The ε-TEX manual

Version 2, February 1998

by The $\mathcal{N}\mathcal{T}\mathcal{S}$ Team

Peter Breitenlohner, Max-Planck-Institut für Physik, München

The preparation of this report was supported in part by DANTE,
Deutschsprachige Anwendervereinigung TEX e.V.

1 Introduction

The $\mathcal{N}\mathcal{T}\mathcal{S}$ project intends to develop an 'New Typesetting System' ($\mathcal{N}\mathcal{T}\mathcal{S}$) that will eventually replace today's TEX3. The $\mathcal{N}\mathcal{T}\mathcal{S}$ program will include many features missing in TEX, but there will also exist a mode of operation that is 100% compatible with TEX3. It will, necessarily, require quite some time to develop $\mathcal{N}\mathcal{T}\mathcal{S}$ to maturity and make it widely available.

Meanwhile ε-TEX intends to fill the gap between TEX3 and the future $\mathcal{N}\mathcal{T}\mathcal{S}$. It consists of a series of features extending the capabilities of TEX3.[1]

Since compatibility between ε-TEX and TEX3 has been a main concern, ε-TEX has two modes of operation:
(1) In TEX compatibility mode it fully deserves the name TEX and there are neither extended features nor additional primitive commands. That means in particular that ε-TEX passes the `TRIP` test [1] without any restriction. There are, however, a few minor modifications that would be legitimate in any implementation of TEX.
(2) In extended mode there are additional primitive commands and the extended features of ε-TEX are available.

We have tried to make ε-TEX as compatible with TEX as possible even in extended mode. In a few cases there are, however, some subtle differences described in detail later on. Therefore the ε-TEX features available in extended mode are grouped into two categories:
(1) Most of them have no semantic effect as long as none of the additional primitives are executed; these 'extensions' are permanently enabled.
(2) The remaining optional ε-TEX features ('enhancements') can be individually enabled and disabled; initially they are all disabled. For each enhancement there is a state variable `\...state`; an enhancement is enabled or disabled by assigning a positive or non-positive value respectively to that state variable.

For ε-TEX Versions 1 and 2 there is just one enhancement: mixed direction typesetting (TEX-**-XET**) with the state variable `\TeXXeTstate`.

[1]The TEX3 program; for the moment there are no plans to extend the software related to TEX.

Version 1.1 of ε-TEX was released in November 1996, Version 2.0 in February 1998. It is expected that there will be about one ε-TEX version per year, where each later version adds new features. It would be desirable if these ε-TEX versions were incorporated into many of the existing implementations of TEX3 without much delay.

With each ε-TEX version there will be an `e-TRIP` test [2] in order to help to verify that a particular implementation deserves the name ε-TEX in the same way as the `TRIP` test [1] helps to verify that an implementation deserves the name TEX.

2 Generating ε-TEX

2.1 Generating the ε-TEX Program

An implementation of TEX consists of a WEB change file `tex.ch` containing all system-dependent changes for a particular system. The WEB system program `TANGLE` applies this change file to the system-independent file `tex.web` defining the TEX program in order to generate a TEX Pascal file for that system [3]. Similarly an implementation of ε-TEX consists of a system-dependent change file `etex.sys` to be applied to the system-independent file `e-tex.web` defining the ε-TEX program. Since ε-TEX differs from TEX by a relatively small fraction of its code `e-tex.web` does, however, not exist as a physical file; it is instead defined in terms of a system-independent change file `e-tex.ch` to be applied to `tex.web`. Similarly it should be possible to define the system-dependent change file `etex.sys` for a particular system in terms of its deviations from the corresponding file `tex.ch` [4].

2.2 Generating Format Files for ε-TEX

When (the INITEX or VIRTEX version of) the TEX program is started, it analyzes the first non-blank input line from the command line or (with the `**` prompt) from the terminal: The first non-blank character of that input line may be an `&` followed immediately by the name of the format to be loaded; otherwise VIRTEX uses a default format whereas INITEX starts without loading a format file.

For eINITEX (the INITEX version of ε-TEX) there is an additional possibility: If the first non-blank input character is an `*` (immediately followed what would be the first non-blank input character for INITEX), the program starts in extended mode without loading a format file. If the first non-blank character is neither `&` nor `*` then eINITEX starts without loading a format but in compatibility mode. Whenever a format file is loaded by eINITEX or eVIRTEX the mode (compatibility or extended) is inherited from the format.

It is recommended that the input file `etex.src` be used instead of `plain.tex` when generating an ε-TEX format in extended mode. That file will first read `plain.tex` (without reading `hyphen.tex`) and will then supply macro definitions supporting ε-TEX features.

3 ε-TEX Extensions

3.1 Compatibility and Extended Mode

Once ε-TEX has entered compatibility mode it behaves as any other implementation of TEX. All of ε-TEX's additional commands are absent; it is therefore impossible to access any of the extensions or enhancements. The ability of eINITEX to initially choose between compatibility and extended mode is, however, by itself a feature not present in any TEX implementation.

The remainder of this document is devoted to a detailed and mostly technical description of all aspects where ε-TEX (in extended mode) behaves differently from TEX. It will be assumed that the reader is familiar with *The TEXbook* [5] describing TEX's behaviour in quite some detail.

