scriptfont	def_family	math_font_base+script_size
\scriptscriptfont	def_family	math_font_base+script_script_size
\hyphenation	hyph_data	0
\patterns	hyph_data	1
\hyphenchar	assign_font_int	0
\skewchar	assign_font_int	1
\batchmode	set_interaction	batch_mode
\nonstopmode	set_interaction	nonstop_mode
\scrollmode	set_interaction	scroll_mode
\errorstopmode	set_interaction	error_stop_mode
\openin	in_stream	1
\closein	in_stream	0
\message	message	0
\errmessage	message	1
\lowercase	case_shift	lc_code_base
\uppercase	case_shift	uc_code_base
\show	xray	show_code
\showbox	xray	show_box_code
\showthe	xray	show_the_code
\showlists	xray	show_lists_code
\openout	extension	open_node
\write	extension	write_node
\closeout	extension	close_node
\special	extension	special_node
\immediate	extension	immediate_code
\setlanguage	extension	set_language_code

Again we show the list of commands. It's about as large as the previous one although we got rid of some. Looking at the list is a test of how well you know the current set of official TEX primitives.

above	6	car_ret	2	eq_no	2
accent	1	case_shift	2	ex_space	1
advance	1	char_num	1	expand_after	1
after_assignment	1	convert	6	extension	6
after_group	1	cs_name	1	fi_or_else	3
assign_dimen	21	def	4	halign	1
assign_font_dimen	1	def_code	6	hmove	2
assign_font_int	2	def_family	3	hrule	1
assign_glue	15	def_font	1	hskip	5
assign_int	55	delim_num	1	hyph_data	2
assign_mu_glue	3	discretionary	2	if_test	17
assign_toks	9	divide	1	ignore_spaces	1
begin_group	1	end_cs_name	1	in_stream	2
break_penalty	1	end_group	1	input	2

insert	1	no_align	1	set_prev_graf	1
ital_corr	1	no_boundary	1	set_shape	1
kern	1	no_expand	1	shorthand_def	7
last_item	5	non_script	1	start_par	2
leader_ship	4	omit	1	stop	2
left_right	2	par_end	1	tab_mark	1
let	2	prefix	3	the	1
limit_switch	3	radical	1	toks_register	1
make_box	7	read_to_cs	1	top_bot_mark	5
mark	1	register	4	un_hbox	2
math_accent	1	relax	1	un_vbox	2
math_char_num	1	remove_item	3	vadjust	1
math_choice	1	set_aux	2	valign	1
math_comp	10	set_box	1	vcenter	1
math_style	4	set_box_dimen	3	vmove	2
message	2	set_font	1	vrule	1
mkern	1	set_interaction	4	vskip	5
mskip	1	set_page_dimen	8	xray	4
multiply	1	set_page_int	2		

We will make it easier to see the changes in organizing the commands. The second column shows the number of primitives in the older version, the third column refers to the current version. The bold entries are unchanged names.

above	6	6	**def**	4	4
accent	1	1	**def_code**	6	6
adv_register	3		**def_family**	3	3
advance		1	**def_font**	1	1
after_assignment		1	**delim_num**	1	1
after_group		1	**discretionary**	2	2
assign_dimen	17	21	div_register	3	
assign_font_dimen		1	divide		1
assign_font_int		2	else_code	1	
assign_glue	17	15	end_cs_name		1
assign_int	37	55	end_group		1
assign_mu_glue		3	**eq_no**	2	2
assign_tex_info	1		**ex_space**	1	1
assign_toks	2	9	expand_after		1
begin_group		1	**extension**	4	6
break_penalty	1	1	fi_or_else		3
car_ret	1	2	group_begin	1	
case_branch	1		group_end	1	
case_shift	2	2	**halign**	1	1
char_num	1	1	hang_indent	1	
convert		6	**hmove**	2	2
cs_name		1	**hrule**	1	1

