The **xgalley** package
Galley*

The LaTeX3 Project†

Released 2012/04/23

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 v3570, last revised 2012/04/23.

†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`").

3 Templates

3.1 Layer two: internal dimensions

3.2 The object type 'measure'

Arg:

Semantics:

Sets the width available to typeset material within the galley. The ⟨*left margin*⟩ and ⟨*right margin*⟩ values are used in the adjustment to over-ride any given in the template. Depending upon the template in use, the margins may be absolute (relative only to the edges of the galley) or relative (taking account of `measure` adjustments already made). The template applies to the galley from the point of us forward, unless over-ridden by another use of the `measure` object type.

3.3 The template 'absolute' (object type measure)

Attributes:

left-margin (length) The distance from the left edge of the galley to the left edge of the area for typeset material. A negative value will cause the typeset material to extend beyond the edge of the galley. Default: 0 pt

right-margin (length) The distance from the right edge of the galley to the right edge of the area for typeset material. A negative value will cause the typeset material to extend beyond the edge of the galley.   Default: 0 pt

Semantics & Comments:

This template sets up the typesetting area such that typeset material runs from `left-margin` away from the left edge of the galley to `right-margin` away from the right edge of the galley. Both of these distances are absolute, *i.e.* no account is taken of previous `measure` settings. Either on or both values may be negative, in which case the typeset material will protrude outside of the edges of the galley.

3.4   The template 'relative' (object type measure)

Attributes:

left-margin (length) The distance from the previous left margin of the typeset material within the galley to the new position of the left margin. A negative value will cause the new margin to be "outside" of the previous one, and *may* cause the typeset material to protrude outside of the edge of the galley.   Default: 0 pt

right-margin (length) The distance from the previous right margin of the typeset material within the galley to the new position of the right margin. A negative value will cause the new margin to be "outside" of the previous one, and *may* cause the typeset material to protrude outside of the edge of the galley.   Default: 0 pt

Semantics & Comments:

This template sets up the typesetting area such that it has margins `left-margin` and `right-margin` within those previously set. For a galley within no previous margins, this will result in margins relative to the edges of the galley. Within a galley in which the `measure` has already been set, using the `relative` template will indent the typeset material relative to the existing margins. Either on or both values may be negative, in which case the typeset material may protrude outside of the edges of the galley.

3.5   Layer three: paragraph shape

3.6   The object type 'parshape'

Arg:

Semantics:

Template of this type define any shaping of the paragraph within the current measure of the galley. Thus they are used to generate "special" paragraph shapes, for example placing a cutout in one side of the paragraph. Typically, `parshape` templates will apply in a limited sense (to a single paragraph or a defined number of lines). However, `parshape` templates may also apply in an "ongoing" manner.

Note that `parshape` templates do not alter any first-line indent for paragraphs (or any other "in paragraph" setting). Instead, they define a shape inside which the paragraph material will be placed.

3.7   The template 'hang' (object type parshape)

Attributes:

indent (length) The hanging indent from either the left- or right-hand margin (as determined by `on-left-side`).   Default: 0 pt

on-left-side (boolean) If `true`, causes the hanging indent to be on the left-hand side of the paragraph.   Default: true

lines (integer) The number of lines of full width before hanging begins.   Default: 1

Semantics & Comments:

Sets the paragraph shape such that the after a number of full-width lines, specified by `lines`, the paragraph is indented by the `indent` from a margin. If `on-left-side` is `true` this indent will be from the left-hand margin, otherwise it will be from the right. In either case, the indent is relative to the edge of the current `measure` and may be negative (in which case an outdent will result). This template type applies only to a single paragraph.

3.8   The template 'initial' (object type parshape)

Attributes:

indent (length) The indent for the initial lines from either the left- or right-hand margin (as determined by `on-left-side`).   Default: 0 pt

on-left-side (boolean) If `true`, causes the indent to be on the left-hand side of the paragraph.   Default: true

lines (integer) The number of lines of indented lines before full-width line begins.
Default: 2

Semantics & Comments:

Sets the paragraph shape such that the first `lines` lines are indented by the `indent` given, before lines of full width begin. If `on-left-side` is `true` this indent will be from the left-hand margin, otherwise it will be from the right. In either case, the indent is relative to the edge of the current `measure` and may be negative (in which case an outdent will result). This template type applies only to a single paragraph.

3.9 The template 'std' (object type parshape)

Attributes:

()

Semantics & Comments:

Sets a rectangular paragraph shape which occupies the full width specified by the `measure`. It is therefore intended as a "do nothing" template for use where a paragraph shape is required but where no special formatting is needed. This template type applies only to a single paragraph.

3.10 Layer four: formatting inside the paragraph

3.11 The object type 'hyphenation'

Arg:

Semantics:

Controls whether hyphenation is attempted within the current galley. This object type may also alter the degree to which hyphenation is encouraged by manipulating the underlying TeX parameters. This object type applies to the galley from the point of use forward.

3.12 The template 'std' (object type hyphenation)

Attributes:

enable (boolean) Switches all hyphenation on or off. Default: true

enable-upper-case (boolean) Switches hyphenation on or off for words beginning with upper case letters. Default: true

penalty (choice) Sets the degree to which TeX is discouraged from undertaking hyphenation, from the choices `low`, `medium` and `high`. Default: low

Semantics & Comments:

Determines both whether hyphenation is allowed at all, and if so to what degree it is discouraged. Setting `penalty` to `high` does not prevent hyphenation: this is only done if `enable` is set `false`.

3.13 The object type 'justification'

Arg:

Semantics:

Controls the nature of justification undertaken within the galley. The template applies from the point of use forward.

3.14 The template 'std' (object type justification)

Attributes:

end-skip (skip) The skip inserted to fill the last line of a paragraph.

> Default: 0 pt plus 1 fil

fixed-word-spacing (boolean) Determines whether inter-word spacing has a stretch component (for non-monospaced fonts. Default: false

indent-width (length) The length of the indent inserted at the start of the first line of a new paragraph.

left-skip (skip) The skip between the left margin of the galley and the left edge of a paragraph. Default: 0 pt

right-skip (skip) The skip between the right margin of the galley and the right edge of a paragraph. Default: 0 pt

start-skip (skip) The skip inserted in addition to `indent-width` at the start of a paragraph. Default: 0 pt

Semantics & Comments:

The `std` template for justification provides rubber lengths at the start and end of the paragraph and at each side of the paragraph. It also allows for both flexible and fixed inter-word spacing. The interaction between the settings is demonstrated in the selection of standard instances provided.

