The `ltpara.dtx` code*

Frank Mittelbach

October 22, 2025

Abstract

This code defines four special kernel hooks to support paragraph tagging as well as four public hooks which can be occasionally useful.

1 Introduction

The building of paragraphs in the TEX engine(s) has a number of peculiarities that makes it on one hand fairly flexible but on the other hand somewhat awkward to control or reliably to extend. Thus to better understand the code below we start with a brief introduction of the mechanism; for more details refer to the TEXbook [**?**, chap. 14] (for the full truth you may even have to study the program code).

1.1 The default processing done by the engine

TEX automatically starts building a paragraph when it is currently in vertical mode and encounters anything that can only live in horizontal mode. Most often this is a character, but there are also many commands that can be used only in horizontal mode. If any of them is encountered, TEX will immediately back up (i.e., the character or command is read later again), adds a `\parskip` glue to the current vertical list unless the list is empty, switches to horizontal mode, starts its special "start of paragraph processing" and only then rereads the character or command that caused the mode change.[1]

This "start of paragraph processing" first adds an empty box at the start of the horizontal list of width `\parindent` (which represents the paragraph indentation) unless the paragraph was started with `\noindent` in which case no such box is added[2]. It then reads and processes all tokens stored in the special engine token register `\everypar`. After that it reads and processes whatever has caused the paragraph to start.

Thus out of the box, TEX offers the possibility to put some special code into `\everypar` to gain control at (more or less) the start of the paragraph. For example, in LaTeX and a number of packages, special code like the following is sometimes used:

```
\everypar{{\setbox\z@\lastbox}\everypar{} ...}
```

*This file has version v1.0n dated 2025/02/07, © LATEX Project.

[1] Already not quite true: the command `\noindent` starts the paragraph but influences the special processing by suppressing the paragraph indentation box normally inserted by it.

[2] That's a bit different from placing a zero-sized box!

This removes the paragraph indentation box again (that was already placed by TEX), then resets `\everypar` so that it doesn't do anything on the next paragraph start and then does whatever it wants to do, e.g., in an `\item` of a list it will typeset the label in front of the paragraph text. However, there is only one such `\everypar` token register and if different packages and/or the kernel all attempt to add their own code here, coordination is very difficult if not impossible.

The process when the paragraph ends has different mechanisms and interfaces. A paragraph ends when the engine primitive `\par` is called while TEX is in unrestricted horizontal mode, i.e., is building a paragraph. At other times this primitive does nothing or generates as an error depending on the mode TEX is in, e.g., the `\par` in `\hbox{a\par b}` is ignored, but `$a\par b$` would complain.

If this primitive ends the paragraph it does some special "end of horizontal list" processing, then calls TEX's paragraph builder; this breaks the horizontal list into lines and then these lines are added as boxes to the enclosing vertical list and TEX returns to vertical mode.

This `\par` command can be given explicitly, but there are also situations in which TEX is generating it on the fly. Most often this happens when TEX encounters a blank line which is automatically changed to a `\par` command which is then executed. The other possibility is that TEX encounters a command which is incompatible with horizontal processing, e.g., `\vskip` (a request for adding vertical space). In such cases it silently backs up, and inserts a `\par` in the hope that this gets it out of horizontal mode and makes the vertical command acceptable.

The important point to note here is that TEX really inserts the command with the name `\par`, which can be redefined. Thus, it may not have its original "primitive" meaning and therefore may not end the horizontal list and call the paragraph builder. This approach offers some flexibility but also allows you to easily produce a TEX document that loops forever, for example, the simple line

```
A \let\par\relax \vskip
```

will start a horizontal list at `A`, redefines `\par`, then sees `\vskip` and inserts `\par` to end the paragraph. But this now only runs `\relax` so nothing changes and `\vskip` is read again, issues a `\par` which .... In short, it only takes a plain TEX document with five tokens to run forever (since no memory is consumed and therefore eventually exhausted).