hskip	5	5	**prefix**	3	3
hyph_data	2	2	**radical**	1	1
if_test	10	17	read_to_cs		1
ignore_space	1		**register**	3	4
ignore_spaces		1	**relax**	1	1
in_stream		2	remove_item		3
input	1	2	**set_aux**	2	2
insert	1	1	**set_box**	1	1
ital_corr	1	1	set_box_dimen		3
kern	1	1	set_family	1	
last_item		5	**set_font**	1	1
last_skip	1		**set_interaction**	4	4
leader_ship	4	4	set_page_dimen		8
left_right	2	2	set_page_int		2
let	1	2	set_prev_graf		1
limit_switch	1	3	set_register	3	
make_box	7	7	**set_shape**	1	1
mark	1	1	shorthand_def		7
math_accent	1	1	**start_par**	2	2
math_char_num	1	1	**stop**	2	2
math_choice		1	**tab_mark**	1	1
math_comp	9	10	**the**	2	1
math_style	4	4	toks_register		1
message	2	2	**top_bot_mark**	5	5
mkern	1	1	un_hbox		2
mskip	1	1	un_vbox		2
mult_register	3		unbox	1	
multiply		1	unskip	1	
no_align	1	1	**vadjust**	1	1
no_boundary		1	**valign**	1	1
no_expand		1	**vcenter**	1	1
non_script	1	1	**vmove**	2	2
number	1		**vrule**	1	1
omit	1	1	**vskip**	5	5
par_end	1	1	**xray**	4	4

It is interesting to see how TEX evolved and got more organized with each iteration. It also made me feel a bit less for reorganizing these commands a bit more. In LuaTEX we added primitives, just like for instance e-TEX and pdfTEX had done. In LuaMetaTEX we added even more. In one of the presentations Don shows a trick to add two dimensions: do two `\hskip`'s in a `\hbox` and use the width as sum. He then explains that one can also subtract, multiply and divide using such trickery. However, in the final version of TEX we got `\advance`, `\divide` and `\multiply`, and e-TEX later added simple expressions. So in the end, the fact that LuaMetaTEX got a more powerful expression mechanism and additional data types feels kind of natural. So, why not show the LuaMetaTeX command grouping here? This is what we have 25 years later:

`accent`	1
`active_char`	
`after_something`	8
`alignment`	
`alignment_tab`	1
`arithmic`	10
`association`	1
`auxiliary`	5
`begin_group`	3
`begin_local`	12
`begin_paragraph`	8
`boundary`	10
`box_property`	29
`call`	
`case_shift`	2
`catcode_table`	3
`char_given`	
`char_number`	2
`combine_toks`	10
`comment`	
`constant_call`	
`convert`	35
`cs_name`	4
`deep_frozen_dont_expand`	
`deep_frozen_end_template`	
`deep_frozen_keep_constant`	
`def`	10
`define_char_code`	12
`define_family`	3
`define_font`	1
`delimiter_number`	2
`dimension`	
`discretionary`	4
`end_cs_name`	1
`end_group`	3
`end_job`	2
`end_line`	
`end_local`	1
`end_match`	
`end_paragraph`	2
`end_template`	7
`equation_number`	2
`escape`	
`expand_after`	15
`explicit_space`	2
`font_property`	15
`fontspec`	
`get_mark`	11
`gluespec`	
`halign`	1
`hmove`	2
`hrule`	3
`hskip`	5
`hyphenation`	8
`if_test`	68
`ignore`	
`ignore_something`	6
`index`	
`input`	10
`insert`	1
`integer`	
`interaction`	4
`internal_attribute`	
`internal_attribute_reference`	
`internal_box_reference`	
`internal_dimension`	32
`internal_dimension_reference`	
`internal_glue`	29
`internal_glue_reference`	
`internal_integer`	199
`internal_integer_reference`	
`internal_muglue`	5
`internal_muglue_reference`	
`internal_posit`	
`internal_posit_reference`	
`internal_toks`	15
`internal_toks_reference`	
`invalid_char`	
`italic_correction`	4
`iterator_value`	
`kern`	3
`leader`	5
`left_brace`	
`legacy`	1