3.14.1 The instance 'justified' (template justification/std)

Attribute values:

> **indent-width**   15 pt

Layout description & Comments:

Sets paragraphs fully-justified with the first line indented by `15 pt`.

3.14.2 The instance 'noindent' (template justification/std)

Attribute values:

> **end-skip**   15 pt plus 1 fil
> **indent-width**   0 pt

Layout description & Comments:

Sets paragraphs fully-justified with no indent for the first line. To ensure that paragraphs have some visual distinction, the `end-skip` is set to insert some space in all cases.

3.15 The template 'single' (object type justification)

Attributes:

end-skip (skip) The skip inserted to fill the last line of a paragraph.
> Default: 0 pt plus 1 fil

fixed-word-spacing (boolean) Determines whether inter-word spacing has a stretch component (for non-monospaced fonts. Default: false

indent-width (length) The length of the indent inserted at the start of the first line of a new paragraph.

left-skip (skip) The skip between the left margin of the galley and the left edge of a paragraph. Default: 0 pt

right-skip (skip) The skip between the right margin of the galley and the right edge of a paragraph. Default: 0 pt

start-skip (skip) The skip inserted in addition to `indent-width` at the start of a paragraph. Default: 0 pt

stretch-last-line (boolean) Determines whether inter-word spacing in the last line is stretched. If `true`, the spacing in the last line is stretched in the by the same factor as that in the penultimate line. Default: false

Semantics & Comments:

The `single` template for justification provides rubber lengths at the start and end of the paragraph and at each side of the paragraph. It also allows for both flexible and fixed inter-word spacing. The interaction between the settings is demonstrated in the selection of standard instances provided. The template applies only to a single paragraph.

3.15.1 The instance 'centered' (template justification/std)

Attribute values:

end-skip	0 pt
fixed-word-spacing	true
indent-width	0 pt
left-skip	0 pt plus 1 em
right-skip	0 pt plus 1 em

Layout description & Comments:

Centres typeset material such that hyphenation will still occur and such that very short lines are discouraged. This is similar to the LATEX 2$_\varepsilon$ `ragged2e` `Centering` environment.

3.15.2 The instance 'ragged-left' (template justification/std)

Attribute values:

end-skip	0 pt
fixed-word-spacing	true
indent-width	0 pt
left-skip	0 pt plus 2 em
right-skip	0 pt

Layout description & Comments:

Typesets material with a ragged left margin such that hyphenation will still occur and such that very short lines are discouraged. This is similar to the LATEX 2$_\varepsilon$ `ragged2e` `RaggedLeft` environment.

3.15.3 The instance 'ragged-right' (template justification/std)

Attribute values:

end-skip	0 pt
fixed-word-spacing	true
indent-width	0 pt
left-skip	0 pt
right-skip	0 pt plus 2 em

Layout description & Comments:

Typesets material with a ragged right margin such that hyphenation will still occur and such that very short lines are discouraged. This is similar to the LaTeX 2_ε `ragged2e` `RaggedLeft` environment.

3.15.4 The instance 'centering' (template justification/std)

Attribute values:

end-skip	0 pt
fixed-word-spacing	true
indent-width	0 pt
left-skip	0 pt plus 1 fil
right-skip	0 pt plus 1 fil

Layout description & Comments:

Centres typeset material such that hyphenation is strongly discouraged and short lines are allowed. This template is suited to centring arbitrary material (such as boxes) rather than centring text. In the later case, the `centered` instance should be used.

3.16 The template 'compound' (object type justification)

Attributes:

first-paragraph (instance) Justification for the first paragraph.

other-paragraphs (instance) Justification for the remaining paragraphs.

Semantics & Comments:

Here, both keys should themselves be instances of the `justification` template. The `compound` template is used to set up a single "non-standard" paragraph followed by "standard" ones. For example, it can be used to ensure that one `noindent` paragraph is then followed by `std` justification.

3.17 The object type 'line-breaking'

Arg:

Semantics:

Controls the line breaking attempted by TeX when typesetting material for the galley. This does not include whether words are hyphenated, which is handled separately.

3.18 The template 'std' (object type line-breaking)

Attributes:

badness (integer) Boundary that if exceeded will cause TeX to report an underfull line. Default: 1000

binop-penalty (integer) Penalty charged if an inline math formula is broken at a binary operator. Default: 700

double-hyphen-demerits (integer) Extra demerit charge of two (or more) lines in succession end in a hyphen. Default: 10 000

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. Default: 0 pt

final-hyphen-demerits (integer) Extra demerit charge if the second last line is hyphenated. Default: 5000

fuzz (length) Boundary below overfull lines are not reported. Default: 0.1 pt

mismatch-demerits (integer) Extra demerit charge if two visually incompatible lines follow each other. Default: 10000

line-penalty (integer) Extra penalty charged per line in the paragraph. By making this penalty higher TeX will try harder to produce compact paragraphs.Default: 10

pretolerance (integer) Maximum tolerance allowed for individual lines to break the paragraph without attempting hyphenation. Default: 100

relation-penalty (integer) Penalty charged if an inline math formula is broken at a relational symbol. Default: 500

tolerance (integer) Maximum tolerance allowed for individual lines when breaking a paragraph while attempting hyphenation (if this limit can't be met `emergency-stretch` comes into play). Default: 200

Semantics & Comments:

This is an interface to the underlying TeX system for determining line breaking.

