Paragraph building and hooks*

Frank Mittelbach

October 30, 2026

Abstract

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

Contents

1 Introduction **1**
1.1 The default processing done by the engine 2

2 The new mechanism implemented for LaTeX **3**
2.1 The provided hooks 4
2.2 Altered and newly provided commands 6
2.3 Examples 7
2.3.1 Testing the mechanism 7
2.3.2 Mark the first paragraph of each `itemize` 9
2.4 Some technical notes 9
2.4.1 Glue items between paragraphs (found with fancypar) 9

3 The Implementation **9**
3.1 Providing hooks for paragraphs 10
3.2 The error messages 17

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).

*This file has version v1.0p dated 2026-10-29, © LaTeX Project.

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 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

[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!

```
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]

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

[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.

[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.

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.

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.

[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.

`\@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 by emptying the `\IndentBox`.

If the command is used in vertical mode then it simply issues a `\noindent`, i.e., starts a paragraph without indentation.

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. The `\OmitIndent` command clears its content.

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)
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
```

[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.

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
```

[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!

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
```

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.

3 The Implementation

```
1 ⟨@@=para⟩
2 ⟨*2ekernel | latexrelease⟩
3 \ExplSyntaxOn
4 ⟨latexrelease⟩\NewModuleRelease{2021/06/01}{ltpara}
5 ⟨latexrelease⟩                        {Paragraph~handling~and~hooks}
```

3.1 Providing hooks for paragraphs

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

The public hooks. They are implemented as a paired set of hooks.

```
\hook_new_pair:nn{para/before}{para/after}
\hook_new_pair:nn{para/begin}{para/end}
```

(*End of definition for* `para/before` *and others. These functions are documented on page* 4.)

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

The corresponding kernel hooks (for tagging and future extensions).

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

(*End of definition for* `\@kernel@before@para@before` *and others. These functions are documented on page* 5.)

\g__para_standard_everypar_tl

Whenever TEX starts a paragraph it inserts first an indentation box and then executes the tokens stored in `\tex_everypar:D` (known to LATEX as `\everypar`). We alter this behavior slightly here, so that hooks are added into the right place. Otherwise the process change remains transparent to any legacy code for this space.

We keep the standard code to be used by `\tex_everypar:D` in a separate token list because we have to switch back and forth for error recovery and so altering `\tex_everypar:D` all the time should be a tiny bit faster.

```
⟨latexrelease⟩\IncludeInRelease{2023/06/01}
⟨latexrelease⟩        {\g__para_standard_everypar_tl}{minipage~ fix}
\tl_new:N \g__para_standard_everypar_tl
```

Here is now its definition:

```
\tl_gset:Nn \g__para_standard_everypar_tl {
```

First we remove the indentation box and store it in `\g_para_indent_box`. If there was none because the paragraph was started by `\noindent` the box register will be void.

```
  \box_gset_to_last:N \g_para_indent_box
```

This will make the newly started horizontal list empty, so if we stop it now and return to vertical mode it will be dropped by TEX. We do that but inside a group so that any `\parshape` settings will not get lost as we need them for later.

```
  \group_begin:
    \tex_par:D
  \group_end:
```

We then change `\tex_everypar:D` to generate an error so that we can detect and report if the `para/before` hook illegally changed out of vmode.

```
  \tex_everypar:D { \msg_error:nnnn { hooks }{ para-mode }{before}{vertical} }
  \@kernel@before@para@before
  \hook_use:n {para/before}
```

Assuming the hooks have been well behaved it is time to return to horizontal mode and start the paragraph in earnest. We already have the indentation box saved away so we now have to restart the paragraph with an empty `\tex_everypar:D` and with `\tex_noindent:D`. And we need to make sure not to get another `\parskip` or rather (since we can't prevent that) that it is of zero size.

```
  \group_begin:
    \tex_everypar:D {}
```

