The l3galley package
Galley code*

The LaTeX3 Project†

Released 2013/05/24

1 Introduction

In LaTeX3 terminology a galley is a rectangular area which receives text and other material filling it from top. The vertically extend of a galley is normally not restricted: instead certain chunks are taken off the top of an already partially filled galley to form columns or similar areas on a page. This process is typically asynchronous but there are ways to control or change its behaviour.

Examples for galleys are "the main galley", where the continuous document data gets formatted into and from which columns and pages are constructed, and "vertical box galleys", such as the body of a minipage environment. The latter galleys are typically not split after formatting, though there can be exceptions.

2 Formatting layers

The present module is mainly concerned with the formatting of text in galleys. The mechanism by which this is achieved uses four (somewhat) distinct layers, some of which can be addressed using the templates provided here.

2.1 Layer one: external dimensions

The bottom layer of the system is the external dimensions of the galley. Normally only the horizontal dimension is fixed externally, while the vertical (filling) dimension is unspecified. The external dimensions are fixed when starting a new galley, and are therefore not modifiable within the galley.

There are no templates for setting this layer directly, although the external values are influenced by other parts of the system (for example when creating minipage environments).

*This file describes v4494, last revised 2013/05/24.

†E-mail: latex-team@latex-project.org

2.2 Layer two: internal dimensions

The second layer is the internal dimensions of the galley: the *measure* used for paragraph text and the position of the paragraph relative to the edges of the galley.

This layer is normally accessed by higher-level templates *via* the object type `measure`. Changes made using level two templates will often extend for large parts of a document (up to and including the entire document).

2.3 Layer three: paragraph shape

The third layer defines the paragraph shape within the measure as provided by the second layer. In the absence of any specification for that layer the paragraph shape used will be that of a rectangular area of the width of the current measure.

There are some restrictions imposed on the shape of a paragraph by the underlying TEX mechanisms. For example, cut out sections in paragraphs can be specified from the top of the paragraph but not from the bottom.

2.4 Layer four: formatting inside the paragraph

The forth layer deals with the paragraph formatting aspects such as hyphenation and justification within the paragraph (this is sometimes referred to as "`h&j`" or "`hj`"). This layer is somewhat distinct from the galley as such, but is handled in the same place as there is, internally, interaction between the different layers.

3 Code interfaces

3.1 Galley layers

`\l_galley_width_dim`

The total width of a galley, set either by the page geometry code for the main vertical galley or when creating an independent galley, such as a minipage.

`\galley_level:`

`\galley_level:`

Sets up a vertical box to contain a new galley level. The box should be "color safe", which is automatic for LATEX3 coffins but must be included manually (using `\color_group_begin:` and `\color_group_end:`) in "raw" vertical boxes.

3.2 Measure

\galley_margins_set_absolute:nn \galley_margins_set_absolute:nn {⟨*left margin*⟩} {⟨*right margin*⟩}
\galley_margins_set_relative:nn \galley_margins_set_relative:nn {⟨*left margin*⟩} {⟨*right margin*⟩}

Sets the width of the measure to have the ⟨*left margin*⟩ and ⟨*right margin*⟩ specified by the arguments, both of which are ⟨*dimension expressions*⟩. The `relative` function will adjust the text width within any existing margins, whereas the `absolute` measure sets the margins based on the edges of the galley only. One or both of the ⟨*margins*⟩ may be negative, to specify and outdent.

3.3 Between paragraphs

\g_galley_previous_par_lines_int

The number of lines in the preceding conceptual paragraph. This may not correspond to the TEX `\prevgraf` primitive value as one conceptual paragraph may contain several TEX `\par` primitive tokens.

\g_galley_restore_running_tl

When galley settings need to be reset at the end of a paragraph, the appropriate detail should be added to this token list. It is inserted immediately before the start of each paragraph, and can therefore be used to clear otherwise global settings. The token list itself is also cleared as part of this process.

\g_galley_no_break_next_bool

Indicates that no page break should be allowed between the current paragraph and the next paragraph.

\g_galley_omit_next_indent_bool

Indicates that the indent should be omitted from the start of the next paragraph started.

\l_galley_interpar_penalty_int

The ⟨*penalty*⟩ for a break between paragraphs. The ⟨*penalty*⟩ should be in the range −10 000 to 10 000, where −10 000 forces a page break, 0 has no effect at all and 10 000 forbids a page break. Note that setting `\g_galley_no_break_next_bool` to `true` will override any setting of `\l_galley_interpar_penalty_int`.

\l_galley_interpar_vspace_skip

Stretchable space to be inserted between paragraphs, set at the design or template level.

`\galley_penalty_set_single:n` `\galley_penalty_set_single:n {⟨penalty⟩}`

Sets the ⟨*penalty*⟩ for a break between the current and next paragraph on a one-off basis. This function is intended for user-level adjustments to design, and takes precedent over both settings from `\l_galley_interpar_penalty_int` and from `\galley_no_break_next:`.

`\galley_vspace_set_single:n` `\galley_vspace_set_single:n {⟨space⟩}`

Sets the ⟨*space*⟩ to be inserted between the current and next paragraph on a one-off basis. This function is intended for user-level adjustments to design, and otherwise is analogous to `\galley_set_vspace:n`.

3.4 Paragraph shape

`\galley_parshape_set_multi:nnnN` `\galley_parshape_set_multi:nnnN {⟨unaltered lines⟩} {⟨left indents⟩}`
`\galley_parshape_set_multi:nVVN` `{⟨right indents⟩} ⟨resume flag⟩`
`\galley_parshape_set_single:nnnN` `\galley_parshape_set_single:nnnN {⟨unaltered lines⟩} {⟨left indents⟩}`
`\galley_parshape_set_single:nVVN` `{⟨right indents⟩} ⟨resume flag⟩`

Sets the current paragraph shape to create an arbitrary paragraph shape. The paragraph shape is set such that there are ⟨*unaltered lines*⟩ which have width and indent as set by the measure. The "altered" lines are then defined by the comma-separated lists of ⟨*left indents*⟩ and ⟨*right indents*⟩. These are both indents from the edge of the measure, and may be negative, and should both contain the same number of items. If the ⟨*resume flag*⟩ is `true`, after the last altered line the paragraph shape returns to that of the measure. On the other hand, if the flag is `false` then the shape of the last line is retained for the rest of the paragraph. For example,

```
\galley_parshape_set_multi_par:nnnN { 1 }
  { 2 pt , 4 pt , 6 pt } { 2 pt , 4 pt , 6 pt } \c_true_bool
```

would create a paragraph shape in which the first line is the full width of the measure, the second line is indented by 2 pt on each side, the third line by 4 pt and the fourth line and subsequent lines by 6 pt from the edge of the measure on each side.

The `single_par` version applies only to a single paragraph, while the `multi_par` function sets the paragraph shape on an ongoing basis within the scope of the current TEX group.

`\galley_cutout_left:nn` `\galley_cutout_left:nn {⟨unaltered lines⟩} {⟨indents⟩}`
`\galley_cutout_right:nn` `\galley_cutout_right:nn {⟨unaltered lines⟩} {⟨indents⟩}`

Adds a cutout section to the active paragraph shape, leaving ⟨*unaltered lines*⟩ unchanged and then applying the ⟨*indents*⟩ (a comma list). The cutout will be placed on the left or right as indicated by the function name, and will apply to exactly the number of lines specified (the total of the ⟨*unaltered lines*⟩ and the number of entries in the ⟨*indents*⟩ list). Several cutouts may be applied sequentially: these act in an additive sense.

3.5 Formatting inside the paragraph

The settings described here apply "inside" the paragraph, and so are active irrespective of any paragraph shape within the measure.

`\l_galley_line_left_skip`
`\l_galley_line_right_skip`

Stretchable space added to the appropriate side each line in a paragraph.

`\l_galley_par_begin_skip`
`\l_galley_par_end_skip`

Stretchable space added to the beginning of the first line and end of the last line of a paragraph, respectively.

`\l_galley_par_indent_dim`

Fixed space added to the start of each paragraph except for those where `\g_galley_omit_next_indent_bool` is `true`.

`\l_galley_last_line_fit_int`

Determines how the inter-word stretch is set for the last line of a paragraph when

1. the value of `\l_galley_par_end_skip` contains an infinite (`fil(l)`) component;
2. the values of `\l_galley_line_left_skip` and `\l_galley_line_right_skip` do *not* contain an infinite (`fil(l)`) component.

Under these circumstances, `\l_galley_last_line_fit_int` is active, and applies as follows:

- when set to 0, the last line of the paragraph is set with the inter-word spacing at natural width;
- when set to a 1000 (or above), the inter-word spacing in the last line is stretched by the same factor as that applied to the penultimate line;
- when set to n between these extremes, the inter-word spacing in the last line is stretched by $n/1000$ times the factor used for the penultimate line.

`\galley_interword_spacing_set:N` `\galley_interword_spacing_set:N` ⟨*fixed spacing bool*⟩

Sets the inter-word spacing used based on the values supplied by the current font. If the ⟨*fixed spacing bool*⟩ flag is `true` then no stretch is permitted between words, otherwise the stretch specified by the font designer is used.

3.6 Display material

Material which is set in "display-style" require additional settings to control the relationship with the surrounding material.

`\galley_display_begin:`
`\galley_display_end:`

```
\galley_display_begin:
...
\galley_display_end:
```

Sets up a group to contain display-style material. Unlike an independent galley level, settings are inherited from the surroundings. However, the interaction of a display block with the paragraphs before and after it can be adjusted independent of the design of text.

3.7 Hyphenation

`\l_galley_hyphen_left_int`

THIS IS A HACK: SEE CODE!

3.8 Line breaking

`\l_galley_binop_penalty_int`

Penalty charged if an inline math formula is broken at a binary operator.

`\l_galley_double_hyphen_demerits_int`

Extra demerit charge of two (or more) lines in succession end in a hyphen.

`\l_galley_emergency_stretch_skip`

Additional stretch assumed for each line if no better line breaking can be found without it. This stretch is not actually added to lines, so its use may result in underfull box warnings.

`\l_galley_final_hyphen_demerits_int`

Extra demerit charge if the second last line is hyphenated.

`\l_galley_linebreak_badness_int`

Boundary that if exceeded will cause TEX to report an underfull line.

`\l_galley_linebreak_fuzz_dim`

Boundary below which overfull lines are not reported.

`\l_galley_linebreak_penalty_int`

Extra penalty charged per line in the paragraph. By making this penalty higher TEX will try harder to produce compact paragraphs.

`\l_galley_linebreak_pretolerance_int`

Maximum tolerance allowed for individual lines to break the paragraph without attempting hyphenation.

`\l_galley_linebreak_tolerance_int`

Maximum tolerance allowed for individual lines when breaking a paragraph while attempting hyphenation (if this limit can't be met `\l_galley_emergency_stretch_skip` comes into play).

`\l_galley_mismatch_demerits_int`

Extra demerit charge if two visually incompatible lines follow each other.

`\l_galley_relation_penalty_int`

Penalty charged if an inline math formula is broken at a relational symbol.