`let`	15
`letter`	
`local_box`	4
`lua_call`	
`lua_function_call`	2
`lua_local_call`	
`lua_protected_call`	
`lua_semiprotected_call`	
`lua_value`	
`make_box`	27
`mark`	4
`match`	
`math_accent`	2
`math_char_number`	5
`math_choice`	3
`math_component`	11
`math_fence`	8
`math_fraction`	16
`math_modifier`	14
`math_parameter`	127
`math_radical`	10
`math_script`	16
`math_shift`	
`math_shift_cs`	6
`math_style`	21
`mathspec`	
`message`	2
`mkern`	1
`mskip`	2
`mugluespec`	
`mvl`	2
`no_expand`	1
`node`	
`other_char`	
`page_property`	42
`parameter`	2
`parameter_reference`	
`penalty`	3
`posit`	
`prefix`	22
`protected_call`	
`register`	7
`register_attribute`	
`register_attribute_reference`	
`register_dimension`	
`register_dimension_reference`	
`register_glue`	
`register_glue_reference`	
`register_integer`	
`register_integer_reference`	
`register_muglue`	
`register_muglue_reference`	
`register_posit`	
`register_posit_reference`	
`register_toks`	
`register_toks_reference`	
`relax`	3
`remove_item`	4
`right_brace`	
`semi_protected_call`	
`set_box`	1
`set_font`	1
`shorthand_def`	20
`some_item`	115
`spacer`	
`specification`	18
`specification_reference`	
`specificationspec`	
`subscript`	
`superscript`	
`the`	8
`tolerant_call`	
`tolerant_protected_call`	
`tolerant_semi_protected_call`	
`un_hbox`	3
`un_vbox`	8
`undefined_cs`	
`unit_reference`	
`vadjust`	1
`valign`	1
`vcenter`	1
`vmove`	2
`vrule`	4
`vskip`	5
`xray`	9

Some are renames, others regrouping, but there are also new ones. For instance the command group `association` deals with user defined units (bound to a register, macro or some Lua call). The `begin_local` relates to `\localcontrol` and friends, a way to exercise the main loop inside a macro and avoid side effects; a bit similar to `vardef` in MetaPost. Boundaries are new and have a `boundary` command group. The various parameters (variables) are split into internal and register (user) ones.

Not all commands have a primitive (this is also true for the predecessors where we didn't show them) but here we just show them with no primitive count. We have registers (way more than original TEX) but also have additional pseudo registers that are basically (efficient) macros. This means that we can generate a version with less registers which, even with a decent set, saves about the same amount of memory that good old TEX has available for processing documents, including its own binary.[19] Anyway, some of the codes with no number have similar command codes in old and new TEX, we didn't invent all those wheels.

It makes no sense to show the new primitives but we have for instance iterators that have a command category but primitives that are in other groups. This has to do with the way they behave. Of course we have a bunch of Lua related ones too. The `\..._call` commands are fast accessors to specific definitions, like `\protected` or `\tolerant` ones. Command groups like `active_char` and `letter` are also present in the reference TEX engine, as they relate to the catcodes, but characters are not primitives. The fact that we have Unicode and therefore huge slices of potential entries in TEX's equivalents space already makes for different internals anyway. But looking at the tables of various engines can at least give an idea of how the engines evolved. It definitely made watching these 40 year old videos so much more interesting and fun.

8 The timeline

We started with a rough timeline but there is a very good source for detective work: the `errorlog.tex` file also typeset in Knuths Digital Typography. I wish I had the discipline to come up with something like that but I can use the fact that I write intermediate wrap-ups as a lame excuse. From 1978 onward you can read what bugs were solved, what renames happened, what got introduced, etc. Let's mention a few entries that relate to things discussed earlier in this document, especially in the intermezzo section.