There has been a long-standing problem with LaTeX's minipages in that invisible material at the beginning of a minipage (such as a `\color` setting) would result in `\parskip` being added in front of the first paragraph—something that is not done by TeX if a vertical list is completely empty. As this is happening on a very low-level in the engine it wasn't really possible to find out if this `\parskip` was added or if a space we see in front of the current point is legitimate. However, with the new paragraph handling we are in a better position: while we still don't know if there is such a space or not, we do know if we have just created an empty paragraph. Thus, if we now set `\parskip` to `-\parskip` the two will cancel each other if present and if the first was ignored because the vertical list was empty, then the second will be ignored too because it is still empty. Of course, we don't want to cancel always but only at the start of a minipage and that is signaled with the `@minipage` switch.

```
    \skip_set:Nn \tex_parskip:D
         { \if@minipage -\tex_parskip:D \else: \c_zero_skip \fi: }
    \tex_noindent:D
  \group_end:
```

That brings us back to the start of the horizontal list but we need to change `\tex_everypar:D` back to its normal content in case there are nested paragraphs coming up.

```
  \tex_everypar:D{\g__para_standard_everypar_tl}
```

This is followed by executing the kernel and the public hook. The kernel hook is there to enable tagging.

```
  \@kernel@before@para@begin
  \hook_use:n {para/begin}
```

If we aren't in horizontal mode any longer the hooks above misbehaved.

```
  \if_mode_horizontal: \else:
    \msg_error:nnnn { hooks }{ para-mode }{begin}{horizontal} \fi:
```

Finally we reinsert the indentation box (unless suppressed) and then call `\everypar` the way legacy LaTeX code expects it.

However, adding the public `\everypar` is a bit tricky (see below) so we add that later, and indirectly.

```
  \__para_handle_indent:
% \the \everypar           % <--- done differently below
}

⟨latexrelease⟩\cs_set:Npn \__para_tmp:w #1#2#3#4#5 { }
⟨latexrelease⟩\tl_gput_right:Nx \g__para_standard_everypar_tl {
⟨latexrelease⟩   \exp_not:N \the
⟨latexrelease⟩   \exp_not:N \toks
⟨latexrelease⟩   \exp_after:wN \__para_tmp:w \token_to_meaning:N \everypar
⟨latexrelease⟩   \c_space_tl
⟨latexrelease⟩}
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{2021/06/01}
⟨latexrelease⟩        {\g__para_standard_everypar_tl}{minipage~ fix}
⟨latexrelease⟩
⟨latexrelease⟩\tl_gset:Nn \g__para_standard_everypar_tl {
⟨latexrelease⟩   \box_gset_to_last:N \g_para_indent_box
⟨latexrelease⟩   \group_begin:
⟨latexrelease⟩     \tex_par:D
⟨latexrelease⟩   \group_end:
```

```
⟨latexrelease⟩  \tex_everypar:D { \msg_error:nnnn { hooks }{ para-mode }{before}{vertical} }
⟨latexrelease⟩  \@kernel@before@para@before
⟨latexrelease⟩  \hook_use:n {para/before}
⟨latexrelease⟩  \group_begin:
⟨latexrelease⟩    \tex_everypar:D {}
⟨latexrelease⟩    \skip_zero:N \tex_parskip:D
⟨latexrelease⟩    \tex_noindent:D
⟨latexrelease⟩  \group_end:
⟨latexrelease⟩  \tex_everypar:D{\g__para_standard_everypar_tl}
⟨latexrelease⟩  \@kernel@before@para@begin
⟨latexrelease⟩  \hook_use:n {para/begin}
⟨latexrelease⟩  \if_mode_horizontal: \else:
⟨latexrelease⟩    \msg_error:nnnn { hooks }{ para-mode }{begin}{horizontal} \fi:
⟨latexrelease⟩  \__para_handle_indent:
⟨latexrelease⟩}
```