`\galley_break_line:Nn` `\galley_break_line:Nn` ⟨*boolean*⟩ {⟨*dimexpr*⟩}

Breaks the current line, filling the remaining space with `fil` glue. If the ⟨*boolean*⟩ is `true` then a page break is possible after the broken line. Vertical space as given by the ⟨*dimexpr*⟩ will be inserted between the broken line and the next line.

3.9 Paragraph breaking

`\l_galley_parbreak_badness_int`

Boundary that if exceeded will cause TEX to report an underfull vertical box.

`\l_galley_parbreak_fuzz_dim`

Boundary below which overfull vertical boxes are not reported.

`\l_galley_broken_penalty_int`

Penalty for page breaking after a hyphenated line.

`\l_galley_pre_display_penalty_int`

Penalty for breaking between immediately before display math material.

`\l_galley_post_display_penalty_int`

Penalty for breaking between immediately after display math material.

\galley_club_penalties_set:n
\galley_club_penalties_set:(V|v)
\galley_display_club_penalties_set:n
\galley_display_club_penalties_set:(V|v)
\galley_display_widow_penalties_set:n
\galley_display_widow_penalties_set:(V|v)
\galley_widow_penalties_set:n
\galley_widow_penalties_set:(V|v)

\galley_club_penalties_set:n {⟨*penalty list*⟩}

Set the penalties for breaking lines at the beginning and end of (partial) paragraphs. In each case, the ⟨*penalty list*⟩ is a comma-separated list of penalty values. The list applies as follows:

`club` Penalties for breaking after the first, second, third, *etc.* line of the paragraph.

`display_club` Penalties for breaking after the first, second, third, *etc.* line after a display math environment.

`display_club` Penalties for breaking before the last, penultimate, antepenultimate, *etc.* line before a display math environment.

`widow` Penalties for breaking before the last, penultimate, antepenultimate, *etc.* line of the paragraph.

In all cases, these penalties apply in addition to the general interline penalty or to any "special" line penalties.

\galley_interline_penalty_set:n

\galley_interline_penalty_set:n {⟨*penalty*⟩}

Sets the standard interline penalty applied between lines of a paragraph. This value is added to any (display) club or widow penalty in force.

\galley_interline_penalties_set:n
\galley_interline_penalties_set:V

\galley_interline_penalties_set:n {⟨*penalty list*⟩}

Sets "special" interline penalties to be used in place of the standard value, specified as a comma-separated ⟨*penalty list*⟩. The ⟨*penalties*⟩ apply to the first, second, third, *etc.* line of the paragraph.

\galley_save_club_penalties:N
\galley_save_display_club_penalties:N
\galley_save_display_widow_penalties:N
\galley_save_interline_penalties:N
\galley_save_widow_penalties:N

\galley_save_club_penalties:N {⟨*comma list*⟩}

These functions save the current value of the appropriate penalty to the comma list specified, within the current TEX group.

\galley_interline_penalty: ⋆

\galley_interline_penalty:

Expands to the current interline penalty as a ⟨*integer denotation*⟩.

4 Hooks and insertion points

\g_galley_par_begin_hook_tl

Token list inserted at the beginning of every paragraph in horizontal mode. This is inserted after any paragraph indent but before any other horizontal mode material.

\g_galley_par_end_hook_tl Token list inserted at the end of every paragraph in horizontal mode.

\g_galley_par_reset_hook_tl

Token list inserted after each paragraph. This is used for resetting galley parameters, and is therefore cleared after use. It is inserted in vertical mode and must not contain horizontal mode material.

\g_galley_whatsit_next_tl Token list for whatsits to be inserted at the very beginning of the next paragraph started.

\g_galley_whatsit_previous_tl

Token list for whatsits to be inserted at the very end of the last paragraph started.

5 Paragraphs

\galley_par: `\galley_par:`

Finalises the material collected as the last paragraph, inserts tokens at the start of the new paragraph, setting paragraph shape and any special values as required.

\galley_par:n `\galley_par:n {`⟨*tokens*⟩`}`

Adds the ⟨*tokens*⟩ to the material collected for the last paragraph before finalising it in the usual way. This function should therefore be the *first* non-expandable entry used when a function needs to add tokens to the preceding paragraph.

6 Information variables

Some of the variables for the galley mechanism may be of interest to the programmer. These should all be treated as read-only values and accessed only through the defined interfaces described above.

\l_galley_total_left_margin_dim

The total margin between the left side of the galley and the left side of the text block. This may be negative if the measure is set to overlap the text beyond the edge of the galley.

\l_galley_total_right_margin_dim

The total margin between the right side of the galley and the right side of the text block. This may be negative if the measure is set to overlap the text beyond the edge of the galley.

\l_galley_text_width_dim

The width of a line of text within the galley, taking account of any margins added. This may be larger than \l_galley_width_dim if the margins are negative.

7 l3galley Implementation

At the implementation level, there are a number of challenges which have to be overcome in order to make the galley easy to use at the designer and user levels. Inserting material into the main vertical list is in many ways an irreversible operation. Inserting items as they appear in the source is therefore not desirable. Instead, inserting vertical-mode material needs to be delayed until the start of the "next" paragraph. This is particularly notable for invisible items such as whatsits and specials, which will otherwise cause changes in spacing. Delaying insertion enables user-supplied settings to override design settings in a reliable fashion. This can be achieved as the design-level material can be ignored if a user value is supplied. There is a need to allow proper nesting of galleys, which means that all of the above needs to be set up so that it can be saved and restored. All of these manipulations require altering the meaning of the \par token, which is particularly awkward as TEX inserts a token *called* \par rather than one with a particular meaning. This makes moving \par to somewhere "safe" extremely challenging.

Added to all of this complexity, there is a need to deal with "display-like" material. The most obvious example is the way lists are handled. These use \par tokens to achieve the correct appearance, but at the same time

```
Text
\begin{itemize}
  \item An item
\end{itemize}
More text
```

should form one visual paragraph while

```
Text
\begin{itemize}
  \item An item
\end{itemize}

More text
```

should be shown as two paragraphs. This requires an additional level of handling so that the \par token used to end the list in the first case does not start a new paragraph in a visual sense while the second does.

Another factor to bear in mind is that \tex_everypar:D may be executed inside a group. For example, a paragraph starting

```
{Text} here
```

will insert the tokens such that the current group level is 1 higher for "Text" than for "here". The result of this is that it's very important to watch how flags are set and reset. This can only be done reliably on a global level, which then has a knock-on effect on the rest of the implementation.

At a TEX level, settings can only apply to the current paragraph, but conceptually there is a need to allow for both single-paragraph and "running" settings. Whenever the code switches galley level both of these need to be correctly saved.

```
⟨*initex | package⟩
⟨@@=galley⟩
⟨*package⟩
\ProvidesExplPackage
  {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
⟨/package⟩
```

7.1 Support items

Functions or settings which are needed by the galley but perhaps also elsewhere.

`\seq_map_function:Nc` Used to allow a mapping to choose the outcome using a passed option.

```
\cs_generate_variant:Nn \seq_map_function:NN { Nc }
```

(*End definition for* `\seq_map_function:Nc`*. This function is documented on page* ??.)

The default hyphenation character should be set and hyphenation should be enabled.

```
⟨*initex⟩
\tex_defaulthyphenchar:D 45 \scan_stop:
⟨/initex⟩
```

7.2 Scratch variables

`\l__galley_tmp_int` `\l__galley_tmpa_seq` `\l__galley_tmpb_seq` `\l__galley_tmpa_tl` Used for manipulating cutouts and paragraph shapes.

```
\int_new:N \l__galley_tmp_int
\seq_new:N \l__galley_tmpa_seq
\seq_new:N \l__galley_tmpb_seq
\tl_new:N \l__galley_tmp_tl
```

(*End definition for* `\l__galley_tmp_int`*. This function is documented on page* ??.)

7.3 Galley settings

Settings for application by the galley respect the usual TEX grouping and so are all local variables.

`\l__galley_parshape_multipar_bool` Indicates whether the paragraph shape in use applies to one paragraph only or to more than one.

```
\bool_new:N \l__galley_parshape_multipar_bool
```

(*End definition for* `\l__galley_parshape_multipar_bool`*. This variable is documented on page* ??.)

\l__galley_parshape_left_indent_seq
\l__galley_parshape_right_indent_seq

Used to convert user input comma lists into sequences for mapping, and also to keep the information on paragraph shape available for the case where a cutout is also active.

```
\seq_new:N \l__galley_parshape_left_indent_seq
\seq_new:N \l__galley_parshape_right_indent_seq
```

(*End definition for* `\l__galley_parshape_left_indent_seq` *and* `\l__galley_parshape_right_indent_seq`. *These variables are documented on page* ??.)

`\l_galley_text_width_dim`

The width of the current measure: the "running" setting can be inherited from LaTeX 2_ε.

```
⟨*initex⟩
\dim_new:N \l_galley_text_width_dim
⟨/initex⟩
⟨*package⟩
\cs_new_eq:NN \l_galley_text_width_dim \linewidth
⟨/package⟩
```

(*End definition for* `\l_galley_text_width_dim`. *This variable is documented on page 10.*)

\l_galley_total_left_margin_dim
\l_galley_total_right_margin_dim

Margins of the current text within the galley: these plus the galley width are one way to define the measure width. See also the text width, which is an alternative view (and should be in sync with this one!).

```
⟨*initex⟩
\dim_new:N \l_galley_total_left_margin_dim
⟨/initex⟩
⟨*package⟩
\cs_new_eq:NN \l_galley_total_left_margin_dim \@totalleftmargin
⟨/package⟩
\dim_new:N \l_galley_total_right_margin_dim
```

(*End definition for* `\l_galley_total_left_margin_dim` *and* `\l_galley_total_right_margin_dim`. *These variables are documented on page 10.*)

\l_galley_interpar_penalty_int
\l_galley_interpar_vspace_skip

Items between paragraphs at the design level.

```
\int_new:N \l_galley_interpar_penalty_int
\skip_new:N \l_galley_interpar_vspace_skip
```

(*End definition for* `\l_galley_interpar_penalty_int` *and* `\l_galley_interpar_vspace_skip`. *These variables are documented on page 3.*)

`\l_galley_width_dim`

The external size of a galley is the stored in the TeX primitive `\tex_hsize:D`, which is renamed. This will only ever be reset by the code constructing a new galley, for example the start of a minipage. This value will be set for the main galley by the page layout system.

```
\cs_new_eq:NN \l_galley_width_dim \tex_hsize:D
```

(*End definition for* `\l_galley_width_dim`. *This variable is documented on page 2.*)

7.4 Galley data structures

In contrast to settings, the data structures used by the galley are all set globally. To allow different galley levels to exist, a local variant is defined for each one to save the value when starting a new level.

\g__galley_begin_level_bool
\l__galley_begin_level_bool

Indicates that the galley is at the very beginning of the level, and that no material has yet been set. As a result, the global version is set `true` to begin with.

```
\bool_new:N \g__galley_begin_level_bool
\bool_new:N \l__galley_begin_level_bool
```

(*End definition for* `\g__galley_begin_level_bool` *and* `\l__galley_begin_level_bool`*. These variables are documented on page* **??**.)