All of ε-TEX's extensions and enhancements available in extended mode are activated by either executing some new primitive command or by assigning a nonzero value to some new integer parameter or state variable. Since all these new variables are initially zero,[2] ε-TEX behaves as TEX as long as none of ε-TEX's new control sequences are used, with the following exceptions which should, however, have no effect on the typesetting of error-free TEX documents (produced with error-free formats):
(1) When `\tracingcommands` has a value of 3 or more, or
when `\tracinglostchars` has a value of 2 or more, ε-TEX will display additional information not available in TEX.
(2) When using a count, dimen, skip, muskip, box, or token register number in the range 256–32767, ε-TEX will access one of its additional registers whereas TEX will produce an error and use register number zero.

3.2 Optimization

When a value is assigned to an ⟨internal quantity⟩ within a save group, the former value is restored when the group ends, provided the assignment was not global. This is achieved by saving the former value on TEX's 'save stack'. ε-TEX refrains from creating such save stack entries when the old and new value are the same ('reassignments').

`\aftergroup` tokens are also kept on TEX's save stack. When the current group ends, TEX converts each `\aftergroup` token into a token list and inserts this list as new 'input level' into the input stack. ε-TEX collects all `\aftergroup` tokens from one group into one token list and thus conserves input levels.

[2] To be precise all state variables are zero when eINITEX or eVIRTEX is started; integer parameters that are not state variables are zero when eINITEX is started without loading a format file or inherited from the format file otherwise.

When a completed page is written to the DVI file (shipped out), TeX multiplies the relevant stretch or shrink components of glue nodes in a box by the glue expansion factor of that box and converts the product to DVI units. In order to avoid overflow each resulting value x is artificially limited to the range $|x| \leq 10^9$. Consider the example:

```
\shipout\vbox to100pt{
  \hrule width10pt
  \vskip 0pt plus1000fil
  \vskip 0pt plus1000fil
  \vskip 0pt plus-2000fil
  \hrule
  \vskip 0pt plus0.00005fil
  }
```

Here the three glues between the two rules add up to zero; when TeX converts each stretch component individually they will, however, add up to 10^9 DVI units due to the truncation mentioned above. ε-TeX, however, accumulates the relevant stretch or shrink components of consecutive glue nodes (possibly separated by insert, mark, adjust, kern, and penalty nodes) before converting them to DVI units. During this process glue nodes may be converted into equivalent kern nodes and some glue specifications may be recycled; this may affect the memory usage statistics displayed after the page has been shipped out.

3.3 Tracing and Diagnostics

When `\tracingcommands` has a value of 3 or more, the commands following a prefix (`\global`, etc.) are shown as well, e.g.:

```
\global\count0=0    =>    {\global}
                          {\count}
```

When `\tracinglostchars` has a value of 2 or more, missing characters are displayed on the terminal even if the value of `\tracingonline` is 0 or less.

When `\tracingscantokens` has a value of 1 or more, the opening and closing of pseudo-files (generated by `\scantokens`) is recorded as for any other file, with '␣' as filename.

When the program is compiled with the code for collecting statistics and `\tracingassigns` has a value of 1 or more, all assignments subject to TeX's grouping mechanism are traced, e.g.:

```
\def\foo{\relax}    =>    {changing \foo=undefined}
                          {into \foo=macro:->\relax }
\global\count17=7   =>    {globally changing \count17=0}
                          {into \count17=7}
\count17=7          =>    {reassigning \count17=7}
```

When `\tracingifs` has a value of 1 or more, all conditionals (including `\unless`, `\or`, `\else`, and `\fi`) are traced, together with the starting line and nesting level; the `\showifs` command displays the state of all currently active conditionals. Thus the input

```
\unless\iffalse
   \iffalse
   \else
      \showifs
   \fi
\fi
```

might yield

```
{\unless\iffalse: (level 1) entered on line 1}
{\iffalse: (level 2) entered on line 2}
{\else: \iffalse (level 2) entered on line 2}
### level 2: \iffalse\else entered on line 2
### level 1: \unless\iffalse entered on line 1
{\fi: \iffalse (level 2) entered on line 2}
{\fi: \unless\iffalse (level 1) entered on line 1}
```

When `\tracinggroups` has a value of 1 or more, the start and end of each save group is traced, together with the starting line and grouping level; the `\showgroups` command displays the state of all currently active save groups. Thus the input

```
\begingroup
   {
      \showgroups
   }
\endgroup
```

might yield

```
{entering semi simple group (level 1) at line 1}
{entering simple group (level 2) at line 2}
### simple group (level 2) entered at line 1 ({)
### semi simple group (level 1) entered at line 1 (\begingroup)
### bottom level
{leaving simple group (level 2) entered at line 2}
{leaving semi simple group (level 1) entered at line 1}
```

Occasionally conditionals and/or save groups are not properly nested with respect to `\input` files. Although this might be perfectly legitimate, such anomalies are mostly unintentional and may cause quite obscure errors. When `\tracingnesting` has a value of 1 or more, these anomalies are shown; when `\tracingnesting` has a value of 2 or more, the current context (traceback) is shown as well. Thus the input

```
\newlinechar=`\^^J
\begingroup
   \iftrue
      \scantokens{%
   \endgroup
^^J\fi
^^J\bgroup
   ^^\tracingnesting=2
   ^^J\iffalse
   ^^J\else
     }%
  \egroup
\fi
```

might yield[3]

```
Warning: end of semi simple group (level 1) entered at line 2 of
 a different file
Warning: end of \iftrue entered on line 3 of a different file
Warning: end of file when simple group (level 1) entered at line
 3 is incomplete
Warning: end of file when \iffalse\else entered on line 5 is inc
omplete
l.7 \else

l.11      }
           %
```

The command `\showtokens{`⟨token list⟩`}` displays the token list, and allows the display of quantities that cannot be displayed by `\show` or `\showthe`, e.g.:

```
\showtokens\expandafter{\jobname}
\showtokens\expandafter{\topmarks 27}
```

3.4 Status Enquiries

A number of TEX's internal quantities can be assigned values but these values cannot be retrieved in TEX. ε-TEX introduces several new primitives that allow the retrieval of information about its internal state.
`\eTeXversion` returns ε-TEX's (major) version number;
`\eTeXrevision` expands into a list of character tokens representing the revision (minor version) number. Thus

```
\message{\number\eTeXversion\eTeXrevision}
```

should write the complete version as shown when ε-TEX is started.