We also have to add the `\everypar` toks register at the end. In case of rollback this is already allocated and we have to find out the correct number (hope this is correctly done)

```
⟨latexrelease⟩\cs_set:Npn \__para_tmp:w #1#2#3#4#5 { }
⟨latexrelease⟩\tl_gput_right:Nx \g__para_standard_everypar_tl {
⟨latexrelease⟩    \exp_not:N \the
⟨latexrelease⟩    \exp_not:N \toks
⟨latexrelease⟩    \exp_after:wN \__para_tmp:w \token_to_meaning:N \everypar
⟨latexrelease⟩    \c_space_tl
⟨latexrelease⟩}
⟨latexrelease⟩\EndIncludeInRelease
```

(*End of definition for* `\g__para_standard_everypar_tl`.)

`\tex_everypar:D` `\tex_everypar:D` then only has to execute `\g__para_standard_everypar_tl` by default.

```
\tex_everypar:D{\g__para_standard_everypar_tl}
```

(*End of definition for* `\tex_everypar:D`.)

`\everypar` Tokens inserted at the beginning of the paragraph are placed into `\everypar` inside legacy LaTeX code, e.g., by the list environments or by headings to handle `\clubpenalty`, etc. Now this isn't any longer the primitive but simply a toks register used in the code above but to legacy LaTeX code that is transparent.

There is, however, a problem: a handful packages use exactly the same trick and replace the primitive with a token register and call the token register inside the renamed primitive. That is they assume that `\everypar` is the primitive and that it will still be called at the start of the paragraph even if renamed.

But if we have already replaced it by a token register then all they do is to give that token register a new name. Thus our code in `\tex_everypar:D` would call `\everypar` (which is now their token register) and the code that they added ends up in our token register which is then never used at all. A bit mind boggling I guess.

So what we have to do is not to call the token register `\everypar` by its name inside `\tex_everypar:D` but by using its actual register number.

```
\newtoks \everypar
```

After we have allocated a new toks register with the name `\everypar` the actual register number is available (briefly) inside `\allocationnumber`. So instead of `\the\everypar` we have to put `\the\toks`⟨*allocated number*⟩ at the end of `\tex_everypar:D`.

So what remains doing is to append a few tokens to the token list `\g__para_standard_everypar_tl` which we do now. We use `x` expansion here to get the value of `\allocationnumber` in, all the other tokens should not be expanded at this point.

One important point here is to terminate the register allocation number with a real space. This space will get swallowed up when the number is read. Anything else, such as `\scan_stop:` would remain in the input and that would mean that it would interfere with `\everypar` code that attempts to scan ahead to see how the paragraph text starts.

```
\tl_gput_right:Ne \g__para_standard_everypar_tl {
  \exp_not:N \the
  \exp_not:N \toks
  \the \allocationnumber
  \c_space_tl
}
```

(*End of definition for* `\everypar`.)

`\g_para_indent_box` For managing the indentation we need to provide a public accessible box register

```
\box_new:N \g_para_indent_box
```

(*End of definition for* `\g_para_indent_box`. *This function is documented on page 6.*)

`\__para_handle_indent:` Adding (typesetting) the indent box is straight forward. If it was emptied before it does nothing.

```
\cs_new:Npn \__para_handle_indent: {
  \box_use_drop:N \g_para_indent_box
}
```

The declaration `\para_omit_indent:` (or `\OmitIndent`) changes that to do nothing—the case when used inside the `para/begin` hook. If encountered in vertical mode it starts the paragraph without indentation.

```
\cs_new_protected:Npn \para_omit_indent: {
  \mode_if_vertical:TF
     { \noindent }
     { \box_gclear:N \g_para_indent_box }
}
```

(*End of definition for* `\__para_handle_indent:`.)

`\IndentBox` `\OmitIndent` The LaTeX 2_ε names for the indentation box and for suppressing it for use in the `para/begin` hook.

```
\cs_set_eq:NN \IndentBox \g_para_indent_box
\cs_set_eq:NN \OmitIndent \para_omit_indent:
```

(*End of definition for* `\IndentBox` *and* `\OmitIndent`. *These functions are documented on page 6.*)

`\para_end:` Adding hooks to the end of a paragraph is similar but here we need to alter the command that is used by TeX to end horizontal mode and return to vertical mode, i.e., `\par`.