3.19 Between paragraphs

3.20 The object type 'paragraph-breaking'

Arg:

Semantics:

This object type determines how TEX determines the behaviour when the paragraph-breaking algorithm is calculating whether to break up a paragraph. Thus for example an instance of this object type may prevent breaks within a paragraph, forbid widows or orphans, *etc.*

3.21 The template 'std' (object type paragraph-breaking)

Attributes:

badness (integer) Boundary that if exceeded will cause TEX to report an underfull vertical box.   Default: 1000

broken-penalty (integer) Penalty for page breaking after a hyphenated line.
                                                                                   Default: 100

club-penalty (integer) Penalty for generating a club line when page breaking.
                                                                                   Default: 150

display-club-penalty (integer) Penalty for breaking between to leave a club line after display math.   Default: 150

display-widow-penalty (integer) Penalty for breaking between to leave a widow line before display math.   Default: 150

fuzz (length) Boundary below which overfull vertical boxes are not reported.
                                                                                  Default: 0.1 pt

interline-penalty (integer) Penalty for breaking between lines in a paragraph.
                                                                                     Default: 0

pre-display-penalty (integer) Penalty for breaking between immediately before display math material.   Default: 10 000

post-display-penalty (integer) Penalty for breaking between immediately after display math material.   Default: 0

widow-penalty (integer) Penalty for generating a widow line when page breaking.
                                                                                   Default: 150

Semantics & Comments:

This template provides an interface to the underlying TEX mechanism for controlling page breaking. The template applies on an ongoing basis to all paragraphs after the template is used.

3.21.1   The instance 'std' (template paragraph-breaking/std)

Attribute values:

Layout description & Comments:

Sets paragraphs such that they can break with widows and orphans discouraged but not prevented. Breaks are possible after display math material but no immediately before it.

3.21.2   The instance 'nobreak' (template paragraph-breaking/std)

Attribute values:

>   **interline-penalty** 10 000
>   **post-display-penalty** 10 000

Layout description & Comments:

Sets paragraphs such that they cannot be broken at all (as far as is possible in TeX).

3.21.3   The instance 'nolone' (template paragraph-breaking/std)

Attribute values:

>   **club-penalty**   10 000
>   **display-widow-penalty** 10 000
>   **widow-penalty**   10 000

Layout description & Comments:

Sets paragraphs such that they cannot be broken to leave a club or widow line (as far as is possible in TeX).

3.22   The template 'single' (object type paragraph-breaking)

Attributes:

badness (integer) Boundary that if exceeded will cause TeX to report an underfull vertical box.   Default: ⟨*none*⟩

broken-penalty (integer) Penalty for page breaking after a hyphenated line.
                                                                  Default: ⟨*none*⟩

club-penalty (integer) Penalty for generating a club line when page breaking.
                                                                  Default: ⟨*none*⟩

display-club-penalty (integer) Penalty for breaking between to leave a club line after display math.   Default: ⟨*none*⟩

display-widow-penalty (integer) Penalty for breaking between to leave a widow line before display math. Default: ⟨*none*⟩

fuzz (length) Boundary below which overfull vertical boxes are not reported.
Default: ⟨*none*⟩

interline-penalty (integer) Penalty for breaking between lines in a paragraph.
Default: ⟨*none*⟩

pre-display-penalty (integer) Penalty for breaking between immediately before display math material. Default: ⟨*none*⟩

post-display-penalty (integer) Penalty for breaking between immediately after display math material. Default: ⟨*none*⟩

widow-penalty (integer) Penalty for generating a widow line when page breaking.
Default: ⟨*none*⟩

Semantics & Comments:

This template provides an interface to the underlying TeX mechanism for controlling page breaking. The template applies only to the next paragraph, and can thus be used to achieve effects such as non-breaking paragraphs.

3.22.1 The instance 'single-std' (template paragraph-breaking/single)

Attribute values:

Layout description & Comments:

Sets the next paragraph such that it can break with widows and orphans discouraged but not prevented. Breaks are possible after display math material but no immediately before it.

3.22.2 The instance 'single-nobreak' (template paragraph-breaking/single)

Attribute values:

> **interline-penalty** 10 000
> **post-display-penalty** 10 000

Layout description & Comments:

Sets the next paragraph such that it cannot be broken at all (as far as is possible in TeX).

3.22.3   The instance 'single-noclub' (template paragraph-breaking/single)

Attribute values:

    club-penalty   10 000
    display-club-penalty 10 000

Layout description & Comments:

Sets the next paragraph such that it cannot be broken to leave a club line (as far as is possible in T_EX).

3.22.4   The instance 'single-nolone' (template paragraph-breaking/single)

Attribute values:

    club-penalty   10 000
    display-club-penalty 10 000
    display-widow-penalty 10 000
    widow-penalty 10 000

Layout description & Comments:

Sets the next paragraph such that it cannot be broken to leave a club or widow line (as far as is possible in T_EX).

3.22.5   The instance 'single-nowidow' (template paragraph-breaking/single)

Attribute values:

    display-widow-penalty 10 000
    widow-penalty 10 000

Layout description & Comments:

Sets the next paragraph such that it cannot be broken to leave a widow line (as far as is possible in T_EX).

4   **xgalley** Implementation

This module provided a template-level interface for the LaTeX3 galley. As such, the code here is intended for design-level changes which apply to large blocks. The variables provided are therefore used only for supporting the templates, while any documented interfaces are in l3galley.

```
1 ⟨*package⟩

2 \ProvidesExplPackage
3   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
4 \RequirePackage{xparse,xtemplate,l3galley}
```

4.1 Variables

\l_galley_tmpa_clist
\l_galley_tmpb_clist

Scratch space.

```
5 \clist_new:N \l_galley_tmpa_clist
6 \clist_new:N \l_galley_tmpb_clist
```

(*End definition for* \l_galley_tmpa_clist *and* \l_galley_tmpb_clist*. These variables are documented on page* **??***.*)

4.2 Layer two: internal dimensions

There is a single object type for level two, the `measure` for the text in the galley. There are no arguments, as the measure is a design concept.

```
7 \DeclareObjectType { measure } { 0 }
```

There are two templates for galley measures: absolute and relative. Both use the same interface.

```
8 \DeclareTemplateInterface { measure } { absolute } { 0 }
9   {
10     left-margin  : length = 0 pt ,
11     right-margin : length = 0 pt
12   }
13 \DeclareTemplateInterface { measure } { relative } { 0 }
14   {
15     left-margin  : length = 0 pt ,
16     right-margin : length = 0 pt
17   }
```

