concerns	: e–TEX extensions
date	: February 10, 1998
mark	: PRAGMA.ADE.ETEX/HH

... your earlier [large] list of desiderata, drawn up in conjunction with Taco. Could you possibly annotate it, giving a suggested syntax (often clarifies things), expanded semantics and possible applications? ...

Phil

This draft tries to explain some of the wishes of Piet, Taco and me. Don't hesitate to ask for more explanations.

Hans

1 Remark

Many of the primitives below are currently implemented in CONTEXT as macros. More primitive level support would however speed up things considerably.

2 Generalized `\uccode`/`\lccode`

These primitives can be (ab)used for converting characters into other ones. The memory constraints of both primitives and rather low, but reassigning them takes processing time. I therefore suggest introducing a new mapping primitive

```
\newcode\mycode using \codemethod
```

Where `\codemethod` can be used as:

```
\codemethod{\special{...}}
```

This new primitive would enable a more efficient implementation of output filters needed for support of non–TEXbased languages, like PostScript, PDF, METAPOST or JAVA-SCRIPT.

By default Plain TEX could define:

```
\newcode\uccode\uppercase
\newcode\lccode\lowercase
```

The new code primitive should respect the current constraints and functionality imposed by those two primitives.

3 Mapping character tokens into `\csname`'s

A related primitive deals with a bit more advanced conversion. Consider for instance that a `(` must be output in a special as `\(`.

```
\newmapping\mymapping\usemapping
\setmap\mymapping<charcode>={}
```

The `\setmap` primitive for the conversion mentioned is:

```
\setmap\mymapping`(={\string\(}
```

or maybe better (and more simple):

```
\setmap\mymapping`(={\(}   % \( no csname, but two characters!
```

And can be used as:

```
\bgroup\usemapping\special{something (funny)}\egroup
```

4 Binary file I/O

This topic was brought up by Taco. Binary I/O is not that hard to implement and can be considered an extension to the existing file I/O mechanism. More alternatives come into mind.

```
\openbinaryin
\openbinaryout
```

after which the file is tagged as binary and all reads and writes are bytes (actually char codes/8 bit numbers). However, a more versatile alternative is:

```
\binaryread
\binarywrite
```

This would permit mixed binary and line based support. One possible application for this extension is prescanning TIFF images. At this moment prescanning is up to an external program. The `\binaryread` primitive returns a `\chardef`, so the next is valid:

```
\binaryread\sometoken \the\sometoken
```

Complementary `\binarywrite` expects a `\chardef`'d `\csname`.

5 An old one: command line arguments

Quite often I would like to specify on the command line some run time specific processing instructions, like PDF or DVI output, color/gray support, page imposition, DVI driver support etc. One can do as such by using for instance set up files and PERL scripts, but command line handling should be part of any decent program. (Let's kick TEX into the next century!)

```
\getflags
```

One argument against this primitive is that not all operating systems (PASCAL compilers) support this feature, but one may wonder if on such (old) systems there is much need of e-tex. One argument in favour is that for instance Web2c already supports some arguments (like enabling `\write18`). And why not also support:

```
\getenvironment{identifier} to \csname
```

Or something like that.

6 Setting `\jobname`

Another of Taco's suggestions concerns changing the jobs name:

```
\jobname=<whatever>
```

One of the associated features would be that the name of the DVI file (which normally is opened as soon as the first page is shipped out) is taken from the current value of `\jobname`.

7 A `\looseness` that works

The functionality of the next primitive directly follows from its name:

```
\forcelooseness
```

8 Getting on time

When optimizing (low level) macros, obtaining the current time is a necessity, so we need the sytem counter:

```
\currenttime
```

9 Stack support

Some kind of (grouping independent) stack support is also welcome, including push, pop and dup primitives. The stack is a token list stack. Taco knows more about this.

10 Getting rid of spaces

A primitive:

```
\ignoreallspaces
```

that ignores all spaces until the first typeset token, even those resulting from expansion, is not only of interest for macro writers, but can also serve common TEX users very well.

11 Color support

Although implementing color in TEX can be done (quite easy), real color support should be a natural feature of TEX. A color support mechanism should more or less look like the font mechanism.

```
\color\mycolor=rgb <r> <g> <b>
\bgroup\mycolor some text \egroup
```

All other reasonable color systems should be supported as well as color separation, trapping and related features.

Due to the fact that most macro packages already have a `\color` command, more obscure names must be used.

12 Natural multi–column support

The exact specifications of this feature can be distilled from the needs of the available high–end macro packages. Balancing, bottom and top alignment models, grid snapping, insert support, mark, color, objects spanning multiple columns and everything else should be supported. The definitive functionality can best evolve from experimental primitives.

13 Grid snapping and spreads

This feature is well known and discussed at TUG98 with Peter. Some prototyping can be done in TEX. (At this moment I lack the time to do this.)

14 More extensive hyphenation support

First of all we need hyphenation pattern handling on a word by word basis. Other needs can partially be met by extending the TFM, VF and DVI format. Because more advanced DVI viewers are needed for this, some hyphenation and font specific topics are currently under discussion and construction in an special NTG task force. We will report on that later in time.

15 Text flow

When discussing extensions, Piet brought up the topic of text flow around (fixed) objects, especially paragraphs crossing the page boundary as used in hanging floats. This feature involves `\parshape` as well as `\hangindent`. One should be able to let TEX reset those at a pagebreak (which means recalculating the second half of the paragraph).

This possible lays some constraints on pagebreak optimization, but something is better than nothing.

16 Input filters

This topic is probably beyond e–TEX and is dealt with by the NTG task force. When suitable solutions are found, they will be presented to the e–TEXteam.