This is a bit more complicated as this command can appear anywhere either explicitly or implicitly added by TeX in certain situations:

- when using `\par` in the code or the document

- when using a blank line (which is converted to `\par`)
- when TEX finds any commands incompatible with horizontal mode it issues a `\par` and then rereads the command.

Unfortunately, TEX has some (these days) unnecessary optimizations: if a `\vbox` ends and TEX is still in horizontal mode it simply exercises the paragraph builder instead of issuing a `\par`. It is therefore necessary for LATEX to ensure that this case doesn't happen and all boxes internally have a `\par` command at their end.

This `\par` may or may not run the "par primitive" (which is always available as `\tex_par:D` in expl3); it is permissible to have a changed meaning and it is in fact changed by LATEX in various ways at various points inside `latex.ltx`. For this LATEX 2_ε code has the following conventions: `\@@par` and `\endgraf` both refer to the default meaning (in the past this was the initex primitive) while `\par` is the current meaning which maybe does something else.

We are now going to change this default meaning to instead run `\para_end:`, which ultimately executes the initex primitive but additionally adds our hooks when appropriate. This way the change is again transparent to the legacy LATEX 2_ε code.

In most cases `\para_end:` should behave exactly like the primitive and we achieve this by simply expanding it to the primitive which is available to us as `\tex_par:D`. This way we don't have to care about whether TEX just does nothing (e.g., if in vertical mode already) or generates an error, etc.

```
\cs_new_protected:Npn \para_end: {
```

CCC Maybe needs more explanation. TEMP NOTE: What should happen if in outer hmode with an empty hlist?

The only case we care about is when we are in horizontal mode (i.e., doing typesetting) and not also in inner mode (i.e., making paragraphs and not building an `\hbox`.

```
\bool_lazy_and:nnT
     { \mode_if_horizontal_p: }
     { \bool_not_p:n { \mode_if_inner_p: } }
     { ...
```

Since this is executed for each and every paragraph in a document we try to stay as fast as possible, so we do not use the above construct but two conditionals instead. Using low-level `\if_mode...` conditions would be even faster but has the danger to conflict with conditionals in the user hooks.

If `\para_end:` is executed while TEX is currently doing a low-level assignment the test for horizontal mode may get executed as part of the assignment. That is normally not an issue but we just found one case where it is:

```
\afterassignment\lst@vskip\@tempskipa \z@ \par
```

If TEX is in hmode while that assignment happens then the `\par` is seen in hmode because in the above case the assignment may not be finished (one should have used `\z@skip`) and the `\lst@vskip` will get inserted into the middle of the conditional. The `\lst@vskip` then changes to vmode and you get a surprising error about the `para/end` hook having changed modes even if you don't have any hook code(!): it is the inserted `\lst@vskip` that is actually causing the change of mode.This is what happened when the output routines got started while a `lstlisting` environment (that redefines `\vskip` in this way) was active. This is really faulty coding, but we try to be proactive and guard the conditional so that any scanning is first stopped, thus:

```
   \scan_stop:
```

```
\mode_if_horizontal:TF {
  \mode_if_inner:F {
```

In that case the action of the primitive would be to remove the last glue (but no kerns) from the horizontal list (constructed to form a paragraph) and then to append a penalty of 10000 and the `\parfillskip`; it then passes the whole list to the paragraph builder, which breaks it into lines and TEX then returns to vertical mode.

What we want to do is to add this hook code at the end of the horizontal list before any of the above happens. If there was a glue item at the end of the list then it should get removed before the hook code gets added so we have to arrange for this removal.