\l_galley_left_margin_dim
\l_galley_right_margin_dim

In the `absolute` template, the two margin values are relative to the edges of the galley. This means that any existing offset or line-length adjustment are ignored.

```
18 ⟨*package⟩
19 \cs_new_eq:NN \l_galley_left_margin_dim \leftmargin
20 ⟨/package⟩
21 ⟨*package⟩
22 \cs_new_eq:NN \l_galley_right_margin_dim \rightmargin
23 ⟨/package⟩
24 \DeclareTemplateCode { measure } { absolute } { 0 }
25   {
26     left-margin  = \l_galley_left_margin_dim  ,
27     right-margin = \l_galley_right_margin_dim
28   }
29   {
30     \AssignTemplateKeys
31     \galley_margins_set_absolute:nn \l_galley_left_margin_dim
32       \l_galley_right_margin_dim
33   }
```

On the other hand, the `relative` template works relative to the current indentation at both sides.

```
34 \DeclareTemplateCode { measure } { relative } { 0 }
35   {
```

15

```
36     left-margin  = \l_galley_left_margin_dim   ,
37     right-margin = \l_galley_right_margin_dim
38   }
39   {
40     \AssignTemplateKeys
41     \galley_margins_set_relative:nn \l_galley_left_margin_dim
42       \l_galley_right_margin_dim
43   }
```

(*End definition for* \l_galley_left_margin_dim *and* \l_galley_right_margin_dim*. These variables are documented on page* **??**.)

4.3 Layer three: paragraph shape

The object type parshape is a somewhat extended interface to the TeX \tex_parshape:D primitive. As with the measure, the parshape template has no arguments as it is essentially a design-oriented concept.

```
44 \DeclareObjectType { parshape } { 0 }
```

There are two standard templates for paragraph shapes which do something, both with the same interface. The hang template provides one or more standard lines followed by a hanging paragraph, while the initial template cuts out a space at the start of the paragraph.

```
45 \DeclareTemplateInterface { parshape } { hang } { 0 }
46   {
47     indent      : length  = 0 pt ,
48     on-left-side : boolean = true ,
49     lines        : integer = 1
50   }
51 \DeclareTemplateInterface { parshape } { initial } { 0 }
52   {
53     indent      : length  = 0 pt ,
54     on-left-side : boolean = true ,
55     lines        : integer = 2
56   }
```

\l_galley_parshape_indent_dim
\l_galley_parshape_on_left_bool
\l_galley_parshape_lines_int

Both of the templates are implemented as special cases of the more general function defined earlier.

```
57 \DeclareTemplateCode { parshape } { hang } { 0 }
58   {
59     indent       = \l_galley_parshape_indent_dim    ,
60     on-left-side = \l_galley_parshape_on_left_bool ,
61     lines        = \l_galley_parshape_lines_int
62   }
63   {
64     \AssignTemplateKeys
65     \bool_if:NTF \l_galley_parshape_on_left_bool
66       {
67         \galley_parshape_single_par:nVVN
68           \l_galley_parshape_lines_int
```

16

```
69          \l_galley_parshape_indent_dim
70          \c_zero_dim
71          \c_false_bool
72        }
73        {
74          \galley_parshape_single_par:nVVN
75            \l_galley_parshape_lines_int
76            \c_zero_dim
77            \l_galley_parshape_indent_dim
78            \c_false_bool
79        }
80    }
81  \DeclareTemplateCode { parshape } { initial } { 0 }
82    {
83      indent      = \l_galley_parshape_indent_dim    ,
84      on-left-side = \l_galley_parshape_on_left_bool ,
85      lines       = \l_galley_parshape_lines_int
86    }
87    {
88      \AssignTemplateKeys
89      \clist_clear:N \l_galley_tmpa_clist
90      \clist_clear:N \l_galley_tmpb_clist
91      \prg_replicate:nn { \l_galley_parshape_lines_int }
92        {
93          \clist_put_right:Nn \l_galley_tmpa_clist
94            { \l_galley_parshape_indent_dim }
95          \clist_put_right:Nn \l_galley_tmpb_clist
96            { \c_zero_dim }
97        }
98      \bool_if:NTF \l_galley_parshape_on_left_bool
99        {
100          \galley_parshape_single_par:nVVN
101            \c_zero
102            \l_galley_tmpa_clist
103            \l_galley_tmpb_clist
104            \c_true_bool
105        }
106        {
107          \galley_parshape_single_par:nVVN
108            \c_zero
109            \l_galley_tmpb_clist
110            \l_galley_tmpa_clist
111            \c_true_bool
112        }
113    }
```
(*End definition for* \l_galley_parshape_indent_dim. *This function is documented on page* **??**.)

There is also a "do nothing" paragraph shape for cases where a template is needed but no action is desirable.

```
114  \DeclareTemplateInterface { parshape } { std } { 0 } { }
```

```
115  \DeclareTemplateCode { parshape } { std } { 0 } { } { }
```

4.4 Layer four: formatting inside the paragraph

The first type of object within a paragraph is the hyphenation. This object needs no arguments.

```
116  \DeclareObjectType { hyphenation } { 0 }
```

There is only hyphenation template as standard. This provides a semi-flexible interface to the underlying TeX methods. (The detail is therefore hidden within the implementation phase.)

```
117  \DeclareTemplateInterface { hyphenation } { std } { 0 }
118    {
119      enable            : boolean                        = true ,
120      enable-upper-case : boolean                        = true ,
121      penalty           : choice { low , medium , high } = low
122    }
```

The implementation for hyphenation mainly sets low-level values. The minimum number of characters after a hyphen is set directly, whereas the number before is not. This is so that `\tex_lefthyphenmin:D` can also be used to completely prevent hyphenation.

```
123  \DeclareTemplateCode { hyphenation } { std } { 0 }
124    {
125      enable            = \l_galley_hyphen_enable_bool    ,
126      enable-upper-case = \l_galley_hyphen_uppercase_bool ,
127      penalty           =
128        {
129          low    =
130            {
131              \int_set:Nn \tex_hyphenpenalty:D   { 51 }
132              \int_set:Nn \tex_exhyphenpenalty:D { 51 }
133            } ,
134          medium =
135            {
136              \int_set:Nn \tex_hyphenpenalty:D   { 151 }
137              \int_set:Nn \tex_exhyphenpenalty:D { 151 }
138            } ,
139          high   =
140            {
141              \int_set:Nn \tex_hyphenpenalty:D   { 301 }
142              \int_set:Nn \tex_exhyphenpenalty:D { 301 }
143            } ,
144        }
145    }
146    {
147      \AssignTemplateKeys
148      \int_set:Nn \tex_lefthyphenmin:D
149        {
150          \bool_if:NTF \l_galley_hyphen_enable_bool
151            { \l_galley_hyphen_left_int  }
```

```
152          { 63 }
153        }
154      \int_set:Nn \tex_uchyph:D
155        {
156          \bool_if:NTF \l_galley_hyphen_uppercase_bool
157            { 1 }
158            { 0 }
159        }
160    }
```