At 19 May 1978 we note "G249. Add a `\topbaseline` feature [later called `\topskip`]. @1001". Before that we had some skips after top inserts and before bottom inserts but these became skip registers eventually. On 5 Aug 1978 we read "G336. Generalize `\pageno` to `\count <digit>`. @236" but `\pageno` later became the de-facto standard for `\the\count 0` so it never really went away. From mid 1979 it becomes clear that the new par-builder is being worked on, we also see the mentioned fillers show up on 23 Jul 1979: "E409. Introduce new primitives `\hfil`, `\vfil`, `\hfilneg`, `\vfilneg`. @1058"

[19] The e-TEX extension pushed TEX from 256 to 32.786 and LuaTEX doubled that but it makes no sense to have that many: who needs that many muskips or attributes? Some 8K seems more than enough for each with probably 4K for the glue and muglue.

Not mentioned in the book as upcoming are active characters. These are mentioned 25 Jan 1980: “G427. Introduce active characters (one-stroke control sequences). [I don’t yet go all the way: The meanings of `x` and `\x` have to be identical.] @344” Actually, active characters are an interesting concept although in today’s usage (think Unicode) they make less sense.

We already saw `\def` and variants being in the SAIL version but in 1980 we also got “G444. Add a new `\newname` feature (soon changed to `\let`). @1221”.

A milestone is 13 Jun 1980: “Today I’m beginning to overhaul the line-breaking routine, and I’ll also install miscellaneous goodies.” There is also a new feature: “G459. Add a new parameter `\loose` [later `\looseness`]; now parameters are allowed to take negative values. @875”. A day later 14 Jun 1980 we read “Q461. Install new line-breaking routines, including `\parshape`. (These major changes are introduced as Michael Plass and I write our article.) @813”.

The inserts are maturing: “G482. Add new `\topsep` and `\botsep` features. [These are TEX78’s way to put space at the edge of inserts, replaced in TEX82 by the `\skip` register corresponding to an `\insert` class.] @1009”. These changes happen when Don is also writing the upcoming books, and I think it demonstrates his valid point that writing a manual is a great way to improve the code, for instance with respect to consistency. Implementation, documenting and using go hand-in-hand.

Here’s one for us: “G490. Add the dimension `cc` for European users. @458”. I never used it as I prefer `cm`, `mm` and `pt`, as silent protest against the American `in` and `bp`.

This is a nice one: “C491. Make `scan_keyword` match uppercase letters as alternatives to lowercase ones (suggested by Barbara Beeton’s experiments with `\uppercase`). @407”.

Then there is the comment “Am freezing current program as version −0.25; a week of tug lectures begins tomorrow.” Are these the ones from the video? Was that something tug? In one of those videos the 5 Aug 1982 change “IX158. The `.err` file should be `.log` instead. @534” is announced, a decision driven by user input during a break in the course.

Around that time there is also the optimistic “I believe the `line_break` routine has passed its test perfectly.” and then “FX247. Initialize `second_indent` in the easy case. @848”.[20]

The class related character definitions show up: “GX260. Introduce new primitive `\mathchardef`, to save space and time. @1224” which is nicer that a mix with character catcodes. In this perspective it’s good to remind the reader that we have at most 256 characters in a font. In a modern Unicode font we have way more which also means that

20 Mikael and I still try to understand what this easy case refers to, other than that it relates to looseness in the current version. Thanks to “IX334 X199. Introduce serial numbers in line-break records, improving readability and independence. @846” we can indeed see how many passive nodes get set.

we have character codes that need a 32 bit integer. Some of the original TeX approaches in LuaTeX got a different implementation, for instance by using sparse arrays.[21]

Relatively late we see `\:` being replaced; the videos indeed still use that short command: "G545. Install a major change: Fonts now have identifiers instead of code letters. Eliminate the `\:` primitive, and give corresponding new features to `\the`. @209".

The par builder gets an update: "A554. Compute demerits more suitably by adding a penalty squared, instead of adding penalties before squaring. @859" and "Previously a slightly loose hyphenated line followed by a decent line was considered worse than a decent hyphenated line followed by a quite loose line." Also "E566. Omit the 'first pass' and try hyphenations immediately, if `\pretolerance` is negative (suggested by DRF). @863". How many users do the latter? We mentioned this already but it happens late 1982: "C578. Change `\hangindent` to a normal dimension parameter. [It had been a combination of `\hangindent` and `\hangafter`, with special syntax.] @247".