As in other similar cases, it may be best to add here a `\nobreak` in case the hook itself adds glue and thus creates a non-explicit and unwanted potential breakpoont. On the other hand (as has been argued) the code in the hook should perhaps have the responsibility for adding such a guard penalty in this casse. This needs further analysis and decisions (as in emails).

In either case, good documentation of these hooks is essential, covering what the hook may or should provide and all such related considerations convernimg the content.

There is not much point in checking if there was really a glue item at the end of the horizontal list, instead we simply try to remove one using `\tex_unskip:D`: if there wasn't one this will do nothing.

```
        \tex_unskip:D
```

We then execute the public hook (which may add some final typeset material) followed by the kernel hook that we need for adding tagging support. None of this is supposed to change the mode—at the moment we make only a very simple test for this, more devious changes go unnoticed, but too bad as they will then probably backfire badly.

```
        \hook_use:n{para/end}
        \@kernel@after@para@end
        \mode_if_horizontal:TF {
```

The final action (before getting to the point where `\tex_par:D` is called) is to add an extra glue item so that the primitive is prevented from removing intended glue (if there was some). If we don't do this and the horizontal list ends in several glue items we would end up removing two glue items instead of just the last one, which would be wrong. We use glue (rather than a kern) as that will be removed by the primitive.

There is however one other TEX optimization that hurts: in a sequence like this `$$ ... $$ \par` (with `\par` being the primitive) TEX will be in horizontal mode after the display, ready to receive further paragraph text, but since the `\par` follows immediately there is a "null" paragraph at the end and TEX simply throws that away. The space between `$$` and `\par` got already dropped during the display processing so the `\par` is not removing any space and appending `\parfillskip`, instead it simply goes silently to vmode.

Now if we would have added something (to prevent glue removal) that would look to TEX like material after the display and so we would end up with an empty paragraph just containing a penalty and `\parfillskip`.

We therefore check if the current hlist does end in glue (`\tex_lastnodetype:D` has the value `11`) and if so we add a zero-length guard skip which will be removed by the following `\tex_par:D`.

```
        \if_int_compare:w 11 = \tex_lastnodetype:D
          \tex_hskip:D \c_zero_dim
        \fi:
```

To run the para/after hook we first end the paragraph. This means that the \tex_par:D at the very end is unnecessary but executing it there unnecessarily is better than having code that tests for all the different mode possibilities.

```
      \tex_par:D
      \hook_use:n{para/after}
      \@kernel@after@para@after
    }
```

If we were not horizontal mode (the F case from above) then the earlier hook para/end must have been at fault, so we report that.

```
    { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
```

Finally close out the nested conditionals.

```
  }
}
```

And then we can use the primitive to truly end the paragraph.

```
  \tex_par:D
}
```

(*End of definition for* \para_end:. *This function is documented on page* 6.)

\para_raw_indent:
\para_raw_noindent:
\para_raw_end:

The commands \para_raw_indent: and \para_raw_noindent: are like the primitives \indent and \noindent except that they can only be used in vertical mode.

To avoid issues a paragraph started by them should always be ended by \para_raw_end: and not by \para_end: or \par as 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.

```
\cs_new:Npn \para_raw_indent: {
  \mode_if_vertical:TF
      {
        \tex_everypar:D {
          \box_gset_to_last:N \g_para_indent_box
          \tex_everypar:D { \g__para_standard_everypar_tl }
          \__para_handle_indent:
          \the\everypar }
      }
      { \msg_error:nn { latex2e }{ raw-para } }
  \tex_indent:D
}
\cs_new:Npn \para_raw_noindent: {
  \mode_if_vertical:TF
      {
        \tex_everypar:D {
          \tex_everypar:D { \g__para_standard_everypar_tl }
          \the\everypar }
      }
      { \msg_error:nn { latex2e }{ raw-para } }
  \tex_noindent:D
}
\cs_new_eq:NN \para_raw_end: \tex_par:D
```

(*End of definition for* \para_raw_indent:, \para_raw_noindent:, *and* \para_raw_end:. *These functions are documented on page* 7.)