At this stage, the default hyphenation character should be set and hyphenation should be enabled.

```
161 \UseTemplate { hyphenation } { std } { }
162 \tex_defaulthyphenchar:D 45 \scan_stop:
```

\l_galley_justification_other_tl Used for the reset system for justification: using this token list means that there is no need to remove anything from \g_galley_restore_running_tl.

```
163 \tl_new:N \l_galley_justification_other_tl
```

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

The second level four object is the justification, which again takes no arguments.

```
164 \DeclareObjectType { justification } { 0 }
```

There are two templates here with the same interface: the standard one to apply from this point onward, and one which applies only to a single paragraph.

```
165 \DeclareTemplateInterface { justification } { std } { 0 }
166    {
167      end-skip          : skip    = 0 pt plus 1 fil ,
168      fixed-word-spacing : boolean = false            ,
169      indent-width      : length                      ,
170      left-skip         : skip    = 0 pt              ,
171      right-skip        : skip    = 0 pt              ,
172      start-skip        : skip    = 0 pt              ,
173      stretch-last-line : boolean = false
174    }
175 \DeclareTemplateInterface { justification } { single } { 0 }
176    {
177      end-skip          : skip    = 0 pt plus 1 fil ,
178      fixed-word-spacing : boolean = false            ,
179      indent-width      : length                      ,
180      left-skip         : skip    = 0 pt              ,
181      right-skip        : skip    = 0 pt              ,
182      start-skip        : skip    = 0 pt              ,
183      stretch-last-line : boolean = false
184    }
```

\l_galley_fixed_spacing_bool The implementation here is pretty simple as almost everything that goes on is a simple case of saving the settings, which are then applied either by TeX itself or the rest of the galley system.

```
185 \DeclareTemplateCode { justification } { std } { 0 }
```

19

```
186   {
187     end-skip          = \l_galley_par_end_skip           ,
188     fixed-word-spacing = \l_galley_fixed_spacing_bool     ,
189     indent-width      = \l_galley_par_indent_dim         ,
190     left-skip         = \l_galley_line_left_skip         ,
191     right-skip        = \l_galley_line_right_skip        ,
192     start-skip        = \l_galley_par_begin_skip         ,
193     stretch-last-line  = \l_galley_par_stretch_last_bool
194   }
195   {
196     \AssignTemplateKeys
197     \tl_clear:N \l_galley_justification_other_tl
198     \galley_set_interword_spacing:N \l_galley_fixed_spacing_bool
199     \bool_if:NTF \l_galley_par_stretch_last_bool
200       { \int_set_eq:NN \l_galley_last_line_fit_int \c_one_thousand }
201       { \int_zero:N \l_galley_last_line_fit_int }
202     \skip_set:Nn \@rightskip { \l_galley_line_right_skip }
203   }
```

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

To deal with a single paragraph, the approach used is to save the current settings to the paragraph-reset code, then to assign the template in the same way as for the std template.

```
204 \DeclareTemplateCode { justification } { single } { 0 }
205   {
206     end-skip          = \l_galley_par_end_skip           ,
207     fixed-word-spacing = \l_galley_fixed_spacing_bool     ,
208     indent-width      = \l_galley_par_indent_dim         ,
209     left-skip         = \l_galley_line_left_skip         ,
210     right-skip        = \l_galley_line_right_skip        ,
211     start-skip        = \l_galley_par_begin_skip         ,
212     stretch-last-line  = \l_galley_par_stretch_last_bool
213   }
214   {
215     \tl_put_left:Nx \l_galley_justification_other_tl
216       {
217         \skip_set:Nn \exp_not:N \l_galley_par_end_skip
218           { \skip_use:N \l_galley_par_end_skip }
219         \bool_if:NTF \l_galley_fixed_spacing_bool
220           { \bool_set_true:N  \exp_not:N \l_galley_fixed_spacing_bool }
221           { \bool_set_false:N \exp_not:N \l_galley_fixed_spacing_bool }
222         \galley_set_interword_spacing:N
223           \exp_not:N \l_galley_fixed_spacing_bool
224         \dim_set:Nn \exp_not:N \l_galley_par_indent_dim
225           { \dim_use:N \l_galley_par_indent_dim  }
226         \skip_set:Nn \l_galley_line_left_skip
227           { \skip_use:N \l_galley_line_left_skip }
228         \skip_set:Nn \exp_not:N \l_galley_line_right_skip
229           { \skip_use:N \l_galley_line_right_skip }
230         \skip_set:Nn \exp_not:N \l_galley_par_begin_skip
```

```
231            { \skip_use:N \l_galley_par_begin_skip }
232          \int_set:Nn \exp_not:N \l_galley_last_line_fit_int
233            { \int_use:N \l_galley_last_line_fit_int }
234          \skip_set:Nn \exp_not:N \@rightskip
235            { \skip_use:N \l_galley_line_right_skip }
236        }
237      \tl_gput_right:Nn \g_galley_restore_running_tl
238        { \l_galley_justification_other_tl }
239      \AssignTemplateKeys
240      \galley_set_interword_spacing:N \l_galley_fixed_spacing_bool
241      \bool_if:NTF \l_galley_par_stretch_last_bool
242        { \int_set_eq:NN \l_galley_last_line_fit_int \c_one_thousand }
243        { \int_zero:N \l_galley_last_line_fit_int }
244      \skip_set:Nn \@rightskip { \l_galley_line_right_skip }
245    }
```

The standard instance for justification is very simple to set up as the default values for
the template are set up for exactly this case. The advantage of this scheme is that at a
design level altering the indent used for justified paragraphs is very easy to do. As this
is the standard template for all LaTeX3 documents, it is applied here.

```
246 \DeclareInstance { justification } { justified } { std }
247   { indent-width = 15 pt }
248 \UseInstance { justification } { justified }
```