There is no way other than changing `\par` to gain control at the end of a paragraph, i.e., there is no token list like `\everypar` that is inserted. Hence the only way to change the default behavior is to modify the action that `\par` executes, with similar issues as outlined before: different processes need to ensure that they do not overwrite their modifications or worse, think that the `\par` in front of them is the engine primitive while in fact it has already been changed by other code.

To make matters slightly worse there are a few places where TEX handles the situation differently (most likely for speed reasons back when computers were much slower). If TEX finds itself in unrestricted horizontal mode at the end of building a vertical box (for an `\insert`, `\vadjust` or executing the output routine code), it will finish the horizontal list not by issuing a `\par` command (which would be consistent with all other places) but by simply executing the primitive meaning of `\par`, regardless of the actual definition that `\par` has at the time.

Thus, if you have carefully crafted a redefined `\par` to execute some special actions at the end of a paragraph and you write something like

```
\vbox{Some paragraph ... text.}
```

you will find that your code does not get run for the last paragraph in that box. LaTeX avoids this problem, by making sure that its boxes (such as `\parbox` or the `minipage` environment, etc.) all internally add an explicit `\par` at the end so that such code is run and TeX finds itself in vertical mode already without the need to start up the paragraph builder internally. But, of course, this only works for boxes under direct control of the LaTeX kernel; if some package uses low-level `\vbox`es without adding this precaution the TeX optimization kicks in and no special `\par` code is executed.

And there is another optimization that is painful: if a paragraph is interrupted by a mathematical display, e.g., `\[...\]` in LaTeX or `$$...$$` in plain TeX, then TeX will resume horizontal mode afterward, i.e., it will start to build a new horizontal list without inserting an indentation box or `\everypar` at that point. However, if that list immediately ends with an explicit or implicit `\par` then TeX will simply throw away this "null" paragraph and not do its usual "end of horizontal list" processing, so this special case also needs to be accounted for when introducing any extended processing.

2 The new mechanism implemented for LaTeX

To improve the situation (and also to support automatic tagging of PDF documents) we now offer public as well as private hooks at the start and end of the paragraph processing. The public hooks can be used by packages (or by the user in the preamble or within the document) and using the hook mechanisms it is possible to reorder or arrange code from different packages in such a way that these can safely coexist.

To make that happen we have to make use of the basic functionality that is offered by TeX, e.g., we install special code inside `\everypar` to provide hooks at the beginning and we redefine `\par` to do some special processing when appropriate to install hooks at the end of the paragraph.

In order to make this work, we have to ensure that package use of `\everypar` is not overwriting our code. This is done through a trick: we basically hide the real `\everypar` from the packages and offer them a new token register (with the same name). So if they install their own code it doesn't overwrite ours. Our code then inserts the new `\everypar` at the right place inside the process so that it looks as if it was the primitive `\everypar`.[3]

At the end of the paragraph it would be great if we could use a similar trick. However, due to the fact that TeX inserts the token `\par` (that doesn't have a defined meaning) we can't hide "the real thing™" and offer the package an indistinguishable alternate.

Fortunately, LaTeX has already redefined `\par` for its own purposes. As a result there aren't many packages that attempt to change `\par`, because without a lot of extra care that would fail miserably. But the bottom line is that, if you load a package that alters `\par` then the end of paragraph hooks are most likely not executing while that redefinition is active.[4]

[3] Ideally, `\everypar` wouldn't be used at all by packages and instead they would simply write their code into the hooks now offered by the kernel. However, while this is the longterm goal and clearly an improvement (because then the packages do no longer need to worry about getting their code overwritten or needing to account for already existing code in `\everypar`), this will not happen overnight. For that reason support for this legacy method is retained.

[4] Similarly to the `\everypar` situation, the remedy is that such packages stop doing this and instead add their alterations into the paragraph hooks now provided.