\g__galley_cutout_left_seq
\l__galley_cutout_left_seq
\g__galley_cutout_right_seq
\l__galley_cutout_right_seq

To apply cutouts across paragraphs, they need to be tracked. The information has to be global, so is handled here.

```
\seq_new:N \g__galley_cutout_left_seq
\seq_new:N \l__galley_cutout_left_seq
\seq_new:N \g__galley_cutout_right_seq
\seq_new:N \l__galley_cutout_right_seq
```

(*End definition for* `\g__galley_cutout_left_seq` *and others. These variables are documented on page* **??**.)

\g_galley_omit_next_indent_bool
\l__galley_omit_next_indent_bool

A global flag is needed for suppressing indentation of the next paragraph. This does not need a "running" version since that should be handled using the `justification` object: the two concepts are related but not identical. The flag here is needed in cases such as the very first paragraph in a galley or immediately following a heading.

```
\bool_new:N \g_galley_omit_next_indent_bool
\bool_new:N \l__galley_omit_next_indent_bool
```

(*End definition for* `\g_galley_omit_next_indent_bool` *and* `\l__galley_omit_next_indent_bool`*. These variables are documented on page* **??**.)

\g_galley_no_break_next_bool
\l__galley_no_break_next_bool

Dealing with the no-break flag is pretty much the same as the case for the indent: this applies on a single paragraph basis.

```
\bool_new:N \g_galley_no_break_next_bool
\bool_new:N \l__galley_no_break_next_bool
```

(*End definition for* `\g_galley_no_break_next_bool` *and* `\l__galley_no_break_next_bool`*. These variables are documented on page* **??**.)

\g_galley_par_begin_hook_tl
\l__galley_par_begin_hook_tl
\g_galley_par_end_hook_tl
\l__galley_par_end_hook_tl

Hooks for user-level code: these are not used by the galley itself.

```
\tl_new:N \g_galley_par_begin_hook_tl
\tl_new:N \l__galley_par_begin_hook_tl
\tl_new:N \g_galley_par_end_hook_tl
\tl_new:N \l__galley_par_end_hook_tl
```

(*End definition for* `\g_galley_par_begin_hook_tl` *and others. These variables are documented on page* **??**.)

\g_galley_par_reset_hook_tl
\l__galley_par_reset_hook_tl

This one is used by the galley: it happens "after" the current paragraph, and is used for reset purposes.

```
\tl_new:N \g_galley_par_reset_hook_tl
\tl_new:N \l__galley_par_reset_hook_tl
```

(*End definition for* `\g_galley_par_reset_hook_tl` *and* `\l__galley_par_reset_hook_tl`. *These variables are documented on page* **??**.)

\g_galley_previous_par_lines_int
\l__galley_previous_par_lines_int
\g__galley_current_par_lines_int
\l__galley_current_par_lines_int

The number of lines in the previous typeset paragraph. This is reset at the start of the paragraph and *added to* when each `\tex_par:D` primitive is used: LaTeX uses the primitive in places that do not end a (conceptual) paragraph. There is also a set of variables to deal with the current (conceptual) paragraph, so that the information can be build up correctly.

```
\int_new:N \g_galley_previous_par_lines_int
\int_new:N \l__galley_previous_par_lines_int
\int_new:N \g__galley_current_par_lines_int
\int_new:N \l__galley_current_par_lines_int
```

(*End definition for* `\g_galley_previous_par_lines_int` *and* `\l__galley_previous_par_lines_int`. *These functions are documented on page* **??**.)

\g_galley_restore_running_tl
\l__galley_restore_running_tl

When a parameter is altered from the "running" value to a different "current" one, there needs to be a method to restore the "running" value. This is done by adding the necessary assignment to a token list, which can be executed when needed. At the same time, this information is itself part of the galley parameter structure, and so there has to be a local save version.

```
\tl_new:N \g_galley_restore_running_tl
\tl_new:N \l__galley_restore_running_tl
```

(*End definition for* `\g_galley_restore_running_tl` *and* `\l__galley_restore_running_tl`. *These variables are documented on page* **??**.)

\g_galley_whatsit_next_tl
\l__galley_whatsit_next_tl
\g_galley_whatsit_previous_tl
\l__galley_whatsit_previous_tl

Whatsits only apply on a per-paragraph basis and so there is no need to differentiate between current and running values. However, there is a need to differentiate between whatsits that attach to the previous (completed) paragraph and those that attach to the next paragraph.

```
\tl_new:N \g_galley_whatsit_next_tl
\tl_new:N \l__galley_whatsit_next_tl
\tl_new:N \g_galley_whatsit_previous_tl
\tl_new:N \l__galley_whatsit_previous_tl
```

(*End definition for* `\g_galley_whatsit_next_tl` *and others. These variables are documented on page* **??**.)

\g__galley_interpar_penalty_user_tl
\l__galley_interpar_penalty_user_tl

The user may want to over-ride the penalty for a break between paragraphs, for example to prevent a break when the overall design allows one. This is handled using an additional penalty.

```
\tl_new:N \g__galley_interpar_penalty_user_tl
\tl_new:N \l__galley_interpar_penalty_user_tl
```

(*End definition for* `\g__galley_interpar_penalty_user_tl` *and* `\l__galley_interpar_penalty_user_tl`. *These variables are documented on page* **??**.)

\g__galley_interpar_vspace_user_tl
\l__galley_interpar_vspace_user_tl

Arbitrary vertical space can be inserted by the user on a one-off basis. This is used in place of any running space between paragraphs.

```
\tl_new:N \g__galley_interpar_vspace_user_tl
\tl_new:N \l__galley_interpar_vspace_user_tl
```

(*End definition for* `\g__galley_interpar_vspace_user_tl` *and* `\l__galley_interpar_vspace_user_tl`. *These variables are documented on page* **??**.)

7.5 Independent galley levels

As well as the main vertical list, independent galleys are required for items such as minipages and marginal notes. Each of these galleys requires an independent set of global data structures. This is achieved by storing the data structures in *local* variables. The later are only used to save and restore the global value, and so TEX grouping will manage the values correctly. This implies that each galley level must form a group: galley levels are tied to vertical boxes and so this is a reasonable requirements.

__galley_initialise_variables:

At the start of a galley level, both the global and local variables will need to be reset to standard values. For example, the measure is set to the galley width and any paragraph shape is cleared.

```
\cs_new_protected_nopar:Npn \__galley_initialise_variables:
  {
    \bool_gset_true:N \g__galley_begin_level_bool
    \seq_gclear:N \g__galley_cutout_left_seq
    \seq_gclear:N \g__galley_cutout_right_seq
    \tl_gclear:N \g__galley_interpar_penalty_user_tl
    \tl_gclear:N \g__galley_interpar_vspace_user_tl
    \bool_gset_true:N \g_galley_omit_next_indent_bool
    \bool_gset_false:N \g_galley_no_break_next_bool
    \tl_gclear:N \g_galley_par_begin_hook_tl
    \tl_gclear:N \g_galley_par_end_hook_tl
    \tl_gclear:N \g_galley_par_reset_hook_tl
    \int_gzero:N \g__galley_current_par_lines_int
    \int_gzero:N \g_galley_previous_par_lines_int
    \tl_gclear:N \g_galley_restore_running_tl
    \tl_gclear:N \g_galley_whatsit_previous_tl
    \tl_gclear:N \g_galley_whatsit_next_tl
  }
\__galley_initialise_variables:
```

(*End definition for* `\__galley_initialise_variables:`. *This function is documented on page* **??**.)

__galley_initialise_settings:

This sets the local values of the various galley settings.

```
\cs_new_protected_nopar:Npn \__galley_initialise_settings:
  {
    \dim_set_eq:NN \l_galley_text_width_dim \l_galley_width_dim
    \dim_zero:N \l_galley_total_left_margin_dim
    \dim_zero:N \l_galley_total_right_margin_dim
  }
```

(*End definition for* `\__galley_initialise_settings:`. *This function is documented on page* **??**.)

`\__galley_save_parameters:`
`\__galley_restore_parameters:`

Saving and restoring parameters is carried out by a series of copy functions.

```
\cs_new_protected_nopar:Npn \__galley_save_parameters:
  {
    \bool_set_eq:NN \l__galley_begin_level_bool
      \g__galley_begin_level_bool
    \seq_set_eq:NN \l__galley_cutout_left_seq
      \g__galley_cutout_left_seq
    \seq_set_eq:NN \l__galley_cutout_right_seq
      \g__galley_cutout_right_seq
    \tl_set_eq:NN \l__galley_interpar_penalty_user_tl
      \g__galley_interpar_penalty_user_tl
    \tl_set_eq:NN \l__galley_interpar_vspace_user_tl
      \g__galley_interpar_vspace_user_tl
    \bool_set_eq:NN \l__galley_omit_next_indent_bool
      \g_galley_omit_next_indent_bool
    \bool_set_eq:NN \l__galley_no_break_next_bool
      \g_galley_no_break_next_bool
    \tl_set_eq:NN \l__galley_par_begin_hook_tl
      \g_galley_par_begin_hook_tl
    \tl_set_eq:NN \l__galley_par_end_hook_tl
      \g_galley_par_end_hook_tl
    \tl_set_eq:NN \l__galley_par_reset_hook_tl
      \g_galley_par_reset_hook_tl
    \int_set_eq:NN \l__galley_current_par_lines_int
      \g__galley_current_par_lines_int
    \int_set_eq:NN \l__galley_previous_par_lines_int
      \g_galley_previous_par_lines_int
    \tl_set_eq:NN \l__galley_restore_running_tl
      \g_galley_restore_running_tl
    \tl_set_eq:NN \l__galley_whatsit_previous_tl
      \g_galley_whatsit_previous_tl
    \tl_set_eq:NN \l__galley_whatsit_next_tl
      \g_galley_whatsit_next_tl
  }
\cs_new_protected_nopar:Npn \__galley_restore_parameters:
  {
    \bool_gset_eq:NN \g__galley_begin_level_bool
      \l__galley_begin_level_bool
    \seq_gset_eq:NN \g__galley_cutout_left_seq
      \l__galley_cutout_left_seq
    \seq_gset_eq:NN \g__galley_cutout_right_seq
      \l__galley_cutout_right_seq
    \tl_gset_eq:NN \g__galley_interpar_penalty_user_tl
      \l__galley_interpar_penalty_user_tl
    \tl_gset_eq:NN \g__galley_interpar_vspace_user_tl
      \l__galley_interpar_vspace_user_tl
    \bool_gset_eq:NN \g_galley_omit_next_indent_bool
      \l__galley_omit_next_indent_bool
    \bool_gset_eq:NN \g_galley_no_break_next_bool
```

```
    \l__galley_no_break_next_bool
  \tl_gset_eq:NN \g_galley_par_begin_hook_tl
    \l__galley_par_begin_hook_tl
  \tl_gset_eq:NN \g_galley_par_end_hook_tl
    \l__galley_par_end_hook_tl
  \tl_gset_eq:NN \g_galley_par_reset_hook_tl
    \l__galley_par_reset_hook_tl
  \int_gset_eq:NN \g__galley_current_par_lines_int
    \l__galley_current_par_lines_int
  \int_gset_eq:NN \g_galley_previous_par_lines_int
    \l__galley_previous_par_lines_int
  \tl_gset_eq:NN \g_galley_restore_running_tl
    \l__galley_restore_running_tl
  \tl_gset_eq:NN \g_galley_whatsit_previous_tl
    \l__galley_whatsit_previous_tl
  \tl_gset_eq:NN \g_galley_whatsit_next_tl
    \l__galley_whatsit_next_tl
}
```

(*End definition for* `\__galley_save_parameters:` *and* `\__galley_restore_parameters:`*. These functions are documented on page* **??**.)