The instance for no indentation at all but with justified text is intended for layouts
which leave white space between paragraphs. With no indentation, some space has to be
included at the end of each paragraph. This is set up to mirror the indent that has been
removed.

```
249 \DeclareInstance { justification } { noindent } { std }
250   {
251     end-skip     = 15 pt plus 1 fil ,
252     indent-width = 0 pt
253   }
```

The other standard justification schemes are for text which is either centred or ragged.
The settings here are taken from the LaTeX 2ε ragged2e package, as they maintain a
reasonable appearance by ensuring that TeX will not be too tolerant of very short lines.
To keep the design clear here, no default values are relied on even though this would
make the instance declarations shorter.

```
254 \DeclareInstance { justification } { centered } { std }
255   {
256     end-skip          = 0 pt            ,
257     fixed-word-spacing = true           ,
258     indent-width      = 0 pt            ,
259     left-skip         = 0 pt plus 1 em ,
260     right-skip        = 0 pt plus 1 em
261   }
262 \DeclareInstance { justification } { ragged-left } { std }
263   {
264     end-skip          = 0 pt            ,
```

```
265    fixed-word-spacing = true            ,
266    indent-width       = 0 pt            ,
267    left-skip          = 0 pt plus 2 em  ,
268    right-skip         = 0 pt
269  }
270 \DeclareInstance { justification } { ragged-right } { std }
271  {
272    end-skip           = 0 pt plus 1 fil ,
273    fixed-word-spacing = true            ,
274    indent-width       = 0 pt            ,
275    left-skip          = 0 pt            ,
276    right-skip         = 0 pt plus 2 em
277  }
```

The `centering` instance is used to centre material without hyphenation: this is used for centring arbitrary material rather than text.

```
278 \DeclareInstance { justification } { centering } { std }
279  {
280    end-skip           = 0 pt            ,
281    fixed-word-spacing = true            ,
282    indent-width       = 0 pt            ,
283    left-skip          = 0 pt plus 1 fil ,
284    right-skip         = 0 pt plus 1 fil
285  }
```

`\galley_justification_first:`
`\galley_justification_other:`
A second form of justification template is the case where the first paragraph is different from all of the others. This is set up by getting the justification to reset itself after the first paragraph. The code built into the `std` version will ensure that any subsequent template use will over-ride the setting here correctly.

```
286 \DeclareTemplateInterface { justification } { compound } { 0 }
287  {
288    first-paragraph  : instance { justification } ,
289    other-paragraphs : instance { justification }
290  }
291 \DeclareTemplateCode { justification } { compound } { 0 }
292  {
293    first-paragraph  = \galley_justification_first: ,
294    other-paragraphs = \galley_justification_other:
295  }
296  {
297    \AssignTemplateKeys
298    \galley_justification_first:
299    \tl_set:Nn \l_galley_justification_other_tl
300      { \galley_justification_other: }
301    \tl_gput_right:Nn \g_galley_restore_running_tl
302      { \l_galley_justification_other_tl }
303  }
```

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

How TeX breaks text into lines is influences by a number of parameters, most of which are not actually likely to change. These work with the `hyphenation` but are independent of whether any hyphenation is actually active. The math values here could be set up as a separate template, but in practice this seems likely to be overkill.

```
304 \DeclareObjectType { line-breaking } { 0 }
```

The only template provided for line breaking is a simple interface to TeX's parameters. There is not really much that can be added to this: after all, the way that penalties work is more or less arbitrary but works well! The default values given here are intended to be sensible for a lot of cases.

```
305 \DeclareTemplateInterface { line-breaking } { std } { 0 }
306   {
307     badness                : integer = 1000   ,
308     binop-penalty          : integer = 700    ,
309     double-hyphen-demerits : integer = 10 000 ,
310     emergency-stretch      : skip    = 0 pt    ,
311     final-hyphen-demerits  : integer = 5000   ,
312     fuzz                   : length  = 0.1 pt ,
313     line-penalty           : integer = 10     ,
314     mismatch-demerits      : integer = 10 000 ,
315     pretolerance           : integer = 100    ,
316     relation-penalty       : integer = 500    ,
317     tolerance              : integer = 200
318   }
319 \DeclareTemplateCode{ line-breaking } { std } { 0 }
320   {
321     badness                = \l_galley_linebreak_badness_int      ,
322     binop-penalty          = \l_galley_binop_penalty_int          ,
323     double-hyphen-demerits = \l_galley_double_hyphen_demerits_int ,
324     emergency-stretch      = \l_galley_emergency_stretch_skip      ,
325     final-hyphen-demerits  = \l_galley_final_hyphen_demerits_int  ,
326     fuzz                   = \l_galley_linebreak_fuzz_dim          ,
327     line-penalty           = \l_galley_linebreak_penalty_int      ,
328     mismatch-demerits      = \l_galley_mismatch_demerits_int       ,
329     pretolerance           = \l_galley_linebreak_pretolerance_int ,
330     relation-penalty       = \l_galley_relation_penalty_int       ,
331     tolerance              = \l_galley_linebreak_tolerance_int
332   }
333   { \AssignTemplateKeys }
```

The default values are set such that they are suitable for good quality typesetting. So the standard template changes nothing at all from the template. This instance should also be applied now, as it will then apply to the entire document unless changed deliberately.

```
334 \DeclareInstance { line-breaking } { std } { std } { }
335 \UseInstance { line-breaking } { std }
```

4.5 Between paragraphs

`\l_galley_club_penalty_int`
`\l_galley_display_club_penalty_int`
`\l_galley_display_widow_penalty_int`
`\l_galley_interline_penalty_int`
`\l_galley_widow_penalty_int`