2.1 The provided hooks

para/before
para/begin
para/end
para/after

The following four public hooks are defined and executed for each paragraph:

para/before This hook is executed after the kernel hook `\@kernel@before@para@before` (discussed below) in vertical mode immediately after TEX has contributed `\parskip` to the vertical list and before the actual paragraph processing in horizontal mode starts.

This hook should either not produce any typeset material or add only vertical material. If it starts a paragraph an error is generated. The reason is that we are in the starting process of processing a paragraph and so this would lead to endless recursion.[5]

para/begin This hook is executed after the kernel hook `\@kernel@before@para@begin` (discussed below) in horizontal mode immediately before the indentation box is placed (if there is any, i.e., if the paragraph hasn't been started with `\noindent`).

The indentation box to be typeset is available to the hook as `\IndentBox` and its automatic placement (after the hook is executed) can be prevented through `\OmitIndent`. More precisely `\OmitIndent` voids the box.

The indentation box is then typeset directly after the hook execution by something equivalent to `\box\IndentBox` followed by the current content of the token register `\everypar` that it is available to the kernel or to packages (that run some legacy code).

One has to be careful not to add any code to the hook that starts its own paragraph (e.g., by adding a `\parbox` or a `\marginpar` inside) because that would call the hook inside again (as a new paragraph is started there) and thus lead to an endless recursion ending only after exhausting the available memory. This can only be done by making sure that is not executed for the inner paragraphs (or at least not recursively forever).

para/end This hook is executed at the end of a paragraph when TEX is ready to return to vertical mode and after it has removed the last horizontal glue (but not any kerns) placed on the horizontal list. The code is still executed in horizontal mode so it is possible to add further horizontal material at this point, but it should not alter the mode (even a temporary exit from horizontal mode would create chaos—any attempt will cause an error message)! After the hook has ended the kernel hook `\@kernel@after@para@end` is executed and then TEX returns to vertical mode.

The hook is offered as public hook, but because of the requirement to stay within horizontal mode one needs to be careful in what is placed into the hook.[6]

This hook is implemented as a reversed hook.

para/after This hook is executed directly after TEX has returned to vertical mode and after any material that migrated out of the horizontal list (e.g., from a `\vadjust`) has processed.

[5] One could allow it but only if the newly started paragraph is processed without any hooks. Furthermore correct spacing would be a bit of a nightmare so for now this is forbidden.

[6] Maybe we should guard against that, but it would be rather tricky to implement as mode changes can happen across group boundaries so one would need to keep a private stack just for that. Well, something to ponder.

This hook should either not produce any typeset material or add only vertical material. However, for this hook starting a new paragraph is not a disaster so that it isn't prevented.

This hook is implemented as a reversed hook.

Once that hook code has been processed the kernel hook `\@kernel@after@para@after` is executed as the final action of the paragraph processing.

`\@kernel@before@para@before`
`\@kernel@after@para@after`
`\@kernel@before@para@begin`
`\@kernel@after@para@end`

As already mentioned above there are also four kernel hooks that are executed at the start and end of the processing.

`\@kernel@before@para@before` For future extensions, not currently used by the kernel.

`\@kernel@after@para@after` For future extensions, not currently used by the kernel.

`\@kernel@before@para@begin` Used by the kernel to implement tagging. This hook is executed at the very beginning of a paragraph after TEX has switched to horizontal mode but before any indentation box got added or any `\everypar` was run.

It should not generate typeset material that could alter the position. Note that it should never leave hmode, otherwise you will end with a loop! We could guard against this, but since it is an internal kernel hook that shouldn't be touched this isn't checked.

`\@kernel@after@para@end` Used by the kernel to implement tagging. It is executed directly after the public `para/end` hook. After it there is a quick check that we are still in horizontal mode, i.e., that the public hook has not mistakenly ended horizontal mode prematurely (this is an incomplete check just testing the mode and could perhaps be improved (at the cost of speed)).