[3]The `\scantokens` command will be discussed later.

When used as number, `\interactionmode` returns one of the values 0 (batchmode), 1 (nonstopmode), 2 (scrollmode), or 3 (errorstopmode). Assigning one of these values to `\interactionmode` changes the current interaction mode accordingly; such assignments are always global.
`\currentgrouplevel` returns the current save group level;
`\currentgrouptype` returns a number representing the type of the innermost group:

0:	bottom level (no group)	9:	math group
1:	simple group	10:	disc group
2:	hbox group	11:	insert group
3:	adjusted hbox group	12:	vcenter group
4:	vbox group	13:	math choice group
5:	vtop group	14:	semi simple group
6:	align group	15:	math shift group
7:	no align group	16:	math left group
8:	output group		

`\currentiflevel` returns the number of currently active conditionals;
`\currentifbranch` indicates which branch of the innermost conditional is taken: 1 'then branch', -1 'else branch', or 0 not yet decided;
`\currentiftype` returns 0 if there are no active conditionals, a positive number indicating the type of the innermost active conditional, or the negative of that number when the conditional was prefixed by `\unless`:

1:	`\if`	8:	`\ifmmode`	15:	`\iftrue`
2:	`\ifcat`	9:	`\ifinner`	16:	`\iffalse`
3:	`\ifnum`	10:	`\ifvoid`	17:	`\ifcase`
4:	`\ifdim`	11:	`\ifhbox`	18:	`\ifdefined`
5:	`\ifodd`	12:	`\ifvbox`	19:	`\ifcsname`
6:	`\ifvmode`	13:	`\ifx`	20:	`\iffontchar`
7:	`\ifhmode`	14:	`\ifeof`		

`\lastnodetype` returns a number indicating the type of the last node, if any, on the current (vertical, horizontal, or math) list:

-1:	none (empty list)	8:	disc node
0:	char node	9:	whatsit node
1:	hlist node	10:	math node
2:	vlist node	11:	glue node
3:	rule node	12:	kern node
4:	ins node	13:	penalty node
5:	mark node	14:	unset node
6:	adjust node	15:	math mode nodes
7:	ligature node		

The commands \fontcharht, \fontcharwd, \fontchardp, and \fontcharic followed by a font specification and a character code, return a dimension: the height, width, depth, or italic correction of the character in the font, or 0pt if no such character exists; the conditional \iffontchar tests the existence of that character.
When used as number, \parshape returns the number of lines of the current parshape specification (or zero).
ε-TeX's \parshapeindent, \parshapelength, and \parshapedimen, followed by a number n return the dimensions of the parshape specification:
0pt for $n \leq 0$ or when no parshape is currently active, otherwise
\parshapeindent n and \parshapedimen $2n-1$ both return the indentation of line n (explicitly specified or implied by repeating the last specification),
\parshapelength n and \parshapedimen $2n$ both return the length of line n.

3.5 Expressions

ε-TeX introduces the notion of expressions of type number, dimen, glue, or muglue, that can be used whenever a quantity of that type is needed. Such expressions are evaluated by ε-TeX's scanning mechanism; they are initiated by one of the commands \numexpr, \dimexpr, \glueexpr, or \muexpr (determining the type t) and optionally terminated by one \relax (that will be absorbed by the scanning mechanism). An expression consists of one or more terms of the same type to be added or subtracted; a term of type t consists of a factor of that type, optionally multiplied and/or divided by numeric factors; finally a factor of type t is either a parenthesized subexpression or a quantity (number, etc.) of that type. Thus, the conditional

```
\ifdim\dimexpr (2pt-5pt)*\numexpr 3-3*13/5\relax + 34pt/2<\wd20
```

is true if and only if the width of box 20 exceeds 32 pt. Note the use of \relax to terminate the inner (numeric) expression, the outer (dimen) expression is terminated automatically by the token $<_{12}$ that does not fit into the expression syntax.

The arithmetic performed by ε-TeX's expressions does not do much that could not be done by TeX's arithmetic operations \advance, \multiply, and \divide, although there are some notable differences: Each factor is checked to be in the allowed range, numbers must be less than 2^{31} in absolute value, dimensions or glue components must be less than 2^{14} pt, mu, fil, etc. respectively. The arithmetic operations are performed individually, except for 'scaling' operations (a multiplication immediately followed by a division) which are performed as one combined operation with a 64-bit product as intermediate value. The result of each operation is again checked to be in the allowed range. Finally the results of divisions and scalings are rounded, whereas TeX's \divide truncates.

The important new feature is, however, that the evaluation of expressions does not involve assignments and can therefore be performed in circumstances where assignments are not allowed, e.g., inside an `\edef` or `\write`. This also allows the definition of purely expandable loop constructions:

```
\def\foo#1#2{\number#1
  \ifnum#1<#2,
    \expandafter\foo
    \expandafter{\number\numexpr#1+1\expandafter}%
    \expandafter{\number#2\expandafter}%
  \fi}
```

such that, e.g., '`\foo{7}{13}`' expands into '`7, 8, 9, 10, 11, 12, 13`'.