We also mentioned the `\ifeven` to `\ifodd` change; the new one is mentioned: "S591 564. Make `\ifodd 1\else` legal by introducing `if_code`. @489".

The beginning of 1983 also marks the beginning of new grouping names: "C597. Rename `\groupbegin` and `\groupend` as `\begingroup` and `\endgroup`. @265".

Documenting likely lead to: "C639 607. Remove the kludgy math codes introduced earlier; make `\fam` a normal integer parameter and allow `\mathcode` to equal 2^{15}. @1233". We saw in basic that octal was popular but "I641 639. Replace octal output (`print_octal`) by hexadecimal (`print_hex`) so that math codes are clearer. @67".

At 22 Feb 1979 we could read "G387. Add `vu` and `\varunit`. [TeX82 will eventually allow arbitrary internal dimensions as units of measure.] @453" and that announcement between square brackets lead to 25 May 1983 mentioning "G695 X231. Remove `dm` and `vu`; allow the more general `.5\hsize`". We also read "@455. C696. Change `\texinfo <f> <n>` to `\fontdimen <n> <f>`. @578".

We also mentioned the introduction of units to access box dimensions but the 27 May 1983 entry tells "G703 695. Replace (and generalize) the previous uses of `ht`, `wd`, and `dp` in dimensions by introducing the new control sequences `\ht`, `\wd`, and `\dp`. @1247".

If you're interested how software development went in the years 1978 to 1983 this error log is a must-read because it actually is a development log. Between the lines you can read about compilation and compiler issues, access to computers during day and night, and evolving features. It makes one who is only accustomed to fast personal computers, screen editors, maybe code specific features in editors, seemingly unlimited memory and disc space a bit more humble. It also makes clear that thinking beforehand as well as printing and reading a source was part of a skill-set, if only because trial-and-error or just trial-after-coding was less of an option.

[21] In `musing-neat.tex` I explore a bit how characters are stored in a node list, the suggested way to handle large fonts and why we do it differently.

9 Conclusion

It was interesting to compare the SAIL/TEX82 lectures, and TEX82 as-of-now with each other, especially because I spent a few decades with ConTEXt friends on extending LuaTEX and later LuaMetaTEX. I had of course browsed the error log but checking it again in this perspective is nice. Looking at how it all evolved also gives some clues why things are as they are now; to some extent it all makes even more sense: limitations and possibilities. It also matters (for me) that in the meantime, for quite some years, I have been involved in and have been playing around with extending (core features of) the engine, also in the perspective of developing ConTEXt, which has its demands and stresses the machinery.

One has to go back in time and watch the development process, the teamwork, and the inspiring leadership, working towards quality, that made it possible. The success of TEX and friends very much related to the way Don Knuth interacted with the community, and the effort he put in all this, often setting aside his main priority, the art books. The result is still pretty valid, and those who claim otherwise can take a lesson from that. After all, in software development illogical arguments, sometimes ridiculing (the past), bragging about trivial achievements, quick and dirty finalizing, lack of quality control and what more, are quite common in these commercially driven developments. And we're not even talking of using generative artificial intelligence to come up with new code, based on old code, presented as original. But watching these videos confirms why I like TEX, and why I got attracted to it, seeing the books, understanding little without the ability to run the programs, picking up on that, and ending up in the ConTEXt community. Figuring out solutions using a framework for typesetting, one that had been designed with a future usage in mind, can be fun. It's pretty hard to beat!

An important lesson is that coming up with a (original) solution (for your problem) that is also future proof takes time. One needs to play with it and be willing to look at it from a different angle. It makes little sense to work in isolation because constructive user input helps. To that I personally add that if you don't like some development, just ignore it; there is no need to waste time on competition and claiming that this is better than that and that everyone should use whatever you like. Just think of this: TEX has been around for decades for good reasons.

Hans Hagen
December 2025
Hasselt NL