2.2 Altered and newly provided commands

`\par`
`\endgraf`
`\para_end:`

An explicit request for ending a paragraph is provided in plain TEX under the name `\endgraf`, which simply uses the primitive meaning (regardless of what `\par` may have as its current definition). In LATEX `\endgraf` (with that behavior) was originally also available.

With the new paragraph handling in LATEX, ending a paragraph means a bit more than just calling the engine's paragraph builder: the process also has to add any hook code for the end of a paragraph. Thus `\endgraf` was changed to provide this additional functionality (along with `\par` remaining subject to its current meaning).

The expl3 name for this functionality is `\para_end:`.

Note: *The next two commands are still under discussion and may slightly change their semantics (as described in the document) and/or their names between now and the 2021 Spring release!*

\OmitIndent
\para_omit_indent:

Inside the para/begin hook one can use this command to suppress the indentation box at the start of the paragraph. (Technically it is possible to use this command outside the hook as well, but this should not be relied upon.) The box itself remains available for use.

The expl3 name for the function is \para_omit_indent:.

\IndentBox
\g_para_indent_box

The box register holding the indentation box for the paragraph is available for inspection (or changes) inside hooks. It remains available even if the \OmitIndent command was used; in that case it will just not be automatically placed.

The expl3 name for the box register is \g_para_indent_box.

\RawIndent
\para_raw_indent:
\RawNoindent
\para_raw_noindent:
\RawParEnd
\para_raw_end:

\RawIndent *hmode material* \RawParEnd
\RawNoindent *hmode material* \RawParEnd

The commands \RawIndent and \RawNoindent are not meant for normal paragraph building (where the result is a textual paragraph in the traditional meaning of the word), but for special cases where TEX's low-level algorithm is used to achieve special effects, but where the result is not a "paragraph".

They are called "raw", because they bypass LATEX's hook mechanism for paragraphs and simply invoke the low-level TEX algorithm. I.e., they are like the original TEX primitives \indent and \noindent (that is they execute no hooks other than \everypar) except that they can only be used in vertical mode and generate an error if found elsewhere.

To avoid issues a paragraph started by them should always be ended by \RawParEnd[7] and not by \par (or a blank line), because the latter will execute hooks which then have no counterpart at the beginning of the paragraph. It is the responsibility of the programmer to make sure that they are properly paired. This also means that one should not put arbitrary user content between these commands if that content could contain stray \pars.

The expl3 names for the functions are \para_raw_indent:, \para_raw_indent: and \para_raw_end:.

2.3 Examples

None of the examples in this section are meant for real use as they are far too simple-minded but they should give some ideas of what could be possible if a bit more care is applied.

2.3.1 Testing the mechanism

The idea is to output for each paragraph encountered some information: a paragraph sequence number, a level number in roman numerals, the environment in which this paragraph appears, and the line number where the start or end of the paragraph is, e.g., something like

```
PARA: 1-i start (document env. on input line 38)
PARA: 1-i end   (document env. on input line 38)
```

[7] Technical note for those who know their *TEXbook*: the \RawParEnd command invokes the original TEX engine definition of \par that (solely) triggers the paragraph builder in TEX when found inside unrestricted horizontal mode and does nothing in other processing modes.

```
PARA: 2-i start (document env. on input line 40)
PARA: 3-ii start (minipage env. on input line 40)
PARA: 3-ii end   (minipage env. on input line 40)
PARA: 2-i end   (document env. on input line 41)
```

As you can see paragraph 2 starts on line 40 and ends on 41 and inside a minipage started paragraph 3 (start and end on line 40). If you run this on some document you will find that LaTeX considers more things "a paragraph" than you have probably thought.