The commands `\gluestretch` and `\glueshrink` are to be followed by a glue specification and return the stretch or shrink component of that glue as dimensions (with `fil` etc. replaced by `pt`), the commands `\gluestretchorder` and `\glueshrinkorder` return the order of infinity: 0 for `pt`, 1 for `fil`, 2 for `fill`, and 3 for `filll`.

The commands `\gluetomu` and `\mutoglue` convert glue into muglue and vice versa by simply equating 1 `pt` with 1 `mu`, precisely what TEX does (in addition to an error message) when the wrong kind of glue is used.

3.6 Additional Registers and Marks

ε-TEX increases the number of TEX's count, dimen, skip, muskip, box, and token registers from 256 to 32768. The additional registers, numbered 256–32767, can be used exactly as the first 256, except that they can not be used for insertion classes.

As in TEX, the first 256 registers of each kind are realized as static arrays that are part of the 'table of equivalents'; values to be restored when a save group ends are kept on the save stack. The additional registers are realized as sparse arrays built from TEX's main memory and are therefore less efficient. They use a four-level index structure and individual registers are present only when needed. Values to be restored when a particular save group ends are kept in a linked list (again built from main memory) with one save stack entry pointing to that list.[4]

ε-TEX generalizes TEX's mark concept to mark classes 0–32767, with mark class 0 used for TEX's marks.
The command `\marks` followed by a mark class n and a mark text appends a mark node to the current list; `\marks0` is synonymous with `\mark`. The page builder and the `\vsplit` command record information about the mark nodes found on the page or box produced, separately for each mark class. The information for mark class 0 is kept in a small static array as in TEX, the information for the additional mark classes is again kept in a sparse array with entries present only when needed.
The command `\firstmarks` n expands to the mark text for mark class n first

[4] With the effect that the order of restoring (or discarding) saved values may be somewhat surprising.

encountered on the most recent page, etc., and again `\firstmarks0` is synonymous with `\firstmark`.

3.7 Input Handling

The command `\readline`⟨number⟩ `to` ⟨control sequence⟩ defines the control sequence as parameterless macro whose replacement text is the contents of the next line read from the designated file, as for `\read`. The difference is that the current category codes are ignored and all characters on that line (including an endline character) are converted to character tokens with category 12 ('other'), except that the character code 32 gets category 10 ('space').

The command `\scantokens{...}` absorbs a list of unexpanded tokens, converts it into a character string that is treated as if it were an external file, and starts to read from this 'pseudo-file'. A rather similar effect can be achieved by the commands

```
\toks0={...}
\immediate\openout0=file
\immediate\write0{\the\toks0}
\immediate\closeout0
\input file
```

In particular every occurrence of the current newline character is interpreted as start of a new line, and input characters will be converted into tokens as usual. The `\scantokens` command is, however, expandable and does not use token registers, write streams, or external files. Furthermore the conversion from TeX's internal ASCII codes to external characters and back to ASCII codes is skipped. Finally the current context (traceback) shown, e.g., as part of an error message continues beyond an input line from a pseudo-file until an input line from a real file (or the terminal) is found.

When ε-TeX's input mechanism attempts to read beyond the end of an `\input` file or `\scantokens` pseudo-file, and before checking for 'runaway' conditions and closing the file, it will first read a list of tokens that has been predefined by the command `\everyeof={`⟨token list⟩`}`.

3.8 Breaking Paragraphs into Lines

Traditional typesetting with lead type used to adjust (stretch or shrink) the interword spaces in the last line of a paragraph by the same amount as those in the preceding line. With TeX the last line is, however, usually typeset at its natural width due to infinitely stretchable parfillskip glue. ε-TeX allows interpolation between these two extremes by specifying a suitable value for `\lastlinefit`. For a value of 0 or less, ε-TeX behaves as TeX, values from 1 to 1000 indicate a glue adjustment fraction f times 1000, values above 1000 are interpreted as $f = 1$.

The new algorithm is used only if

1. `\lastlinefit` is positive;

2. `\parfillskip` has infinite stretchability; and
3. the stretchability of `\leftskip` plus `\rightskip` is finite.[5]
Thus the last line of a paragraph would normally be typeset at its natural width and the stretchability of parfillskip glue would be used to achieve the desired line width. The algorithm proceeds as usual, considering all possible sequences of feasible break points and accumulating demerits for the stretching or shrinking of lines as well as for visually incompatible lines. When a candidate for the last line has been reached, the following conditions are tested:
4. the previous line was not 'infinitely bad' and was stretched with positive finite stretchability or was shrunk with positive shrinkability;
5. the last line has infinite stretchability entirely due to parfillskip glue;
6. if the previous line was stretched or shrunk the last line has positive finite stretchability or shrinkability respectively.
If all three conditions are satisfied, a glue adjustment factor of f times that of the preceding line will be applied to the relevant stretch or shrink components of all glue nodes in the last line, and the corresponding demerits are computed. (The last line will, however, not be stretched beyond the desired line width.)

When all possible candidates for the last line of the paragraph have been examined, the one having fewest accumulated demerits is chosen. If ε-TeX's modified algorithm was applied to that last line, the actual stretching or shrinking is achieved by suitably modifying the parfillskip glue node.