\RawIndent
\RawNoIndent
\RawParEnd

The LaTeX 2_ε names for starting and ending a paragraph without adding any hooks.

```
\cs_set_eq:NN \RawIndent   \para_raw_indent:
\cs_set_eq:NN \RawNoindent \para_raw_noindent:
\cs_set_eq:NN \RawParEnd   \para_raw_end:
```

(*End of definition for* `\RawIndent`, `\RawNoIndent`, *and* `\RawParEnd`. *These functions are documented on page* 7.)

This ends the `para` module code.

```
⟨@@=⟩
```

\par
\endgraf
\@@par

Having the new default definition for `\par` we also have to set it up so that it gets used. This involves three commands: `\par`, `\@@par` (to which LaTeX resets `\par` occasionally) and `\endgraf`, which is another name for the "default" action of `\par`.

```
\cs_set_eq:NN \par     \para_end:
\cs_set_eq:NN \@@par   \para_end:
\cs_set_eq:NN \endgraf \para_end:
```

(*End of definition for* `\par`, `\endgraf`, *and* `\@@par`. *These functions are documented on page* 6.)

While this is not integrated properly into the format we have to redo the `\everypar` setting from the kernel, otherwise that gets lost (as it happens before that file is loaded).

```
\everypar{\@nodocument} %% To get an error if text appears before the \document
```

3.2 The error messages

This one is used when we detect that some hook code has changed the mode where it shouldn't, e.g., by starting or ending a paragraph. The first argument is the hook name second the mode it should have stayed in but didn't.

```
\msg_new:nnnn { hooks } { para-mode }
  {
    Illegal~mode~ change~ in~ hook~ 'para/#1'.\\
    Hook~ code~ did~ not~ remain~ in~ #2~ mode.
  }
  {
    Paragraph~ hooks~ cannot~ change~ the~ TeX~ mode~ without~ causing~
    endless~ recursion.~ The~ hook~ code~ in~ 'para/#1'~ needs~ to~ stay~
    in~ #2~ mode,~ but~ it~ didn't.~ Examine~ the~ hook~
    code~ with~ \iow_char:N \\ShowHook~ to~ find~ the~ issue.
  }
```

And here is one used in the "raw" commands when they are used outside of vertical mode.

```
\msg_new:nnnn { latex2e } { raw-para }
  {
     Not~ in~ vertical~ mode.
  }
  {
    Starting~ a~ paragraph~ with~ \iow_char:N \\RawIndent~ or~
    \iow_char:N \\RawNoindent \\
    (or~ \iow_char:N \\para_raw_indent:~ or~
    \iow_char:N \\para_raw_noindent:)~ is~ only~ allowed \\
    if~ LaTeX~ is~ in~ vertical~ mode.
  }
```

```
%
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩                  {ltpara}{Undo~hooks~for~paragraphs}
⟨latexrelease⟩
⟨latexrelease⟩\let \OmitIndent  \@undefined
⟨latexrelease⟩\let \IndentBox   \@undefined
⟨latexrelease⟩\let \RawIndent   \@undefined
⟨latexrelease⟩\let \RawNoindent \@undefined
⟨latexrelease⟩\let \RawParEnd   \@undefined
⟨latexrelease⟩
⟨latexrelease⟩\cs_set_eq:NN \par     \tex_par:D
⟨latexrelease⟩\cs_set_eq:NN \@@par   \tex_par:D
⟨latexrelease⟩\cs_set_eq:NN \endgraf \tex_par:D
⟨latexrelease⟩
```

We also need to clean up the primitive "everypar" as that should no longer execute any code by default. And, of course, make `\everypar` become the primitive again.

```
⟨latexrelease⟩\tex_everypar:D {}
⟨latexrelease⟩\cs_set_eq:NN \everypar \tex_everypar:D
⟨latexrelease⟩
⟨latexrelease⟩\EndModuleRelease
\ExplSyntaxOff
⟨/2ekernel | latexrelease⟩
```