The second object here sets up how TeX acts to break paragraphs at page boundaries. As with the `line-breaking` object, there is not much to do except provide an interface

to the TeX internals. The **std** template does *not* make the ε-TeX array nature of various penalties available.

```
336 \DeclareObjectType { paragraph-breaking } { 0 }
337 \DeclareTemplateInterface { paragraph-breaking } { std } { 0 }
338   {
339     badness                : integer = 1000    ,
340     broken-penalty         : integer = 100     ,
341     club-penalty           : integer = 150     ,
342     display-club-penalty   : integer = 150     ,
343     display-widow-penalty  : integer = 150     ,
344     fuzz                   : length  = 0.1 pt  ,
345     interline-penalty      : integer = 0       ,
346     post-display-penalty   : integer = 0       ,
347     pre-display-penalty    : integer = 10 000  ,
348     widow-penalty          : integer = 150
349   }
350 \DeclareTemplateCode { paragraph-breaking } { std } { 0 }
351   {
352     badness               = \l_galley_parbreak_badness_int      ,
353     broken-penalty        = \l_galley_broken_penalty_int        ,
354     club-penalty          = \l_galley_club_penalty_int          ,
355     display-club-penalty  = \l_galley_display_club_penalty_int  ,
356     display-widow-penalty = \l_galley_display_widow_penalty_int ,
357     fuzz                  = \l_galley_parbreak_fuzz_dim         ,
358     interline-penalty     = \l_galley_interline_penalty_int     ,
359     post-display-penalty  = \l_galley_post_display_penalty_int  ,
360     pre-display-penalty   = \l_galley_pre_display_penalty_int   ,
361     widow-penalty         = \l_galley_widow_penalty_int
362   }
363   {
364     \AssignTemplateKeys
365     \galley_set_club_penalties:V          \l_galley_club_penalty_int
366     \galley_set_display_club_penalties:V  \l_galley_display_club_penalty_int
367     \galley_set_display_widow_penalties:V \l_galley_display_widow_penalty_int
368     \galley_set_interline_penalty:n       \l_galley_interline_penalty_int
369     \galley_set_widow_penalties:V         \l_galley_widow_penalty_int
370   }
```

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

The standard instance of the **paragraph-breaking** object simply applies the defaults: this is used.

```
371 \DeclareInstance { paragraph-breaking } { std } { std } { }
372 \UseInstance { paragraph-breaking } { std }
```

Two additional instances are provided: one to prevent any breaks at all, and a second to prevent any widow or club lines.

```
373 \DeclareInstance { paragraph-breaking } { nobreak } { std }
374   {
375     interline-penalty    = 10 000 ,
```

```
376       post-display-penalty = 10 000
377     }
378  \DeclareInstance { paragraph-breaking } { nolone } { std }
379    {
380       club-penalty          = 10 000 ,
381       display-club-penalty  = 10 000 ,
382       display-widow-penalty = 10 000 ,
383       widow-penalty         = 10 000
384    }
```

There is also a version of this code which applies only to one paragraph. This is done
by storing the input in token list variables with no default: only explicit settings will be
picked up.

```
385  \DeclareTemplateInterface { paragraph-breaking } { single } { 0 }
386    {
387       badness              : tokenlist ,
388       broken-penalty       : tokenlist ,
389       club-penalty         : tokenlist ,
390       display-club-penalty  : tokenlist ,
391       display-widow-penalty : tokenlist ,
392       fuzz                 : tokenlist ,
393       interline-penalty    : tokenlist ,
394       post-display-penalty  : tokenlist ,
395       pre-display-penalty   : tokenlist ,
396       widow-penalty        : tokenlist
397    }
398  \DeclareTemplateCode { paragraph-breaking } { single } { 0 }
399    {
400       badness               = \l_galley_parbreak_badness_tl          ,
401       broken-penalty        = \l_galley_broken_penalty_tl            ,
402       club-penalty          = \l_galley_club_penalties_tl            ,
403       display-club-penalty  = \l_galley_display_club_penalties_tl    ,
404       display-widow-penalty = \l_galley_display_widow_penalties_tl   ,
405       fuzz                  = \l_galley_parbreak_fuzz_tl             ,
406       interline-penalty     = \l_galley_interline_penalty_tl         ,
407       post-display-penalty  = \l_galley_post_display_penalty_tl      ,
408       pre-display-penalty   = \l_galley_pre_display_penalty_tl       ,
409       widow-penalty         = \l_galley_widow_penalties_tl
410    }
411    {
412       \AssignTemplateKeys
```

The fuzz and interline penalties are handled explicitly as they have particular require-
ments.

```
413       \tl_if_empty:NF \l_galley_interline_penalty_tl
414         {
415           \tl_gput_right:Nx \g_galley_par_after_hook_tl
416             {
417               \int_set:Nn \exp_not:N \l_galley_interline_penalty_int
418                 { \galley_interline_penalty: }
```

\l_galley_parbreak_badness_tl
\l_galley_broken_penalty_tl
\l_galley_club_penalties_tl
\l_galley_display_club_penalties_tl
\l_galley_display_widow_penalties_tl
\l_galley_parbreak_fuzz_tl
\l_galley_interline_penalty_tl
\l_galley_post_display_penalty_tl
\l_galley_pre_display_penalty_tl
\l_galley_widow_penalties_tl
\c_galley_parbreak_multi_seq
\c_galley_parbreak_single_seq

```
419            }
420          \int_set:Nn \l_galley_interline_penalty_int
421            { \l_galley_interline_penalty_tl }
422        }
423      \tl_if_empty:NF \l_galley_parbreak_fuzz_tl
424        {
425          \tl_gput_right:Nx \g_galley_par_after_hook_tl
426            {
427              \dim_set:Nn \exp_not:N \l_galley_parbreak_fuzz_dim
428                { \dim_use:N \l_galley_parbreak_fuzz_dim }
429            }
430          \dim_set:Nn \l_galley_parbreak_fuzz_dim { \l_galley_parbreak_fuzz_tl }
431        }
```

For the single integer penalties, a simple check is needed to save the value.

```
432      \seq_map_inline:Nn \c_galley_parbreak_single_seq
433        {
434          \tl_if_empty:cF { l_galley_ ##1 _tl }
435            {
436              \tl_gput_right:Nx \g_galley_par_after_hook_tl
437                {
438                  \int_set:Nn \exp_not:c { l_galley_ ##1 _int }
439                    { \int_use:c { l_galley_ ##1 _int } }
440                }
441              \int_set:cn { l_galley_ ##1 _int }
442                { \tl_use:c { l_galley_ ##1 _tl } }
443            }
444        }
```