All computations described so far are performed with machine-independent integer arithmetic. Note, however, that the actual stretching requires machine-dependent floating point arithmetic. Therefore, when a paragraph is interrupted by a displayed equation and the line preceding the display is subject to the adjustment just described, the display will in general be preceded by abovedisplayskip and not by abovedisplayshortskip glue.

After breaking a paragraph into lines, TeX computes the interline penalties by adding the values of:
`\interlinepenalty` between any two lines,
`\clubpenalty` after the first line of a (partial) paragraph,
`\widowpenalty` before the last line of the paragraph,
`\displaywidowpenalty` before the line immediately preceding a displayed equation, and
`\brokenpenalty` after lines ending with a discretionary break.
ε-TeX generalizes the concept of interline, club, widow, and display widow penalty by allowing their replacement by arrays of penalty values with the commands
`\interlinepenalties`,
`\clubpenalties`,
`\widowpenalties`, and
`\displaywidowpenalties`.
Each of these commands is to be followed by an optional equal sign and a num-

[5] As usual for parameters influencing TeX's line-breaking algorithm, the values current at the end of the (partial) paragraph are used.

ber n. If $n \leq 0$ the respective array is reset and TEX's corresponding single value is used as usual; a positive value n declares an array of length n and must be followed by n penalty values. When one of these arrays has been set, its values are used instead of TEX's corresponding single values as follows (repeating the last value when necessary):
the i^{th} interline penalty value is used after line i of the paragraph;
the i^{th} club penalty value is used after line i of a partial paragraph;
the i^{th} widow penalty value is used after line $m - i$ of a paragraph without displayed equations or the last partial paragraph of length m;
the i^{th} display widow penalty value is used after line $m-i$ of a partial paragraph of length m that is followed by a displayed equation.

When used after `\the` or in situations where TEX expects to see a number, the same four commands serve to retrieve the arrays of penalties. Specifying, e.g., `\clubpenalties`⟨number⟩ with a number n, returns 0 for $n < 0$ or when the club penalty array has been reset, the length of the declared club penalty array for $n = 0$, or the n^{th} club penalty value for $n > 0$ (again repeating the last value when necessary).

3.9 Math Formulas

TEX's `\left`⟨delimiter⟩`...\right`⟨delimiter⟩ produces two delimiters with a common size adjusted to the height and depth of the enclosed material. In ε-TEX this can be generalized by occurrences of `\middle`⟨delimiter⟩ dividing the enclosed material into segments resulting in a sequence of delimiters with a common size adjusted to the maximal height and depth of all enclosed segments. The spacing between a segment and the delimiter to its left or right is as for TEX's left or right delimiter respectively.

3.10 Hyphenation

TEX uses the `\lccode` values for two quite unrelated purposes:
(1) when `\lowercase` converts character tokens to their lower-case equivalents (in the same way as `\uppercase` uses the `\uccode` values); and
(2) when hyphenation patterns or exceptions are read, and when words are hyphenated during the line-breaking algorithm.

ε-TEX introduces the concept of (language-dependent) hyphenation codes that are used instead of the `\lccode` values for hyphenation purposes. In order to explain the details of ε-TEX's behaviour, we need some technical aspects of hyphenation patterns. When INITEX starts without reading a format file, the (initially empty) hyphenation patterns are in a form suitable for inserting new patterns specified by `\patterns` commands; when INITEX attemps hyphenation or prepares to write a format file, they are compressed into a more compact form suitable for finding hyphens. Only these compressed patterns can be read from a format file (by INITEX or VIRTEX).

In ε-TEX the hyphenation patterns are supplemented by hyphenation codes. When eINITEX starts without reading a format file both are initially empty;

when a `\patterns` command is executed and `\savinghyphcodes` has a positive value, the current `\lccode` values are saved as hyphenation codes for the current language. These saved hyphenation codes are later compressed together with the patterns and written to or read from a format file. When the patterns have been compressed (always true for eVIRTEX) and hyphenation codes have been saved for the current language, they are used instead of the `\lccode` values for hyphenation purposes (reading hyphenation exceptions and hyphenating words).

3.11 Discarded Items

When TeX's page builder transfers (vertical mode) material from the 'recent contributions' to the 'page so far', it discards glue, kern, and penalty nodes (discardable items) preceding the first box or rule on the page under construction and inserts a topskip glue node immediately before that box or rule. Note, however, that this topskip glue need not be the first node on the page, it may be preceded by insertion, mark, and whatsit nodes. Similarly when the `\vsplit` command has split the first part off a vbox, discardable items are discarded from the top of the remaining vbox and a splittopskip glue node is inserted immediately before the first box or rule.

When ε-TeX's parameter `\savingvdiscards` has been assigned a positive value, these 'discarded items' are saved in two lists and can be recovered by the commands `\pagediscards` and `\splitdiscards` that act like 'unvboxing' hypothetical box registers containing a vbox with the discarded items.

The list of items discarded by the page builder is emptied at the end of the output routine and by the `\pagediscards` command; new items may be added as long as the new 'page so far' contains no box or rule.

The list of items discarded by the `\vsplit` command is emptied at the start of a vsplit operation and by the `\splitdiscards` command; new items are added at the end of a vsplit operation.

3.12 Expandable Commands

Chapter 20 of *The TeXbook* gives complete lists of all expandable TeX commands and of all cases where expandable tokens are not expanded. For ε-TeX there are these additional conditionals:

- `\ifdefined`⟨token⟩   (test if token is defined)

True if ⟨token⟩ is defined; creates no new hash table entry.

- `\ifcsname...\endcsname`   (test if control sequence is defined)

True if the control sequence `\csname...\endcsname` would be defined; creates no new hash table entry.