`\galley_level:`
`\__galley_level_end:`

Galley levels are created by saving all of the current global settings, starting a group then initialising both the local and global variables.

```
\cs_new_protected_nopar:Npn \galley_level:
  {
    \__galley_save_parameters:
    \group_begin:
      \__galley_initialise_variables:
      \__galley_initialise_settings:
      \group_insert_after:N \__galley_level_end:
  }
```

At the end of the level, the global values are restored using the saved *local* versions, hence the position of the close-of-group instruction. As this code can be inserted automatically, at the point of use only the start of a galley level needs to be marked up: the end must come in a fixed location. All of this relies on the "color safe" group used inside a box.

```
\cs_new_protected_nopar:Npn \__galley_level_end:
  {
      \par
      \__galley_restore_parameters:
    \group_end:
  }
```

(*End definition for* `\galley_level:`*. This function is documented on page* 2.)

7.6 The \par token

`\s__par_omit`

Used to indicate that a paragraph should be omitted.

```
\__scan_new:N \s__par_omit
```

(*End definition for* `\s__par_omit`*. This variable is documented on page* **??**.)

\galley_par:
__galley_par_auxi:
__galley_par_auxii:
__galley_par_aux:N

The idea here is to expand the next token in exactly the same way as TEX would do anyway. The `f`-type expansion will ignore any protection, but will stop at a scan marker. Thus the code can test for an "omit paragraph" marker.

```
\cs_new_protected_nopar:Npn \galley_par:
  {
    \s__par_omit
    \exp_after:wN \__galley_par_auxi: \tex_romannumeral:D - `0
  }
\cs_new_protected:Npn \__galley_par_auxi:
  {
    \peek_meaning:NTF \s__par_omit
      { \__galley_par_aux:N }
      { \__galley_par_auxii: }
  }
\cs_new_protected:Npn \__galley_par_aux:N #1
  {
    \str_if_eq_x:nnF {#1} { \s__par_omit }
      {
        \__galley_par_auxii:
        #1
      }
  }
```

No marker, so really insert a paragraph: the `\tex_par:D` is inside a group to preserve some dynamic settings (for example `\etex_interlinepenalties:D`). In vertical mode, that means just inserting the primitive.

```
\cs_new_protected_nopar:Npn \__galley_par_auxii:
  {
    \mode_if_vertical:TF
      {
        \group_begin:
          \tex_par:D
        \group_end:
      }
```

In horizontal mode, the paragraph shape is set "just in time" before inserting `\tex_par:D`. Once the paragraph has been typeset, the number of lines is *added* to the running total. It's possible that the conceptual paragraph contains display-like material, and simply setting the number of lines equal to `\tex_prevgraf:D` would "lose" these.

```
      {
        \g_galley_par_end_hook_tl
        \group_begin:
          \tex_par:D
        \group_end:
        \int_gset:Nn \g_galley_previous_par_lines_int
          { \tex_prevgraf:D + \g__galley_current_par_lines_int }
        \__galley_parshape_reset:
        \int_gzero:N \g__galley_current_par_lines_int
      }
    \g_galley_par_reset_hook_tl
```

```
    \tl_gclear:N \g_galley_par_reset_hook_tl
```

The non-breaking penalty is needed here as within the `\tex_everypar:D` hook there is an additional `\tex_par:D`. This leads to an extra `\tex_parskip:D`, which will leave an unwanted break-point here otherwise.

```
    \tex_penalty:D \c_ten_thousand
  }
```

(*End definition for* `\galley_par:`. *This function is documented on page* 9.)

`\galley_par:n` Inserts tokens such that they are appended to the end of the last paragraph, using the paragraph-omitting system.

```
\cs_new_protected:Npn \galley_par:n #1
  {
    \s__par_omit
    \bool_if:nF \g__galley_begin_level_bool
      {
        #1
        \galley_par:
      }
  }
```

(*End definition for* `\galley_par:n`. *This function is documented on page* 9.)

`\par` The meaning of the token `\par` itself starts off as a standard paragraph.

```
\cs_set_protected_nopar:Npn \par { \galley_par: }
```

(*End definition for* `\par`. *This function is documented on page* ??.)

`\@par` LaTeX 2_ε requires a "long term" version of `\par`, which is stored as `\@par`. Things are done a bit differently by LaTeX3 and so this will only be needed in package mode.

```
⟨*package⟩
\tl_set:Nn \@par { \galley_par: }
⟨/package⟩
```

(*End definition for* `\@par`. *This function is documented on page* ??.)

7.7 Display levels

`\galley_display_begin:` `\galley_display_end:` `\__galley_display_penalty:N` `\__galley_display_vspace:N` `\__galley_display_par_setup:` `\__galley_display_par:`

Display items within the galley are a bit like galley levels: they may have different paragraph settings to the main part of the galley. On the other hand, unlike independent galleys they should inherit the settings from the surrounding material. They may also start and end with special spacing values.

```
\cs_new_protected_nopar:Npn \galley_display_begin:
  {
    \group_begin:
      \__galley_save_parameters:
      \mode_if_vertical:TF
        {
          \__galley_display_penalty:N \l_galley_display_begin_par_penalty_tl
          \__galley_display_vspace:N  \l_galley_display_begin_par_vspace_tl
        }
```

```
        {
          \__galley_display_penalty:N \l_galley_display_begin_penalty_tl
          \__galley_display_vspace:N  \l_galley_display_begin_vspace_tl
        }
      \par
  }
```

Two short-cuts for setting up any special penalty or vertical space. The idea is that the standard value is saved to the "restore" token list, before setting up the value to the special value needed in this one case.

```
\cs_new_protected:Npn \__galley_display_penalty:N #1
  {
     \tl_if_empty:NF #1
       {
         \tl_gput_right:Nx \g_galley_restore_running_tl
           {
             \int_gset:Nn \exp_not:N \g_galley_penalty_int
               { \int_use:N \g_galley_penalty_int }
           }
         \int_gset:Nn \g_galley_penalty_int {#1}
       }
  }
\cs_new_protected:Npn \__galley_display_vspace:N #1
  {
     \tl_if_empty:NF #1
       {
         \tl_gput_right:Nx \g_galley_restore_running_tl
           {
             \skip_gset:Nn \exp_not:N \g_galley_vspace_skip
               { \skip_use:N \g_galley_vspace_skip }
           }
         \skip_gset:Nn \g_galley_vspace_int {#1}
       }
  }
```

The `\par` token at the end of the display needs to go in at the same group level as the text, hence this function cannot be placed using `\group_insert_after:N`. Resetting the meaning of the `\par` token needs to be carried out after the group used for the environment.

```
\cs_new_protected_nopar:Npn \galley_display_end:
  {
      \par
      \__galley_restore_parameters:
    \group_end:
    \group_insert_after:N \__galley_display_par_setup:
  }
```

The method used here is to assume that the next piece of horizontal mode material will follow on from the displayed output without an intervening `\par` token (probably a blank

line). The meaning of the \par token is then altered so that a check can be made to see if this assumption was correct.

```
\cs_new_protected_nopar:Npn \__galley_display_par_setup:
  {
    \bool_gset_false:N \g_galley_omit_next_indent_bool
    \cs_set_eq:NN \par \__galley_display_par:
  }
```

The "special" meaning of the paragraph token starts by putting things back to normal: there should never need to be more than one special paragraph marker in one group. If TEX is in vertical mode, then there has been a paragraph token inserted, most likely by a blank line. Thus the next piece of material is a separate conceptual paragraph from the display. In that case, the assumption from above is undone and the indent is turned back on. On the other hand, for the case where TEX is in horizontal mode then a \tex_par:D primitive is required in the same way as in \galley_standard_par:.

```
\cs_new_protected_nopar:Npn \__galley_display_par:
  {
    \cs_set_eq:NN \par \galley_par:
    \mode_if_vertical:TF
      {
        \par
        \bool_gset_false:N \g_galley_omit_next_indent_bool
        \__galley_display_penalty:N \l_galley_display_end_par_penalty_tl
        \__galley_display_vspace:N  \l_galley_display_end_par_vspace_tl
      }
      {
        \group_begin:
          \tex_par:D
        \group_end:
        \int_gadd:Nn \g__galley_current_par_lines_int \tex_prevgraf:D
        \__galley_display_penalty:N \l_galley_display_end_penalty_tl
        \__galley_display_vspace:N  \l_galley_display_end_vspace_tl
      }
  }
```

(*End definition for* \galley_display_begin: *and* \galley_display_end:. *These functions are documented on page* 6.)

7.8 Insertions using \tex_everypar:D

The key to the entire galley mechanism is hooking into the \tex_everypar:D token register. This requires that the original is moved out of the way, with appropriate hooks left attached for further modification by other modules and by the user. This is all done such that there is no danger of accidentally deactivating the galley mechanism.

\everypar When used on top of LATEX 2ε the original primitive name needs to be available without the risk of completely overwriting the new mechanism. This is implemented as a token register in case low-level TEX is used. The TEX primitive is set here as otherwise the

LaTeX 2_ε `\@nodocument` is never removed from the register. This precaution is not be needed for a stand-alone format.

```
⟨*initex⟩
\tex_everypar:D % TEMP
  {
    \bool_if:NTF \g__galley_begin_level_bool
      { \__galley_start_paragraph_first: }
      { \__galley_start_paragraph_std: }
  }
⟨/initex⟩
⟨*package⟩
\cs_undefine:N \everypar
\newtoks \everypar
\AtBeginDocument
  {
    \tex_everypar:D
      {
        \bool_if:NTF \g__galley_begin_level_bool
          { \__galley_start_paragraph_first: }
          { \__galley_start_paragraph_std: }
        \tex_the:D \everypar
      }
  }
⟨/package⟩
```

(*End definition for* `\everypar`. *This function is documented on page* **??**.)

7.9 The galley mechanism

`\g__galley_last_box` A temporary box to hold the box inserted by TeX when a paragraph is inserted with an indent. The galley actually inserts the space (*i.e.* `\tex_parindent:D` is globally zero), but there is still an empty box to test for.

```
\box_new:N \g__galley_last_box
```

(*End definition for* `\g__galley_last_box`. *This variable is documented on page* **??**.)