17 More control over whatsit

`\Mark`'s, `\special`'s and `\write`'s can interfere with the other objects of the vertical list. More control is needed, like ignoring them, that is: not letting them interfere with skipping and unskipping etc.

18 Booleans and reals

Everyone will understand these needs; `\dimen` precession will suffice.

19 More control over `\parkip`'s

TEX automatically adds `\parskip`'s, `\lineskip`'s and alike. More advanced spacing algoritms (macros) sometimes have to guess if TEX will add or has added such a skip. I won't go into details, but one should be able to test for such automatically added skips, penalties and whatever.

```
\ifparskipadded
\iflineskipadded
\if...added
```

There should also be another primitive

```
\noparskip
```

which can be considered a cousin of `\nointerlineskip`.

And how about a primitive that rolls up everything we sometimes don't want:

```
\removelast... % I've forgotten the official collective name
```

as well as:

```
\forcebaseline
```

This one explictly sets the last baseline to the natural depth.

20 A little sister for `\everypar`

We call her: `\everyendofpar`.

21 And a twin brother for `\aftergroup`

Named `\beforegroup` and to be expanded before `\egroup`.

22 A real inner test

We already discussed this problem with Philip and Peter. Some simple experiments show that `\ifinner` does not always report the state one expects (although it conforms the specifications). We therefore desperately want:

```
\ifinsidebox
```

23 A new `\every...`

How about `\everytoken` that just does what its name says: expanding its content before each following token (until the group ends).

24 When we don't want to `\leavehmode`

Sometimes we don't want to leave horizontal mode while TEX feels the opposite. Therefore we need:

```
\dontleavehmode
```

Which holds us in horizontal mode until the next implicit or explicit `\par`.

25 The fourth type of box

From earlier discussions, Philip probably understand what we mean: a `\vtopbox` (or `\vtox` with dual baselines (this would make life more easy, improve spacing in TEX much, and would have saved me many frustrating weeks of programming hacks).

26 Two new types of rules

When inserting rules, TEX sets the `\prevdepth` and does not obey the normal spacing rules. Therefore we need:

```
\xhrule
\xvrule
```

Both act like normal lines, that is, their baseline lays in the baseline grid.

27 Two dimensional tables

Piet can best explain this special wish. What we need is an easy way to let fot instance the first column entry span four rows. A sort of center in columns.

28 Optimizing paragraph setting

It is possible to influence the typesetting by setting `\emergencystretch` and other parameters. It is not particularly charming to set such parameters at the paragraph or whenever another layout is used, especially not in situations where TEX acts as back–end. So how about extending TEX's paragraph typesetting mechanism with something:

```
\parpass 1 {settings} 2 {settings} 3 {settings}
```

And let TEX go through this list when the default (document wide) settings end up in over– and underfull boxes.

29 appending to a `\csname`

When implementing conversion macros (or list macros or stacks or whatever) one sometimes end up with loops that do things like:

```
\edef\whatever{\whatever ...}
```

Of course one could use the magic `\edef`/`\aftergroup` trick, but this is not that fast as well as exhausts the stacks. So we need:

```
\appendcsname\whatever{...}
```

No expansion is needed; we can use `\expandafter` for that purpose.

More control over alignments

When implementing row/column/cell color support, auto spacing, table breaking and table optimizing mechanisms, one has to program quite tactfully to prevent problems with `\omit`'s and `\span`'s. Therefore we need:

```
\notalign{}
```

This primitive should not not interfere with alignments entries, etc.

Versatile rules

One nice feature of rules is that their width and height when not specified adapts to the current width or height. When implementing more advanced and fancy graphic features, one needs access to this mechanism. One way of doing this is:

```
\hspecial width <dim> height <dim> depth <dim> using \csname
\vspecial width <dim> height <dim> depth <dim> using \csname
```

I've got lots of applications for this, like alternative frames around tables, just one of those areas where TEX is way behind.

Here `\hspecial` acts like a rule (and therefore automatically sets the width or height when omitted), but is a last pass calls for `\csname`. The dimensions must come available in `\specialwidth`, `\specialheight` and `\specialdepth`. This enables for instance figures to be inserted instead of rules!

30 String handling

Being completely parameter driven, CONTEXT would benefit much from some simple string handling primitives (both Taco and I guess on about 50% gain in speed):

```
\loopoverlist\csname{separator}{list}
```

Like: `\loopoverlist\message,{aa,bbb,cc}`. More speed can be gained when we also have:

```
\loopoverexpandedlist\csname{separator}{list}
```

Two other welcome primitives are:

```
\ifsamestrings{}{}
\ifemptystring{}
```

These should fully expand their arguments; we're talking about a sort of fast:

```
\edef\first{#1}\edef\second{#2}\ifx\first\second
```

Some even more versatile companions are:

```
\ifsametokens{}{}
\ifemptytokens{}
```

These (probably slower) alternatives should accept:

```
\def\oeps#1{}
\ifsametokens{abc}{\oeps}          % false
\ifsametokens{abc}{abc\oeps{def}}  % true
```

I can come up with more of such primitives.

31 A tricky one

Why not implement a # mechanism that passes only pointers and does not copy the argument?

32 Again alignments

How nice it would be to be able to unwind alignments, just like boxes can be unwind.

33 Nop boxes

I implemented a visual debugger, but to do things really nice I need some nop boxes. Such boxes should be typeset and visible, but not interfere with the typesetting.

```
\vnop{content}
\hnop{content}
```

34 Error recovery

Sometimes TEX aborts or hangs on an error. For common users a more tolerant recovery is welcome. In general, much of TEX's log features can be improved, especially the formatting (to enable more advanced postprocessing of the log file).

35 That's it

I've quite certain forgotten some features. Let it be.