- `\iffontchar`⟨font⟩⟨8-bit number⟩   (test if char exists)

True if \char⟨8-bit number⟩ in \font⟨font⟩ exists.

These are ε-TeX's additional expandable commands:

- \unless.
  The next (unexpanded) token must be a boolean conditional (i.e., not \ifcase); the truth value of that conditional is reversed.

- \eTeXrevision.
  The expansion is a list of character tokens of category 12 ('other') representing ε-TeX's revision (minor version) number, e.g., '.0' or '.1'.

- \topmarks⟨15-bit number⟩, \firstmarks⟨15-bit number⟩, \botmarks⟨15-bit number⟩, \splitfirstmarks⟨15-bit number⟩, and \splitbotmarks⟨15-bit number⟩.
  These commands generalize TeX's \topmark etc. to 32768 distinct mark classes; the special case \topmarks0 is synonymous with \topmark etc.

- \unexpanded⟨general text⟩.
  The expansion is the token list ⟨balanced text⟩.

- \detokenize⟨general text⟩.
  The expansion is a list of character tokens representing the token list ⟨balanced text⟩. As with the lists of character tokens produced by TeX's \the and ε-TeX's \readline, these tokens have category 12 ('other'), except that the character code 32 gets category 10 ('space').

- \scantokens⟨general text⟩.
  The expansion is null; but ε-TeX creates a pseudo-file containing the characters representing the token list ⟨balanced text⟩ and prepares to read from this pseudo-file before looking at any more tokens from its current source.

These are the additional ε-TeX cases when expandable tokens are not expanded:

- When ε-TeX is reading the argument token for \ifdefined.

- When ε-TeX is absorbing the token list for \unexpanded, \detokenize, \scantokens, or \showtokens.

- Protected macros (defined with the \protected prefix) are not expanded when building an expanded token list (for \edef, \xdef, \message, \errmessage, \special, \mark, \marks or when writing the token list for \write to a file) or when looking ahead in an alignment for \noalign or \omit.[6]

[6] Whereas protected macros were introduced with ε-TeX Version 1, suppression of their expansion in alignments was introduced with Version 2.

- When building an expanded token list, the tokens resulting from the expansion of `\unexpanded` are not expanded further (this is the same behaviour as is exhibited by the tokens resulting from the expansion of `\the`⟨token variable⟩ in both TEX and ε-TEX).

4 ε-TEX Enhancements

The execution of most new primitives related to enhancements is disallowed when the corresponding enhancement is currently disabled and will lead to an '`Improper...`' error message. The offending command may nevertheless already have had some effect such as, e.g., bringing ε-TEX into horizontal mode.

4.1 Mixed-Direction Typesetting

This feature supports mixed left-to-right and right-to-left typesetting and introduces the four text-direction primitives `\beginL`, `\endL`, `\beginR`, and `\endR`. The code is inspired by but different from TEX-XET [6].

In order to avoid confusion with TEX-XET the present implementation of mixed-direction typesetting is called TEX--XET. It uses the same text-direction primitives, but differs from TEX-XET in several important aspects:
(1) Right-to-left text is reversed explicitly by ε-TEX and is written to a normal DVI file without any `begin_reflect` or `end_reflect` commands;
(2) a math node is (ab)used instead of a whatsit node to record the text-direction primitives in order to minimize the influence on the line-breaking algorithm for pure left-to-right text;
(3) right-to-left text interrupted by a displayed equation is automatically resumed after that equation;
(4) display math material is always printed left-to-right, even in constructions such as:

```
\hbox{\beginR\vbox{\noindent$$abc\eqno(123)$$}\endR}
```

TEX--XET is enabled or disabled by assigning a positive or non-positive value respectively to the `\TeXXeTstate` state variable. As long as TEX--XET is disabled, ε-TEX and TEX3 build horizontal lists and paragraphs in exactly the same way. Even TEX--XET will, in general, produce the same results as TEX3 for pure left-to-right text. There are, however, circumstances where some differences may arise. This is best illustrated by an example:

```
\vbox{\noindent
   $\hfil\break
   \null\hfil\break
   \null$\par
```

Here TEX will produce three lines containing the following nodes:
1. mathon, hfil glue, break penalty, and rightskip glue;
2. empty hbox, hfil glue, break penalty, and rightskip glue;

3. empty hbox, mathoff, nobreak penalty, parfillskip glue, and rightskip glue. These lines can be retrieved via:

```
\setbox3=\lastbox
\unskip\unpenalty
\setbox2=\lastbox
\unskip\unpenalty
\setbox1=\lastbox
```

Later on these lines can be 'unhboxed' as part of a new paragraph and possibly their contents analyzed. As a consequence in TEX (and ε-TEX in compatibility mode) there may be horizontal lists where mathon and mathoff nodes are not properly paired. Therefore TEX might attempt hyphenation of 'words' originating from math mode or prevent hyphenation of words originating from horizontal mode.

Math-mode material is always typeset left-to-right by TEX--XET, even when it is contained inside right-to-left text. Therefore TEX--XET will insert additional `beginM` and `endM` math nodes such that material originating from math mode is always enclosed between properly paired math nodes. Consequently TEX--XET will never attempt hyphenation of 'words' originating from math mode nor prevent hyphenation of words originating from horizontal mode.

The additional math nodes introduced by TEX--XET are, however, transparent to operations such as `\lastpenalty` that inspect or remove the last node of a horizontal list.[7]

When TEX--XET is enabled or disabled during the construction of a box, that box may contain text-direction directives or math nodes that are not properly paired. Such unpaired nodes may cause warning messages when the box is shipped out. It is, therefore, advisable that TEX--XET be enabled or disabled only in vertical mode.