The "start of paragraph" routines are fired by `\tex_everypar:D`. This can take place within a group in a construction such as

```
... end of last par.

{\Large Start} of par
```

and so anything applied here must be done globally.

`\__galley_start_paragraph_std:` The routine at the start of a paragraph starts by removing any (empty) indent box from the vertical list. As there may be vertical mode items still to insert, a `\tex_-par:D` primitive is used to get back into vertical mode before they are tidied up. To get back again to horizontal mode, `\tex_noindent:D` can be used. To avoid an infinite loop, `\tex_everypar:D` is locally cleared before doing that. Back in horizontal mode,

the horizontal mode items can be tidied up before sorting out any items which have been set on a single-paragraph basis.

```
\cs_new_protected_nopar:Npn \__galley_start_paragraph_std:
  {
    \group_begin:
      \box_gset_to_last:N \g__galley_last_box
      \tex_par:D
      \__galley_insert_vertical_items:
      \tex_everypar:D { }
      \tex_noindent:D
    \group_end:
    \__galley_insert_horizontal_items:
    \__galley_restore_running_parameters:
  }
```

(*End definition for* `\__galley_start_paragraph_std:`.)

`\__galley_start_paragraph_first:` For the very first paragraph in a galley, the code needs to avoid adding any unnecessary vertical items at the top as it will interfere with vertical positioning in `\tex_vtop:D`.

```
\cs_new_protected_nopar:Npn \__galley_start_paragraph_first:
  {
    \bool_gset_false:N \g__galley_begin_level_bool
    \mode_if_horizontal:TF
      {
        \group_begin:
          \box_gset_to_last:N \g__galley_last_box
          \tex_par:D
          \__galley_insert_vspace:
          \tex_everypar:D { }
          \tex_noindent:D
        \group_end:
      }
      { \__galley_insert_vspace: }
    \__galley_insert_horizontal_items:
    \__galley_restore_running_parameters:
  }
```

(*End definition for* `\__galley_start_paragraph_first:`.)

`\__galley_insert_vertical_items` `\__galley_insert_vspace:` The aim here is to insert the vertical items such that they attach to the correct place. This function is used as part of the `\tex_everypar:D` mechanism, meaning that the immediately-preceding item on the vertical list is the `\tex_parskip:D`, always zero-length but an implicit penalty. So any whatsits "attached" to the previous paragraph should stay glued on. After the whatsits, a penalty for breaking will be inserted. This will be the user penalty if supplied, or the running penalty unless the no-break flag is set. Finally, the inter-paragraph space is applied.

```
\cs_new_protected_nopar:Npn \__galley_insert_vertical_items:
  {
    \g_galley_whatsit_previous_tl
    \tl_gclear:N \g_galley_whatsit_previous_tl
```

```
    \tl_if_empty:NTF \g__galley_interpar_penalty_user_tl
      {
        \bool_if:NTF \g_galley_no_break_next_bool
          { \tex_penalty:D \c_ten_thousand }
          { \tex_penalty:D \l_galley_interpar_penalty_int }
      }
      {
        \tex_penalty:D
          \__int_eval:w \g__galley_interpar_penalty_user_tl \__int_eval_end:
        \tl_gclear:N \g__galley_interpar_penalty_user_tl
      }
    \bool_gset_false:N \g_galley_no_break_next_bool
    \__galley_insert_vspace:
  }
```

Inserting vertical space is set up as a separate function as it comes up in a few places. The idea here is that any user-set space will override the design value, and only one space is ever inserted.

```
\cs_new_protected_nopar:Npn \__galley_insert_vspace:
  {
    \tl_if_empty:NTF \g__galley_interpar_vspace_user_tl
      { \skip_vertical:N \l_galley_interpar_vspace_skip }
      {
        \skip_vertical:n { \g__galley_interpar_vspace_user_tl }
        \tl_gclear:N \g__galley_interpar_vspace_user_tl
      }
  }
```

(*End definition for* `\__galley_insert_vertical_items` *and* `\__galley_insert_vspace:`.)

`\__galley_insert_horizontal_items:` Horizontal mode objects start with the whatsits for the next paragraph. An indent is then included if the removed box was not void.

```
\cs_new_protected_nopar:Npn \__galley_insert_horizontal_items:
  {
    \g_galley_whatsit_next_tl
    \tl_gclear:N \g_galley_whatsit_next_tl
    \bool_if:NF \g_galley_omit_next_indent_bool
      {
        \box_if_empty:NF \g__galley_last_box
          { \hbox_to_wd:nn \l_galley_par_indent_dim { } }
      }
    \skip_horizontal:N \l_galley_par_begin_skip
    \g_galley_par_begin_hook_tl
    \bool_gset_false:N \g_galley_omit_next_indent_bool
  }
```

(*End definition for* `\__galley_insert_horizontal_items:`.)

`\__galley_restore_running_parameters:` Restoring the ongoing parameters just means using the token list variable in which the appropriate assignments are stored. The list can then be cleared.

```
\cs_new_protected_nopar:Npn \__galley_restore_running_parameters:
```

```
  {
    \g_galley_restore_running_tl
    \tl_gclear:N \g_galley_restore_running_tl
  }
```

(*End definition for* `\__galley_restore_running_parameters:`.)

7.10 Measure

`\l__galley_total_left_margin_dim`
`\l__galley_total_right_margin_dim`

Used to set the measure, first by providing a place to save the existing values, then allowing calculation of the difference between old and new settings.

```
\dim_new:N \l__galley_total_left_margin_dim
\dim_new:N \l__galley_total_right_margin_dim
```

(*End definition for* `\l__galley_total_left_margin_dim` *and* `\l__galley_total_right_margin_dim`. *These variables are documented on page* **??**.)

`\galley_margins_set_absolute:nn`
`\galley_margins_set_relative:nn`
`\__galley_save_margins:`
`\__galley_set_measure:`
`\__galley_set_measure_aux:n`

Setting the measure is just a question of adjusting margins, either in a relative or absolute sense.

```
\cs_new_protected:Npn \galley_margins_set_absolute:nn #1#2
  {
    \__galley_save_margins:
    \dim_set:Nn \l_galley_total_left_margin_dim  {#1}
    \dim_set:Nn \l_galley_total_right_margin_dim {#2}
    \dim_set:Nn \l_galley_text_width_dim
      {
          \l_galley_width_dim
        - \l_galley_total_left_margin_dim
        - \l_galley_total_right_margin_dim
      }
    \__galley_set_measure:
  }
\cs_new_protected:Npn \galley_margins_set_relative:nn #1#2
  {
    \__galley_save_margins:
    \dim_add:Nn \l_galley_total_left_margin_dim  {#1}
    \dim_add:Nn \l_galley_total_right_margin_dim {#2}
    \dim_set:Nn \l_galley_text_width_dim
      {
          \l_galley_width_dim
        - \l_galley_total_left_margin_dim
        - \l_galley_total_right_margin_dim
      }
    \__galley_set_measure:
  }
```

Saves the previous margin state so that it can be used for calculation during the update.

```
\cs_new_protected_nopar:Npn \__galley_save_margins:
  {
    \dim_set_eq:NN \l__galley_total_left_margin_dim
      \l_galley_total_left_margin_dim
```

```
    \dim_set_eq:NN \l__galley_total_right_margin_dim
      \l_galley_total_right_margin_dim
  }
```

Setting the measure first requires a quick test to see if there is any existing shape: if not, just use the values directly. Assuming we do need to allow for the existing shape, the idea is to preserve whatever shape has already been applied and adjust the edges. This can be done by iterating though the existing shape, recovering the values and applying the changes.

```
\cs_new_protected_nopar:Npn \__galley_set_measure:
  {
    \int_compare:nNnTF \tex_parshape:D = \c_zero
      { \__galley_parshape_measure: }
      {
        \dim_sub:Nn \l__galley_total_left_margin_dim
          \l_galley_total_left_margin_dim
        \dim_sub:Nn \l__galley_total_right_margin_dim
          \l_galley_total_right_margin_dim
        \tex_parshape:D
          \tex_parshape:D
          \int_step_function:nnnN
            \c_one \c_one \tex_parshape:D
            \__galley_set_measure_aux:n
      }
  }
\cs_new:Npn \__galley_set_measure_aux:n #1
  {
    \__dim_eval:w
      \etex_parshapeindent:D #1 - \l__galley_total_left_margin_dim
    \__dim_eval_end:
    \__dim_eval:w
      \etex_parshapelength:D #1
        + \l__galley_total_left_margin_dim + \l__galley_total_right_margin_dim
    \__dim_eval_end:
  }
```

(*End definition for* `\galley_margins_set_absolute:nn` *and* `\galley_margins_set_relative:nn`. *These functions are documented on page 3.*)

`\__galley_parshape_reset:`

Test for paragraph shapes to be removed: a multi-step procedure! First, check to see if there is a shape at all. If there is, the second check looks for cutouts. If there are no cutouts to worry about, the final check is whether the current shape applies to one paragraph or on an ongoing sense.

```
\cs_new_protected_nopar:Npn \__galley_parshape_reset:
  {
    \int_compare:nNnF \tex_parshape:D = \c_zero
      {
        \bool_if:nTF
          {
               \seq_if_empty_p:N \g__galley_cutout_left_seq
```

```
      && \seq_if_empty_p:N \g__galley_cutout_right_seq
    }
    {
      \bool_if:NF \l__galley_parshape_multipar_bool
        { \__galley_parshape_measure: }
    }
```

If there was a cutout active, the "done" part is removed from the tracking sequences. The "normal" paragraph shape is then reapplied before removing the cutout parts. This is done using separate loops as in general that is as easy as trying to work out all of the parts in advance (the same number of loops would still be needed).

```
    {
      \prg_replicate:nn \g_galley_previous_par_lines_int
        {
          \seq_gpop:NN \g__galley_cutout_left_seq \l__galley_tmp_tl
          \seq_gpop:NN \g__galley_cutout_right_seq \l__galley_tmp_tl
        }
      \bool_if:NTF \l__galley_parshape_multipar_bool
        { \__galley_parshape_set: }
        { \__galley_parshape_measure: }
      \__galley_cutout_set:Nn \g__galley_cutout_left_seq { left }
      \__galley_cutout_set:Nn \g__galley_cutout_right_seq { right }
    }
  }
}
```

(*End definition for* `\__galley_parshape_reset:`*. This function is documented on page* **??**.)

`\__galley_parshape_measure:` Sets the parshape to the full measure at the current time: used in a few places, so worth spinning out.

```
\cs_new_protected_nopar:Npn \__galley_parshape_measure:
  {
    \tex_parshape:D
      \c_one
      \l_galley_total_left_margin_dim
      \l_galley_text_width_dim
  }
```

(*End definition for* `\__galley_parshape_measure:`.)