This was generated by the following hook code:

```
\newcounter{paracnt}          % sequence counter
\newcounter{paralevel}        % level counter
```

To support paragraph nesting we need to maintain a stack of the sequence numbers. This is most easily done using expl3 functions, so we switch over. This is not a very general implementation, just enough for what we need and a bit of LaTeX 2_ε thrown in as well. When popping, the result gets stored in `\paracntvalue` and the `\ERROR` should never happen because it means we have tried to pop from an empty stack.

```
\ExplSyntaxOn
\seq_new:N \g_para_seq
\cs_new:Npn \ParaPush
  {\seq_gpush:No  \g_para_seq {\the\value{paracnt}}}
\cs_new:Npn \ParaPop  {\seq_gpop:NNF  \g_para_seq \paracntvalue \ERROR }
\ExplSyntaxOff
```

At the start of the paragraph increment both sequence counter and level and also save the then current sequence number on our stack.

```
\makeatletter % because we use a few internal 2e commands
\AddToHook{para/begin}{%
  \stepcounter{paracnt}\stepcounter{paralevel}%
  \ParaPush
```

To display the sequence number we `\typeout` the current sequence and level number. The command `\@currenvir` gives us the current environment and `\on@line` produces a space and the current input line number.

```
  \typeout{PARA: \arabic{paracnt}-\roman{paralevel} start
    (\@currenvir\space env.\on@line)}%
```

We also typeset the sequence number as a tiny red number in a box that takes up no horizontal space. This helps us seeing where LaTeX sees the start and end of the paragraphs in the document.

```
  \llap{\color{red}\tiny\arabic{paracnt}\ }%
}
```

At the end of the paragraph we display sequence number and level again. The level counter has the correct value but we need to retrieve the right sequence value by popping it off the stack after which it is available in `\paracntvalue` the way we have set this up above.

```
\AddToHook{para/end}{%
  \ParaPop
  \typeout{PARA: \paracntvalue-\roman{paralevel} end \space\space
    (\@currenvir\space env.\on@line)}%
```

We also typeset again a tiny red number with that value, this time sticking out to the right.[8] We also decrement the level counter since our level has finished.

```
  \rlap{\color{red}\tiny\ \paracntvalue}%
  \addtocounter{paralevel}{-1}%
}
\makeatother
```

2.3.2 Mark the first paragraph of each `itemize`

The code for this is rather simple. We supply some code that is executed only once inside a hook at the start of each `itemize`. We explicitly change the color back and forth so that we don't introduce grouping around the paragraph.

```
\AddToHook{env/itemize/begin}{%
  \AddToHookNext{para/begin}{\color{blue}}%
  \AddToHookNext{para/end}{\color{black}}%
}
```

As a result the first paragraph of each `itemize` will appear in blue.

2.4 Some technical notes

The code tries hard to be transparent for package code, but of course any change means that there is a potential for breaking other code. So in section we collect a few cases that may be of importance if low-level code is dealing with paragraphs that are now behaving slightly differently. The notes are from issues we observed and will probably grow over time.

2.4.1 Glue items between paragraphs (found with fancypar)

In the past LaTeX placed two glue items between two consecutive paragraphs, e.g.,

```
text1 \par text2 \par
```

would show something like

```
\glue(\parskip) 0.0 plus 1.0
\glue(\baselineskip) 5.16669
```

but now there is another `\parskip` glue (that is always 0pt):

```
\glue(\parskip) 0.0 plus 1.0
\glue(\parskip) 0.0
\glue(\baselineskip) 5.16669
```

[8] Note that this can alter the document pagination, because a paragraph ending in a display (e.g., an equation) will get an extra line—in that case our tiny number has an effect even though it doesn't take up any space, because it paragraph is no longer empty and thus isn't dropped!

The reason is that we generate a "fake" paragraph to gain control and safely add the early hooks, but this generates an additional glue item. That item doesn't contribute anything vertically but if somebody writes code that unravels a constructed list using `\lastbox`, `\unskip` and `\unpenalty` then the code has to remove one additional glue item or else it will fail.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.