5 Syntax Extensions for ε-TEX

5.1 Mode-Independent Commands

The syntax for TEX's mode-independent commands, as described in the first part of Chapter 24 of *The TEXbook*, is extended by modifications of existing commands as well as by new commands.

First, ε-TEX has 32768 `\count`, `\dimen`, `\skip`, `\muskip`, `\box`, and `\toks` registers instead of TEX's 256. Thus it allows a ⟨15-bit number⟩ instead of an ⟨8-bit number⟩ in almost all syntax constructions referring to these registers; the only exception to this is the `\insert` command: insertion classes are restricted to the range 0–254 in ε-TEX as they are in TEX.

Next, ε-TEX extends the list of TEX's internal quantities:

⟨internal integer⟩ ⟶ whatever *The TEXbook* defines | `\eTeXversion`
  | `\interactionmode` | ⟨penalties⟩⟨number⟩

[7] This was not the case for some earlier TEX--XET implementations.

| \lastnodetype | \currentgrouplevel | \currentgrouptype
| \currentiflevel | \currentiftype | \currentifbranch
| \gluestretchorder⟨glue⟩ | \glueshrinkorder⟨glue⟩
| \numexpr⟨integer expr⟩⟨optional spaces and \relax⟩

⟨penalties⟩ ⟶ \interlinepenalties | \clubpenalties
| \widowpenalties | \displaywidowpenalties

⟨internal dimen⟩ ⟶ whatever *The TEXbook* defines
| \parshapeindent⟨number⟩ | \parshapelength⟨number⟩
| \parshapedimen⟨number⟩
| \gluestretch⟨glue⟩ | \glueshrink⟨glue⟩
| \fontcharht⟨font⟩⟨8-bit number⟩ | \fontcharwd⟨font⟩⟨8-bit number⟩
| \fontchardp⟨font⟩⟨8-bit number⟩ | \fontcharic⟨font⟩⟨8-bit number⟩
| \dimexpr⟨dimen expr⟩⟨optional spaces and \relax⟩

⟨internal glue⟩ ⟶ whatever *The TEXbook* defines | \mutoglue⟨muglue⟩
| \glueexpr⟨glue expr⟩⟨optional spaces and \relax⟩

⟨internal muglue⟩ ⟶ whatever *The TEXbook* defines | \gluetomu⟨glue⟩
| \muexpr⟨muglue expr⟩⟨optional spaces and \relax⟩

The additional possibilities for ⟨integer parameter⟩ are:

\TeXXeTstate (positive if mixed-direction typesetting is enabled)
\tracingassigns (positive if showing assignments)
\tracinggroups (positive if showing save groups)
\tracingifs (positive if showing conditionals)
\tracingscantokens (positive if showing the opening and closing of \scantokens pseudo-files)
\tracingnesting (positive if showing improper nesting of groups and conditionals within files)
\predisplaydirection (text direction preceding a display)
\lastlinefit (adjustment ratio for last line of paragraph, times 1000)
\savingvdiscards (positive if saving items discarded from vertical lists)
\savinghyphcodes (positive if \patterns saves \lccode values as hyphenation codes)

Note that the ε-TEX state variable \TeXXeTstate (the only one so far) is an ⟨integer parameter⟩. That need not be the case for all future state variables; it might turn out that some future enhancements can be enabled and disabled only globally, not subject to grouping.

The additional possibilities for ⟨token parameter⟩ are:

\everyeof (tokens to insert when an \input file ends)

Here is the syntax for ε-TEX's expressions:

⟨integer expr⟩ ⟶ ⟨integer term⟩
| ⟨integer expr⟩⟨add or sub⟩⟨integer term⟩

⟨integer term⟩ ⟶ ⟨integer factor⟩
| ⟨integer term⟩⟨mul or div⟩⟨integer factor⟩

⟨integer factor⟩ ⟶ ⟨number⟩
| ⟨left paren⟩⟨integer expr⟩⟨right paren⟩

⟨dimen expr⟩ ⟶ ⟨dimen term⟩
    | ⟨dimen expr⟩⟨add or sub⟩⟨dimen term⟩
⟨dimen term⟩ ⟶ ⟨dimen factor⟩
    | ⟨dimen term⟩⟨mul or div⟩⟨integer factor⟩
⟨dimen factor⟩ ⟶ ⟨dimen⟩
    | ⟨left paren⟩⟨dimen expr⟩⟨right paren⟩
⟨glue expr⟩ ⟶ ⟨glue term⟩
    | ⟨glue expr⟩⟨add or sub⟩⟨glue term⟩
⟨glue term⟩ ⟶ ⟨glue factor⟩
    | ⟨glue term⟩⟨mul or div⟩⟨integer factor⟩
⟨glue factor⟩ ⟶ ⟨glue⟩
    | ⟨left paren⟩⟨glue expr⟩⟨right paren⟩
⟨muglue expr⟩ ⟶ ⟨muglue term⟩
    | ⟨muglue expr⟩⟨add or sub⟩⟨muglue term⟩
⟨muglue term⟩ ⟶ ⟨muglue factor⟩
    | ⟨muglue term⟩⟨mul or div⟩⟨integer factor⟩
⟨muglue factor⟩ ⟶ ⟨muglue⟩
    | ⟨left paren⟩⟨muglue expr⟩⟨right paren⟩
⟨optional spaces and `\relax`⟩ ⟶ ⟨optional spaces⟩
    | ⟨optional spaces⟩`\relax`
⟨add or sub⟩ ⟶ ⟨optional spaces⟩`+`$_{12}$ | ⟨optional spaces⟩`-`$_{12}$
⟨div or mul⟩ ⟶ ⟨optional spaces⟩`*`$_{12}$ | ⟨optional spaces⟩`/`$_{12}$
⟨left paren⟩ ⟶ ⟨optional spaces⟩`(`$_{12}$
⟨right paren⟩ ⟶ ⟨optional spaces⟩`)`$_{12}$