7.11 Paragraph shape

`\galley_parshape_set_multi:nnnN`
`\galley_parshape_set_multi:nVVN`
`\galley_parshape_set_single:nnnN`
`\galley_parshape_set_single:nVVN`

Setting the paragraph shape is mainly a question of converting the input. First, though, a flag is set.

```
\cs_new_protected_nopar:Npn \galley_parshape_set_multi:nnnN
  {
    \bool_set_true:N \l__galley_parshape_multipar_bool
    \__galley_parshape_set:nnnN
  }
\cs_new_protected_nopar:Npn \galley_parshape_set_single:nnnN
  {
```

```
    \bool_set_false:N \l__galley_parshape_multipar_bool
    \__galley_parshape_set:nnnN
  }
\cs_generate_variant:Nn \galley_parshape_set_multi:nnnN  { nVV }
\cs_generate_variant:Nn \galley_parshape_set_single:nnnN { nVV }
```

(*End definition for* `\galley_parshape_set_multi:nnnN` *and others. These functions are documented on page* **??**.)

`\__galley_parshape_set:nnnN`
`\__galley_parshape_set:`
`\__galley_parshape_set:nn`

Setting the paragraph shape starts by converting the two input lists into sequences. The shape is then set using a mapping with allowance for the unaltered lines and the possibility of resuming the measure. The unaltered lines are added to the sequences as this information may be needed elsewhere (if a cutout is active), and this is the most convenient way to deal with it. Everything ends up stored in the two sequences for possible re-application if there are cutouts.

```
\cs_new_protected:Npn \__galley_parshape_set:nnnN #1#2#3#4
  {
    \seq_set_from_clist:Nn \l__galley_parshape_left_indent_seq {#2}
    \seq_set_from_clist:Nn \l__galley_parshape_right_indent_seq {#3}
    \prg_replicate:nn {#1}
      {
        \seq_put_left:Nn \l__galley_parshape_left_indent_seq { 0 pt }
        \seq_put_left:Nn \l__galley_parshape_right_indent_seq { 0 pt }
      }
    \bool_if:NT #4
      {
        \seq_put_right:Nn \l__galley_parshape_left_indent_seq { 0 pt }
        \seq_put_right:Nn \l__galley_parshape_right_indent_seq { 0 pt }
      }
    \__galley_parshape_set:
  }
\cs_new_protected_nopar:Npn \__galley_parshape_set:
  {
    \tex_parshape:D
      \__int_eval:w
        \int_min:nn
          { \seq_count:N \l__galley_parshape_left_indent_seq }
          { \seq_count:N \l__galley_parshape_right_indent_seq }
      \__int_eval_end:
      \seq_mapthread_function:NNN
        \l__galley_parshape_left_indent_seq
        \l__galley_parshape_right_indent_seq
        \__galley_parshape_set:nn
  }
\cs_new:Npn \__galley_parshape_set:nn #1#2
  {
    \__dim_eval:w \l_galley_total_left_margin_dim + ( #1 ) \__dim_eval_end:
    \__dim_eval:w
      \l_galley_text_width_dim - ( ( #1 ) + ( #2 ) )
    \__dim_eval_end:
```

```
}
```

(*End definition for* `\__galley_parshape_set:nnnN`. *This function is documented on page* **??**.)

7.12 Cutouts

Cutouts are another way of looking at paragraph shapes, but apply to a fixed number of lines within a galley. As such, they require separate handling from the measure and shape.

```
\galley_cutout_left:nn
\galley_cutout_right:nn
  \__galley_cutout:nnn
\__galley_cutout_store:nn
  \__galley_cutout_set:Nn
  \__galley_cutout_left:n
\__galley_cutout_right:n
  \__galley_cutout_end:n
```

Setting up cutouts on the two sides of the paragraph is more or less the same idea with one or two very specific points of difference. As such, the two interface functions both use the same implementation.

```
\cs_new_protected_nopar:Npn \galley_cutout_left:nn
  { \__galley_cutout:nnn { left } }
\cs_new_protected_nopar:Npn \galley_cutout_right:nn
  { \__galley_cutout:nnn { right } }
```

The approach in the main function is first to construct a single sequence which contains details of all of the altered lines. This sequence is then used by `\__galley_cutout_set:Nn`, which actually applies the cutout to the current paragraph shape. One that is done, information on the cutout just applied is merged with the global cutout store: this needs to allow for different cutout lengths. The best way to do that is to loop over whichever list is longer, build a new list of the result and the copy this back to the store.

```
\cs_new_protected:Npn \__galley_cutout:nnn #1#2#3
  {
    \seq_set_from_clist:Nn \l__galley_tmpa_seq {#3}
    \prg_replicate:nn {#2}
      { \seq_put_left:Nn \l__galley_tmpa_seq { 0 pt } }
    \__galley_cutout_set:Nn \l__galley_tmpa_seq {#1}
    \seq_clear:N \l__galley_tmpb_seq
    \int_compare:nNnTF
      { \seq_count:N \l__galley_tmpa_seq } >
        { \seq_count:c { g__galley_cutout_ #1 _ seq }  }
      {
        \seq_map_inline:Nn \l__galley_tmpa_seq
          {
            \__galley_cutout_store:nn
              { \seq_gpop:cNF { g__galley_cutout_ #1 _ seq } }
              {##1}
          }
      }
      {
        \seq_map_inline:cn { g__galley_cutout_ #1 _ seq }
          {
            \__galley_cutout_store:nn
              { \seq_pop:NNF \l__galley_tmpa_seq }
              {##1}
          }
      }
```

```
    \seq_gset_eq:cN { g__galley_cutout_ #1 _ seq } \l__galley_tmpb_seq
  }
\cs_new_protected:Npn \__galley_cutout_store:nn #1#2
  {
    #1 \l__galley_tmp_tl
      { \tl_set:Nn \l__galley_tmp_tl { 0 pt } }
    \seq_put_right:Nx \l__galley_tmpb_seq
      { \dim_eval:n { \l__galley_tmp_tl + #2 } }
  }
```

Actually applying a cutout is done by using the sequence containing the cutout shape plus the current paragraph shape to construct a token list of the required indents and line lengths. That is done in a non-expansion way to allow for the two different kinds of mapping (to the sequence and to the parshape). At the end of the process, there is a need to allow for the last line of `\parshape`: depending on any existing shape and the length of the cutout, this might be the measure or a repeat of the last values in the shape.

```
\cs_new_protected:Npn \__galley_cutout_set:Nn #1#2
  {
    \int_zero:N \l__galley_tmp_int
    \tl_clear:N \l__galley_tmp_tl
    \seq_map_function:Nc #1 { __galley_cutout_ #2 :n }
    \int_compare:nNnTF \tex_parshape:D > \l__galley_tmp_int
      {
        \int_step_function:nnnN
          { \l__galley_tmp_int + \c_one }
          \c_one
          \tex_parshape:D
          \__galley_cutout_end:n
      }
      {
        \int_incr:N \l__galley_tmp_int
        \int_compare:nNnTF \tex_parshape:D = \c_zero
          {
            \tl_put_right:Nn \l__galley_tmp_tl
             { \l__galley_total_left_margin_dim \l_galley_text_width_dim }
          }
          {
            \tl_put_right:Nx \l__galley_tmp_tl
              {
                \dim_eval:n { \etex_parshapeindent:D \tex_parshape:D }
                \c_space_tl
                \dim_eval:n { \etex_parshapelength:D \tex_parshape:D }
                \c_space_tl
              }
          }
      }
    \tex_parshape:D
      \l__galley_tmp_int
      \l__galley_tmp_tl
  }
```

The only place where the position of the cutout matters is in working out the indent and line lengths. That's achieved using two separate auxiliaries and selection by name.

```
\cs_new_protected:Npn \__galley_cutout_left:n #1
  {
    \int_incr:N \l__galley_tmp_int
    \tl_put_right:Nx \l__galley_tmp_tl
      {
        \int_compare:nNnTF \l__galley_tmp_int > \tex_parshape:D
          {
            \dim_eval:n { \l__galley_total_left_margin_dim + ( #1 ) }
            \c_space_tl
            \dim_eval:n { \l_galley_text_width_dim - ( #1 ) }
            \c_space_tl
          }
          {
            \dim_eval:n { \etex_parshapeindent:D \l__galley_tmp_int + ( #1 ) }
            \c_space_tl
            \dim_eval:n { \etex_parshapelength:D \l__galley_tmp_int - ( #1 ) }
            \c_space_tl
          }
      }
  }
\cs_new_protected:Npn \__galley_cutout_right:n #1
  {
    \int_incr:N \l__galley_tmp_int
    \tl_put_right:Nx \l__galley_tmp_tl
      {
        \int_compare:nNnTF \l__galley_tmp_int > \tex_parshape:D
          {
            \dim_use:N \l__galley_total_left_margin_dim
            \c_space_tl
            \dim_eval:n { \l_galley_text_width_dim - ( #1 ) }
            \c_space_tl
          }
          {
            \dim_eval:n { \etex_parshapeindent:D \l__galley_tmp_int }
            \c_space_tl
            \dim_eval:n { \etex_parshapelength:D \l__galley_tmp_int - ( #1 ) }
            \c_space_tl
          }
      }
  }
```

An auxiliary to "recycle" the paragraph shape which is already active *if* the cutout is shorter.

```
\cs_new_protected:Npn \__galley_cutout_end:n #1
  {
    \int_incr:N \l__galley_tmp_int
    \tl_put_right:Nx \l__galley_tmp_tl
      {
```

```
        \dim_eval:n { \etex_parshapeindent:D #1 }
        \c_space_tl
        \dim_eval:n { \etex_parshapelength:D #1 }
        \c_space_tl
      }
  }
```

(*End definition for* `\galley_cutout_left:nn` *and* `\galley_cutout_right:nn`*. These functions are documented on page* **??**.)

7.13 Between paragraphs

`\galley_penalty_set_single:n` `\galley_vspace_set_single:n`

User supplied penalties and spaces only apply for a single paragraph. In both cases, the input values need to be checked for the correct form but are stored as token lists. The x-type expansion deals with this nicely.

```
\cs_new_protected:Npn \galley_penalty_set_single:n #1
  { \tl_gset:Nx \g__galley_interpar_penalty_user_tl { \int_eval:n {#1} } }
\cs_new_protected:Npn \galley_vspace_set_single:n #1
  { \tl_gset:Nx \g__galley_interpar_vspace_user_tl { \skip_eval:n {#1} } }
```

(*End definition for* `\galley_penalty_set_single:n` *and* `\galley_vspace_set_single:n`*. These functions are documented on page* 4.)

`\parskip`

For the package, the `\parskip` primitive is moved out of the way as the code above is handling things.

```
⟨*package⟩
\dim_set:Nn \parskip \c_zero_dim
\cs_undefine:N \parskip
\skip_new:N \parskip
⟨/package⟩
```

(*End definition for* `\parskip`*. This function is documented on page* **??**.)

7.14 Formatting inside the paragraph

Justification is more complex than is necessarily desirable as the various TEX parameters here interact in ways which mean that clear separation between different areas is not so easy.