A bit more complex for the array penalties. Although the interface here does not expose the arrays, it is necessary to correctly save them.

```
445      \seq_map_inline:Nn \c_galley_parbreak_multi_seq
446        {
447          \tl_if_empty:cF { l_galley_ ##1 _tl }
448            {
449              \use:c { galley_save_ ##1 :N } \l_galley_tmpa_clist
450              \tl_gput_right:Nx \g_galley_par_after_hook_tl
451                {
452                  \exp_not:c { galley_set_ ##1 :n }
453                    { \exp_not:o \l_galley_tmpa_clist }
454                }
455              \use:c { galley_set_ ##1 :v } { l_galley_ ##1 _tl }
456            }
457        }
458    }
459  \seq_new:N \c_galley_parbreak_multi_seq
460  \seq_gput_right:Nn \c_galley_parbreak_multi_seq { club_penalties }
461  \seq_gput_right:Nn \c_galley_parbreak_multi_seq { display_club_penalties }
462  \seq_gput_right:Nn \c_galley_parbreak_multi_seq { display_widow_penalties }
463  \seq_gput_right:Nn \c_galley_parbreak_multi_seq { widow_penalties }
```

```
464 \seq_new:N \c_galley_parbreak_single_seq
465 \seq_gput_right:Nn \c_galley_parbreak_single_seq { parbreak_badness }
466 \seq_gput_right:Nn \c_galley_parbreak_single_seq { broken_penalty }
467 \seq_gput_right:Nn \c_galley_parbreak_single_seq { post_display_penalty }
468 \seq_gput_right:Nn \c_galley_parbreak_single_seq { pre_display_penalty }
```
(*End definition for* \l_galley_parbreak_badness_tl *and others. These variables are documented on page* **??**.)
```
469 \DeclareInstance { paragraph-breaking } { single-std } { single } { }
470 \DeclareInstance { paragraph-breaking } { single-nobreak } { single }
471   {
472     interline-penalty    = 10 000 ,
473     post-display-penalty = 10 000
474   }
475 \DeclareInstance { paragraph-breaking } { single-noclub } { single }
476   {
477     club-penalty         = 10 000 ,
478     display-club-penalty = 10 000
479   }
480 \DeclareInstance { paragraph-breaking } { single-nolone } { single }
481   {
482     club-penalty         = 10 000 ,
483     display-club-penalty  = 10 000 ,
484     display-widow-penalty = 10 000 ,
485     widow-penalty         = 10 000
486   }
487 \DeclareInstance { paragraph-breaking } { single-nowidow } { single }
488   {
489     display-widow-penalty = 10 000 ,
490     widow-penalty         = 10 000
491   }
```

4.6   Templates for display material

To allow special handling of display-like material, templates are needed at the beginning and end of the block which set up any special space or breaks. These need to be optional, and so are stored as token lists: rather than "magic" values, empty lists indicate that standard settings are to be used. To ensure that the error checking needed takes place early, each token list is re-set with the appropriate evaluation.

```
492 \DeclareObjectType { display-begin } { 0 }
493 \DeclareObjectType { display-end }   { 0 }
494 \DeclareTemplateInterface { display-begin } { std } { 0 }
495   {
496     par-penalty : tokenlist ,
497     par-space   : tokenlist ,
498     penalty     : tokenlist ,
499     space       : tokenlist
500   }
501 \DeclareTemplateInterface { display-end } { std } { 0 }
```

27

```
502    {
503      par-penalty : tokenlist ,
504      par-space   : tokenlist ,
505      penalty     : tokenlist ,
506      space       : tokenlist
507    }
508  \DeclareTemplateCode { display-begin } { std } { 0 }
509    {
510      par-penalty = \l_galley_display_begin_par_vpenalty_tl ,
511      par-space   = \l_galley_display_begin_par_vspace_tl   ,
512      penalty     = \l_galley_display_begin_vpenalty_tl     ,
513      space       = \l_galley_display_begin_vspace_tl
514    }
515    {
516     \AssignTemplateKeys
517     \tl_if_empty:NF \l_galley_display_begin_par_vpenalty_tl
518       {
519         \tl_set:Nx \l_galley_display_begin_par_vpenalty_tl
520           { \int_eval:n { \l_galley_display_begin_par_vpenalty_tl } }
521       }
522     \tl_if_empty:NF \l_galley_display_begin_par_vspace_tl
523       {
524         \tl_set:Nx \l_galley_display_begin_par_vspace_tl
525           { \skip_eval:n { \l_galley_display_begin_par_vspace_tl } }
526       }
527     \tl_if_empty:NF \l_galley_display_begin_vpenalty_tl
528       {
529         \tl_set:Nx \l_galley_display_begin_vpenalty_tl
530           { \int_eval:n { \l_galley_display_begin_vpenalty_tl } }
531       }
532     \tl_if_empty:NF \l_galley_display_begin_vspace_tl
533       {
534         \tl_set:Nx \l_galley_display_begin_vspace_tl
535           { \skip_eval:n { \l_galley_display_begin_vspace_tl } }
536       }
537    }
538  \DeclareTemplateCode { display-end } { std } { 0 }
539    {
540      par-penalty = \l_galley_display_end_par_vpenalty_tl ,
541      par-space   = \l_galley_display_end_par_vspace_tl   ,
542      penalty     = \l_galley_display_end_vpenalty_tl     ,
543      space       = \l_galley_display_end_vspace_tl
544    }
545    {
546     \AssignTemplateKeys
547     \tl_if_empty:NF \l_galley_display_end_par_vpenalty_tl
548       {
549         \tl_set:Nx \l_galley_display_end_par_vpenalty_tl
550           { \int_eval:n { \l_galley_display_end_par_vpenalty_tl } }
551       }
```

```
552      \tl_if_empty:NF \l_galley_display_end_par_vspace_tl
553        {
554          \tl_set:Nx \l_galley_display_end_par_vspace_tl
555            { \skip_eval:n { \l_galley_display_end_par_vspace_tl } }
556        }
557      \tl_if_empty:NF \l_galley_display_end_vpenalty_tl
558        {
559          \tl_set:Nx \l_galley_display_end_vpenalty_tl
560            { \int_eval:n { \l_galley_display_end_vpenalty_tl } }
561        }
562      \tl_if_empty:NF \l_galley_display_end_vspace_tl
563        {
564          \tl_set:Nx \l_galley_display_end_vspace_tl
565            { \skip_eval:n { \l_galley_display_end_vspace_tl } }
566        }
567    }
568  ⟨/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.

29