Next, ε-TeX extends the syntax for assignments:

⟨prefix⟩ ⟶ whatever *The TeXbook* defines | `\protected`
⟨simple assignment⟩ ⟶ whatever *The TeXbook* defines
    | ⟨penalties assignment⟩
    | `\readline`⟨number⟩ `to` ⟨control sequence⟩
⟨penalties assignment⟩ ⟶ ⟨penalties⟩⟨equals⟩⟨number⟩⟨penalty values⟩
⟨interaction mode assignment⟩ ⟶ whatever *The TeXbook* defines
    | `\interactionmode`⟨equals⟩⟨2-bit number⟩

In a ⟨penalties assignment⟩ for which the ⟨number⟩ is n, the ⟨penalty values⟩ are ⟨empty⟩ if $n \leq 0$, otherwise they consist of n consecutive occurrences of ⟨number⟩.

Finally, the remaining mode-independent ε-TeX commands:

- `\showgroups`, `\showifs`, `\showtokens`⟨general text⟩. These commands are intended to help you figure out what ε-TeX thinks it is doing. The `\showtokens` command displays the token list ⟨balanced text⟩.
- `\marks`⟨15-bit number⟩⟨general text⟩. This command generalizes TeX's `\mark` command to 32768 distinct mark classes; the special case `\marks0` is synonymous with `\mark`.

5.2 Vertical-Mode Commands

The syntax for TEX's vertical-mode commands, as described in the second part of Chapter 24 of *The TEXbook*, is extended by ε-TEX as follows:

- `\pagediscards`, `\splitdiscards`. These two commands are similar to `\unvbox`. When `\savingvdiscards` is positive, items discarded by the page builder and by the `\vsplit` command are collected in two special lists. One of these special lists is appended to the current vertical list (in the same way as `\unvbox` appends the vertical list inside a vbox) and becomes empty.

- Here are the additional possibilities for ⟨horizontal command⟩:

⟨horizontal command⟩ ⟶ whatever *The TEXbook* defines
| `\beginL` | `\endL` | `\beginR` | `\endR`

5.3 Horizontal-Mode Commands

The syntax for TEX's horizontal-mode commands, as described in Chapter 25 of *The TEXbook*, is extended by ε-TEX as follows:

- Here are the additional possibilities for ⟨vertical command⟩:

⟨vertical command⟩ ⟶ whatever *The TEXbook* defines
| `\pagediscards` | `\splitdiscards`

- `\beginL`, `\endL`, `\beginR`, `\endR` (text-direction commands).
  The use of these commands is illegal when the TEX--XET enhancement is currently disabled; otherwise a `beginL`, etc. text-direction node (a new kind of math node) is appended to the current horizontal list. These nodes delimit the beginning and end of hlist segments containing left-to-right (L) or right-to-left (R) text. Before a paragraph is broken into lines, `endL` and `endR` nodes are added to terminate any unfinished L or R segments; when a paragraph is continued after display math mode, any such unfinished segments are automatically resumed, starting the new hlist with `beginL` and `beginR` nodes as necessary.

- `\marks`⟨15-bit number⟩⟨general text⟩. This command generalizes TEX's `\mark` command to 32768 distinct mark classes; the special case `\marks0` is synonymous with `\mark`.

5.4 Math-Mode Commands

The syntax for TEX's math-mode commands, as described in Chapter 26 of *The TEXbook*, is extended by ε-TEX as follows:

- `\left`⟨delim⟩⟨math mode material⟩
  `\middle`⟨delim⟩⟨math mode material⟩`...\right`⟨delim⟩
  (generalizing TEX's `\left`⟨delim⟩⟨math mode material⟩`\right`⟨delim⟩).
  For each ⟨math mode material⟩ ε-TEX begins a new group, starting out with a new math list (always in the same style) that begins with a left boundary item containing everything processed so far. This group must be terminated with either '`\middle`' or '`right`', at which time the internal math list is completed with a new boundary item containing the new delimiter. In the case of '`\middle`', a new group is started again, in the case of '`\right`', ε-TEX appends an Inner atom to the current list; the nucleus of this atom contains the internal math list just completed.

References

[1] *A torture test for TEX*, by Donald E. Knuth, Stanford Computer Science Report 1027.

[2] *A torture test for ε-TEX*, by The $\mathcal{NTS}$ Team (Peter Breitenlohner and Bernd Raichle). Version 2, January 1998.

[3] *The WEB system of structured documentation*, by Donald E. Knuth, Stanford Computer Science Report 980.

[4] *How to generate ε-TEX*, by The $\mathcal{NTS}$ Team (Peter Breitenlohner and Phil Taylor). Version 2, January 1998.

[5] *The TEXbook* (Computers and Typesetting, Vol. A), by Donald E. Knuth, Addison Wesley, Reading, Massachusetts, 1986.

[6] *Mixing right-to-left texts with left-to-right texts*, by Donald E. Knuth and Pierre MacKay, *TUGboat* **8**, 14–25, 1987.