`\l_galley_line_left_skip` `\l_galley_line_right_skip` `\l_galley_par_begin_skip` `\l_galley_par_end_skip` `\l_galley_par_indent_dim`

The variables for setting paragraph shape: essentially, these are the TEX set.

```
\cs_new_eq:NN \l_galley_line_left_skip  \tex_leftskip:D
\cs_new_eq:NN \l_galley_line_right_skip \tex_rightskip:D
\dim_new:N \l_galley_par_begin_skip
\cs_new_eq:NN \l_galley_par_end_skip   \tex_parfillskip:D
\cs_new_eq:NN \l_galley_par_indent_dim \tex_parindent:D
```

(*End definition for* `\l_galley_line_left_skip` *and others. These variables are documented on page* 5.)

`\l_galley_last_line_fit_int`

One from ε-TEX.

```
\cs_new_eq:NN \l_galley_last_line_fit_int \etex_lastlinefit:D
```

(*End definition for* `\l_galley_last_line_fit_int`*. This variable is documented on page* 5.)

7.15 Inter-word spacing

Setting the spacing between words and between sentences is important for achieving the correct output from ragged and centered output. At the same time, as far as possible the aim is to retain the spacing specified by the font designer and not to use arbitrary values (*cf.* the approach in *The TEXbook*, p. 101).

\galley_interword_spacing_set:N

The approach taken to setting a fixed space is to use the information from the current font to set the spacing. This means that only `\tex_spacefactor:D` needs to be set, while `\tex_xspacefactor:D` is left alone. However, this is only necessary for fonts which have a stretch component to the inter-word spacing in the first place, *i.e.* monospaced fonts require no changes. The code therefore checks whether there is any stretch, and if there is uses the fixed component to set `\tex_spaceskip:D`. If there is a stretch component (non-zero `\tex_fontdimen:D 3`), then the `\teX_spaceskip:D` is set to the fixed component from the font.

```
\cs_new_protected:Npn \galley_interword_spacing_set:N #1
  {
    \bool_if:NTF #1
      { % TODO Hook for font changes required!
        \dim_compare:nNnTF { \tex_fontdimen:D \c_three \tex_font:D }
          = \c_zero_dim
          { \tex_spaceskip:D \c_zero_dim }
          { \tex_spaceskip:D \tex_fontdimen:D \c_two \tex_font:D }
      }
      { \tex_spaceskip:D \c_zero_dim }
  }
```

(*End definition for* `\galley_interword_spacing_set:N`. *This function is documented on page* 5.)

7.16 Hyphenation

\l_galley_hyphen_left_int

Currently something of a hack: this links in with language and fonts, so is not so straightforward to handle.

```
\int_new:N \l_galley_hyphen_left_int
⟨*package⟩
\int_set:Nn \l_galley_hyphen_left_int { \tex_lefthyphenmin:D }
⟨/package⟩
```

(*End definition for* `\l_galley_hyphen_left_int`. *This variable is documented on page* 6.)

7.17 Line breaking

\l_galley_binop_penalty_int
\l_galley_double_hyphen_demerits_int
\l_galley_emergency_stretch_skip
\l_galley_final_hyphen_demerits_int
\l_galley_linebreak_badness_int
\l_galley_linebreak_fuzz_dim
\l_galley_linebreak_penalty_int
\l_galley_linebreak_pretolerance_int
\l_galley_linebreak_tolerance_int
\l_galley_mismatch_demerits_int
\l_galley_relation_penalty_int

All TEX primitives renamed.

```
\cs_new_eq:NN \l_galley_binop_penalty_int           \tex_binoppenalty:D
\cs_new_eq:NN \l_galley_double_hyphen_demerits_int \tex_doublehyphendemerits:D
\cs_new_eq:NN \l_galley_emergency_stretch_skip     \tex_emergencystretch:D
\cs_new_eq:NN \l_galley_final_hyphen_demerits_int  \tex_finalhyphendemerits:D
\cs_new_eq:NN \l_galley_linebreak_badness_int      \tex_hbadness:D
\cs_new_eq:NN \l_galley_linebreak_fuzz_dim         \tex_hfuzz:D
\cs_new_eq:NN \l_galley_linebreak_penalty_int      \tex_linepenalty:D
```

```
\cs_new_eq:NN \l_galley_linebreak_pretolerance_int \tex_pretolerance:D
\cs_new_eq:NN \l_galley_mismatch_demerits_int      \tex_adjdemerits:D
\cs_new_eq:NN \l_galley_relation_penalty_int       \tex_relpenalty:D
\cs_new_eq:NN \l_galley_linebreak_tolerance_int    \tex_tolerance:D
```

(*End definition for* `\l_galley_binop_penalty_int` *and others. These variables are documented on page* 7.)

`\galley_break_line:Nn`

Terminating a line early without a new paragraph requires a few steps. First, any skips are removed, then any additional space to add is places on the surrounding vertical list. Finally, the current line is ended, using a penalty to prevents an overfull line ending `\\` giving a totally-blank one in the output. The boolean argument is used to indicate that a break is allowed after the blank line.

```
\cs_new_protected:Npn \galley_break_line:Nn #1#2
  {
    \mode_if_vertical:TF
      { \__msg_kernel_error:nn { galley } { no-line-to-end } }
      {
        \tex_unskip:D
        \bool_if:NF #1
          { \tex_vadjust:D { \tex_penalty:D \c_ten_thousand } }
        \dim_compare:nNnF {#2} = \c_zero_dim
          { \tex_vadjust:D { \skip_vertical:n {#2} } }
        \tex_penalty:D \c_ten_thousand
        \tex_hfil:D
        \tex_penalty:D -\c_ten_thousand
      }
  }
```

(*End definition for* `\galley_break_line:Nn`*. This function is documented on page* 7.)

7.18 Paragraph breaking

`\l_galley_broken_penalty_int`
`\l_galley_interline_penalty_int`
`\l_galley_parbreak_badness_int`
`\l_galley_parbreak_fuzz_dim`
`\l_galley_post_display_penalty_int`
`\l_galley_pre_display_penalty_int`

TEX primitives renamed cover *some* of this.

```
\cs_new_eq:NN \l_galley_broken_penalty_int       \tex_brokenpenalty:D
\cs_new_eq:NN \l_galley_interline_penalty_int    \tex_interlinepenalty:D
\cs_new_eq:NN \l_galley_parbreak_badness_int     \tex_vbadness:D
\cs_new_eq:NN \l_galley_parbreak_fuzz_dim        \tex_vfuzz:D
\cs_new_eq:NN \l_galley_post_display_penalty_int \tex_postdisplaypenalty:D
\cs_new_eq:NN \l_galley_pre_display_penalty_int  \tex_predisplaypenalty:D
```

(*End definition for* `\l_galley_broken_penalty_int` *and others. These variables are documented on page* 7.)

`\l_galley_club_penalties_clist`
`\l_galley_line_penalties_clist`

These are used to keep a track of information which cannot be extracted out of the primitives due to the overlapping nature of the meanings.

```
\clist_new:N \l_galley_club_penalties_clist
\clist_new:N \l_galley_line_penalties_clist
```

(*End definition for* `\l_galley_club_penalties_clist` *and* `\l_galley_line_penalties_clist`*. These functions are documented on page* ??.)

\galley_display_widow_penalties_set:n
\galley_display_widow_penalties_set:V
\galley_display_widow_penalties_set:v
\galley_widow_penalties_set:n
\galley_widow_penalties_set:V
\galley_widow_penalties_set:v
__galley_set_aux:n

By far the easiest penalties to deal with are those for widows. These work exactly as the names imply, with the display version only used immediately before display math, and the standard penalty used at the end of a paragraph. Thus there is only the need to convert the argument into the correct form, and add a 0 penalty at the end to nullify the effect of repeating the last value.

```
\cs_new_protected:Npn \galley_display_widow_penalties_set:n #1
  {
    \etex_displaywidowpenalties:D
      \__int_eval:w \clist_count:n {#1} + \c_one \__int_eval_end:
      \clist_map_function:nN {#1} \__galley_set_aux:n
      \c_zero
  }
\cs_generate_variant:Nn \galley_display_widow_penalties_set:n { V , v }
\cs_new_protected:Npn \galley_widow_penalties_set:n #1
  {
    \etex_widowpenalties:D
      \__int_eval:w \clist_count:n {#1} + \c_one \__int_eval_end:
      \clist_map_function:nN {#1} \__galley_set_aux:n
      \c_zero
  }
\cs_generate_variant:Nn \galley_widow_penalties_set:n { V , v }
\cs_new:Npn \__galley_set_aux:n #1 { \int_eval:n {#1} ~ }
```

(*End definition for* `\galley_display_widow_penalties_set:n` *and others. These functions are documented on page* **??**.)

\galley_club_penalties_set:n
\galley_club_penalties_set:V
\galley_club_penalties_set:v
\galley_interline_penalties_set:n
\galley_interline_penalties_set:V
\galley_interline_penalties_set:v

Setting club or special line penalties is easy, as these are handled mainly by the interline set up function. The two concepts are essentially the same, but having two takes makes some special effects easier to carry out.

```
\cs_new_protected:Npn \galley_club_penalties_set:n #1
  {
    \clist_set:Nn \l_galley_club_penalties_clist {#1}
    \__galley_calc_interline_penalties:
  }
\cs_generate_variant:Nn \galley_club_penalties_set:n { V , v }
\cs_new_protected:Npn \galley_interline_penalties_set:n #1
  {
    \clist_set:Nn \l_galley_line_penalties_clist {#1}
    \__galley_calc_interline_penalties:
  }
\cs_generate_variant:Nn \galley_interline_penalties_set:n { V , v }
```

(*End definition for* `\galley_club_penalties_set:n` *and others. These functions are documented on page* **??**.)

\galley_display_club_penalties_set:n
\galley_display_club_penalties_set:V
\galley_display_club_penalties_set:v

Setting the display club penalties means first setting the primitive, then recalculating the interline array to allow for these new values.

```
\cs_new_protected:Npn \galley_display_club_penalties_set:n #1
  {
    \etex_clubpenalties:D
      \__int_eval:w \clist_count:n {#1} + \c_one \__int_eval_end:
```

```
      \clist_map_function:nN {#1} \__galley_set_aux:n
      \c_zero
    \__galley_calc_interline_penalties:
  }
\cs_generate_variant:Nn \galley_display_club_penalties_set:n { V , v }
```

(*End definition for* `\galley_display_club_penalties_set:n`*,* `\galley_display_club_penalties_set:V`*, and* `\galley_display_club_penalties_set:v`*. These functions are documented on page* **??**.)

`\galley_interline_penalty_set:n`
`\__galley_set_interline_penalty:nn`
`\__galley_set_interline_penalty_auxi:n`
`\__galley_set_interline_penalty_auxii:n`

Dealing with the general interline penalty is handled in one shot. The idea is that for lines with no special penalty, the old general penalty is removed and the new one is added. If there is currently no shape set, then after adding the general interline value the generic build system is invoked (in case the `\etex_interlinepenalties:D` has accidentally been cleared).

```
\cs_new_protected:Npn \galley_interline_penalty_set:n #1
  {
    \int_compare:nNnTF { \etex_interlinepenalties:D \c_zero } = \c_zero
      {
        \etex_interlinepenalties:D \c_one \__int_eval:w #1 \__int_eval_end:
        \__galley_calc_interline_penalties:
      }
      {
        \cs_set:Npn \__galley_set_interline_penalty_auxii:n ##1
          {
            \__int_eval:w
              \etex_interlinepenalties:D ##1
               - \etex_interlinepenalties:D \etex_interlinepenalties:D \c_zero
               + #1
            \__int_eval_end:
          }
        \exp_args:Nf \__galley_set_interline_penalty:nn
          { \clist_count:N \l_galley_line_penalties_clist } {#1}
      }
  }
\cs_new_protected:Npn \__galley_set_interline_penalty:nn #1#2
  {
    \etex_interlinepenalties:D
      \etex_interlinepenalties:D \c_zero
      \int_step_function:nnnN \c_one \c_one {#1}
        \__galley_set_interline_penalty_auxi:n
      \int_step_function:nnnN { #1 + \c_one } \c_one
        { \etex_interlinepenalties:D \c_zero - \c_one }
        \__galley_set_interline_penalty_auxii:n
      \__int_eval:w #2 \__int_eval_end:
  }
\cs_new:Npn \__galley_set_interline_penalty_auxi:n #1
  { \etex_interlinepenalties:D \__int_eval:w #1 \__int_eval_end: }
\cs_new:Npn \__galley_set_interline_penalty_auxii:n #1 { }
```

(*End definition for* `\galley_interline_penalty_set:n`*. This function is documented on page* 8.)

__galley_calc_interline_penalties:
__galley_calc_interline_penalties:nn
__galley_calc_interline_penalties_auxi:n
__galley_calc_interline_penalties_auxii:n

The underlying interline penalty array has to deal with club penalties, display club penalties and any special line penalties, and include the general interline penalty. These requirements lead to a rather complex requirement on how many lines to deal with. This is needed twice, so an `f`-type expansion is used to make life a little less complex.

```
\cs_new_protected_nopar:Npn \__galley_calc_interline_penalties:
  {
    \exp_args:Nff \__galley_calc_interline_penalties:nn
      {
        \int_max:nn
          {
              \clist_count:N \l_galley_club_penalties_clist
            + \c_one
          }
          {
            \int_max:nn
              {
                  \clist_count:N \l_galley_line_penalties_clist
                + \c_one
              }
              { \etex_clubpenalties:D \c_zero }
          }
      }
      { \clist_count:N \l_galley_line_penalties_clist }
  }
```

The idea is now to calculate the correct penalties. Two auxiliary functions are used: one for any "special penalty" lines and a second for normal lines. At the end of the process, the standard interline penalty is always included.

```
\cs_new_protected:Npn \__galley_calc_interline_penalties:nn #1#2
  {
    \etex_interlinepenalties:D #1 ~
      \int_step_function:nnnN \c_one \c_one {#2}
        \__galley_calc_interline_penalties_auxi:n
      \int_step_function:nnnN { #2 + \c_one } \c_one { #1 - \c_one }
        \__galley_calc_interline_penalties_auxii:n
      \etex_interlinepenalties:D \etex_interlinepenalties:D \c_zero
  }
\cs_new:Npn \__galley_calc_interline_penalties_auxi:n #1
  {
    \__int_eval:w
        \clist_item:Nn \l_galley_line_penalties_clist {#1}
      + 0 \clist_item:Nn \l_galley_club_penalties_clist {#1}
      - \etex_clubpenalties:D #1 ~
    \__int_eval_end:
  }
\cs_new:Npn \__galley_calc_interline_penalties_auxii:n #1
  {
    \__int_eval:w
        \etex_interlinepenalties:D \etex_interlinepenalties:D \c_zero
```

```
      + 0 \clist_item:Nn \l_galley_club_penalties_clist {#1}
      - \etex_clubpenalties:D #1 ~
    \__int_eval_end:
  }
```

(*End definition for* `\__galley_calc_interline_penalties:`. *This function is documented on page* **??**.)

\galley_save_club_penalties:N
\galley_save_interline_penalties:N
\galley_save_display_club_penalties:N
\galley_save_display_widow_penalties:N
\galley_save_widow_penalties:N
__galley_save_display_club_penalties:n
__galley_save_display_widow_penalties:n
__galley_save_widow_penalties:n
\galley_interline_penalty:

Saving the array penalties varies in complexity depending on how they are stored internally. The first two are easy: these are simply copies.

```
\cs_new_protected:Npn \galley_save_club_penalties:N #1
  { \clist_set_eq:NN #1 \l_galley_club_penalties_clist }
\cs_new_protected:Npn \galley_save_interline_penalties:N #1
  { \clist_set_eq:NN #1 \l_galley_line_penalties_clist }
```

These all require appropriate mappings, using the fact that `\clist_set:Nx` will tidy up the excess comma.

```
\cs_new_protected:Npn \galley_save_display_club_penalties:N #1
  {
    \clist_set:Nx #1
      {
        \int_step_function:nnnN \c_one \c_one
          { \etex_clubpenalties:D \c_zero - \c_one }
          \__galley_save_display_club_penalties:n
      }
  }
\cs_new:Npn \__galley_save_display_club_penalties:n #1
  { \int_use:N \etex_clubpenalties:D \__int_eval:w #1 \__int_eval_end: , }
\cs_new_protected:Npn \galley_save_display_widow_penalties:N #1
  {
    \clist_set:Nx #1
      {
        \int_step_function:nnnN \c_one \c_one
          { \etex_displaywidowpenalties:D \c_zero - \c_one }
          \__galley_save_display_widow_penalties:n
      }
  }
\cs_new:Npn \__galley_save_display_widow_penalties:n #1
  { \int_use:N \etex_displaywidowpenalties:D \__int_eval:w #1 \__int_eval_end: , }
\cs_new_protected:Npn \galley_save_widow_penalties:N #1
  {
    \clist_set:Nx #1
      {
        \int_step_function:nnnN \c_one \c_one
          { \etex_widowpenalties:D \c_zero - \c_one }
          \__galley_save_widow_penalties:n
      }
  }
\cs_new:Npn \__galley_save_widow_penalties:n #1
  { \int_use:N \etex_widowpenalties:D \__int_eval:w #1 \__int_eval_end: , }
```

This one is not an array, but is stored in a primitive, so there is a simple conversion. The general interline penalty is always the last value in the primitive array.

```
\cs_new_protected_nopar:Npn \galley_interline_penalty:
  { \int_use:N \etex_interlinepenalties:D \etex_interlinepenalties:D \c_zero }
```

(*End definition for* `\galley_save_club_penalties:N` *and others. These functions are documented on page 8.*)

7.19 Messages

```
\__msg_kernel_new:nnn { galley } { no-line-to-end }
  { There's~no~line~here~to~end. }
```

7.20 LaTeX 2_ε functions

```
⟨*package⟩
```

`\clearpage` The `\clearpage` macro needs to place material into the correct structures rather than directly onto the main vertical list. Other than that it is the same as the LaTeX 2_ε version.

```
\RenewDocumentCommand \clearpage { }
  {
    \mode_if_vertical:T
      {
        \int_compare:nNnT \@dbltopnum = \c_minus_one
          {
            \dim_compare:nNnT \tex_pagetotal:D < \topskip
              { \tex_hbox:D { } }
          }
      }
    \newpage
    \tl_gput_right:Nn \g_galley_whatsit_next_tl
      { \iow_shipout:Nx \c_minus_one { } }
    \tex_vbox:D { }
    \galley_penalty_set_single:n { -\@Mi }
  }
```

(*End definition for* `\clearpage`*. This function is documented on page* **??**.)

`\nobreak` `\noindent` `\vspace` In package mode, some of LaTeX 2_ε's functions are re-implemented using the galley system. Not all of the optional arguments currently work!

```
\RenewDocumentCommand \nobreak { }
  { \bool_gset_true:N \g_galley_no_break_next_bool }
```

The `\noindent` primitive will causes problems, as it is used by LaTeX 2_ε documents to implicitly leave vertical mode as well as to prevent indentation. Rather than patch *every* place where we need leave vertical mode, at the moment we stick with the primitive as well as setting the galley flag.

```
\RenewDocumentCommand \noindent { }
  {
    \tex_noindent:D
    \bool_gset_false:N \g_galley_omit_next_indent_bool
  }
```

```
\RenewDocumentCommand \vspace { s m }
  {
    \IfBooleanTF #1
      { \galley_vspace_set_single:n {#2} }
      { \galley_vspace_set_single:n {#2} }
  }
```

(*End definition for* `\nobreak`. *This function is documented on page* **??**.)

`\\` `\newline` These functions pass their arguments straight through to the internal implementation (which is currently just the LaTeX 2_ε one recoded).

```
\RenewDocumentCommand \\ { s O { 0 pt } }
  { \galley_break_line:Nn #1 {#2} }
\RenewDocumentCommand \newline { }
  { \galley_break_line:Nn \c_true_bool { 0 pt } }
```

(*End definition for* `\\`. *This function is documented on page* **??**.)

7.21 LaTeX 2_ε fixes

Purely for testing, some internal LaTeX 2_ε functions are altered to work with the mechanism here. This material is not comprehensive: additions are made as-needed for test purposes.

`\__galleypar` The primitive is moved as otherwise the clever skipping code will fail.

```
\cs_set_eq:cN { @ @ par } \galley_par:
```

(*End definition for* `\__galleypar`. *This function is documented on page* **??**.)

`\@afterheading` Set some flags and hope for the best!

```
\cs_set_protected_nopar:Npn \@afterheading
  {
    \bool_gset_true:N \g_galley_no_break_next_bool
    \if@afterindent
    \else
      \bool_gset_true:N \g_galley_omit_next_indent_bool
    \fi
  }
```

(*End definition for* `\@afterheading`. *This function is documented on page* **??**.)

`\@hangfrom` The `\tex_hangindent:D` primitive is no longer used, so the paragraph shape is set in a different way. As a result, the label is part of the same paragraph as the main body, hence the need to leave vertical mode.

```
\cs_set_protected:Npn \@hangfrom #1
  {
    \bool_gset_true:N \g_galley_omit_next_indent_bool
    \leavevmode
    \setbox \@tempboxa = \hbox { {#1} }
    \galley_parshape_set_single:nnnN
      \c_one
      { \box_wd:N \@tempboxa }
```

```
      \c_zero_dim
      \c_false_bool
    \bool_gset_true:N \g_galley_no_break_next_bool
    \bool_gset_true:N \g_galley_omit_next_indent_bool
    \box \@tempboxa
  }
```

(*End definition for* `\@hangfrom`. *This function is documented on page* ??.)

`\@normalcr` This is needed as `\@parboxrestore` sets `\\` equal to `\@normalcr`, and the new definition must be used

```
\cs_set_eq:Nc \@normalcr { \token_to_str:N \\ }
```

(*End definition for* `\@normalcr`. *This function is documented on page* ??.)

```
⟨/package⟩
⟨/initex | package⟩
```

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.

A

B

C

H

I

L

M

N

P

R

S

T

V