The xgalley package
Galley*

The LaTeX3 Project†

Released 2012/02/07

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 v3330, last revised 2012/02/07.

†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_ε `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_ε `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 TEX).

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 TEX).

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 TEX).

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.

```
\clist_new:N \l_galley_tmpa_clist
\clist_new:N \l_galley_tmpb_clist
```

(*End definition for* `\l_galley_tmpa_clist` *and* `\l_galley_tmpb_clist`*. These functions 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.

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

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

```
\DeclareTemplateInterface { measure } { absolute } { 0 }
  {
    left-margin  : length = 0 pt ,
    right-margin : length = 0 pt
  }
\DeclareTemplateInterface { measure } { relative } { 0 }
  {
    left-margin  : length = 0 pt ,
    right-margin : length = 0 pt
  }
```

\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.

```
⟨*package⟩
\cs_new_eq:NN \l_galley_left_margin_dim \leftmargin
⟨/package⟩
⟨*package⟩
\cs_new_eq:NN \l_galley_right_margin_dim \rightmargin
⟨/package⟩
\DeclareTemplateCode { measure } { absolute } { 0 }
  {
    left-margin  = \l_galley_left_margin_dim  ,
    right-margin = \l_galley_right_margin_dim
  }
  {
    \AssignTemplateKeys
    \galley_margins_set_absolute:nn \l_galley_left_margin_dim
      \l_galley_right_margin_dim
  }
```

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

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

```
    left-margin  = \l_galley_left_margin_dim  ,
    right-margin = \l_galley_right_margin_dim
  }
  {
    \AssignTemplateKeys
    \galley_margins_set_relative:nn \l_galley_left_margin_dim
      \l_galley_right_margin_dim
  }
```

(*End definition for* `\l_galley_left_margin_dim` *and* `\l_galley_right_margin_dim`*. These functions 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.

```
\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.

```
\DeclareTemplateInterface { parshape } { hang } { 0 }
  {
    indent       : length  = 0 pt ,
    on-left-side : boolean = true ,
    lines        : integer = 1
  }
\DeclareTemplateInterface { parshape } { initial } { 0 }
 {
    indent       : length  = 0 pt ,
    on-left-side : boolean = true ,
    lines        : integer = 2
  }
```

\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.

```
\DeclareTemplateCode { parshape } { hang } { 0 }
  {
    indent       = \l_galley_parshape_indent_dim   ,
    on-left-side = \l_galley_parshape_on_left_bool ,
    lines        = \l_galley_parshape_lines_int
  }
  {
    \AssignTemplateKeys
    \bool_if:NTF \l_galley_parshape_on_left_bool
      {
        \galley_parshape_single_par:nVVN
          \l_galley_parshape_lines_int
```

```
          \l_galley_parshape_indent_dim
          \c_zero_dim
          \c_false_bool
      }
      {
        \galley_parshape_single_par:nVVN
          \l_galley_parshape_lines_int
          \c_zero_dim
          \l_galley_parshape_indent_dim
          \c_false_bool
      }
  }
\DeclareTemplateCode { parshape } { initial } { 0 }
  {
    indent       = \l_galley_parshape_indent_dim   ,
    on-left-side = \l_galley_parshape_on_left_bool ,
    lines        = \l_galley_parshape_lines_int
  }
  {
    \AssignTemplateKeys
    \clist_clear:N \l_galley_tmpa_clist
    \clist_clear:N \l_galley_tmpb_clist
    \prg_replicate:nn { \l_galley_parshape_lines_int }
      {
        \clist_put_right:Nn \l_galley_tmpa_clist
          { \l_galley_parshape_indent_dim }
        \clist_put_right:Nn \l_galley_tmpb_clist
          { \c_zero_dim }
      }
    \bool_if:NTF \l_galley_parshape_on_left_bool
      {
        \galley_parshape_single_par:nVVN
          \c_zero
          \l_galley_tmpa_clist
          \l_galley_tmpb_clist
          \c_true_bool
      }
      {
        \galley_parshape_single_par:nVVN
          \c_zero
          \l_galley_tmpb_clist
          \l_galley_tmpa_clist
          \c_true_bool
      }
  }
```

(*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.

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

```
\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.

```
\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.)

```
\DeclareTemplateInterface { hyphenation } { std } { 0 }
  {
    enable            : boolean                        = true ,
    enable-upper-case : boolean                        = true ,
    penalty           : choice { low , medium , high } = low
  }
```

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.

```
\DeclareTemplateCode { hyphenation } { std } { 0 }
  {
    enable            = \l_galley_hyphen_enable_bool    ,
    enable-upper-case = \l_galley_hyphen_uppercase_bool ,
    penalty           =
      {
        low    =
          {
            \int_set:Nn \tex_hyphenpenalty:D   { 51 }
            \int_set:Nn \tex_exhyphenpenalty:D { 51 }
          } ,
        medium =
          {
            \int_set:Nn \tex_hyphenpenalty:D   { 151 }
            \int_set:Nn \tex_exhyphenpenalty:D { 151 }
          } ,
        high   =
          {
            \int_set:Nn \tex_hyphenpenalty:D   { 301 }
            \int_set:Nn \tex_exhyphenpenalty:D { 301 }
          } ,
      }
  }
  {
    \AssignTemplateKeys
    \int_set:Nn \tex_lefthyphenmin:D
      {
        \bool_if:NTF \l_galley_hyphen_enable_bool
          { \l_galley_hyphen_left_int  }
```

```
          { 63 }
      }
    \int_set:Nn \tex_uchyph:D
      {
        \bool_if:NTF \l_galley_hyphen_uppercase_bool
          { 1 }
          { 0 }
      }
  }
```

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

```
\UseTemplate { hyphenation } { std } { }
\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`.

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

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

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

```
\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.

```
\DeclareTemplateInterface { justification } { std } { 0 }
  {
    end-skip           : skip    = 0 pt plus 1 fil ,
    fixed-word-spacing : boolean = false           ,
    indent-width       : length                    ,
    left-skip          : skip    = 0 pt           ,
    right-skip         : skip    = 0 pt           ,
    start-skip         : skip    = 0 pt           ,
    stretch-last-line  : boolean = false
  }
\DeclareTemplateInterface { justification } { single } { 0 }
  {
    end-skip           : skip    = 0 pt plus 1 fil ,
    fixed-word-spacing : boolean = false           ,
    indent-width       : length                    ,
    left-skip          : skip    = 0 pt           ,
    right-skip         : skip    = 0 pt           ,
    start-skip         : skip    = 0 pt           ,
    stretch-last-line  : boolean = false
  }
```

\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.

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

```
  {
    end-skip           = \l_galley_par_end_skip          ,
    fixed-word-spacing = \l_galley_fixed_spacing_bool    ,
    indent-width       = \l_galley_par_indent_dim        ,
    left-skip          = \l_galley_line_left_skip        ,
    right-skip         = \l_galley_line_right_skip       ,
    start-skip         = \l_galley_par_begin_skip        ,
    stretch-last-line  = \l_galley_par_stretch_last_bool
  }
  {
    \AssignTemplateKeys
    \tl_clear:N \l_galley_justification_other_tl
    \galley_set_interword_spacing:N \l_galley_fixed_spacing_bool
    \bool_if:NTF \l_galley_par_stretch_last_bool
      { \int_set_eq:NN \l_galley_last_line_fit_int \c_one_thousand }
      { \int_zero:N \l_galley_last_line_fit_int }
    \skip_set:Nn \@rightskip { \l_galley_line_right_skip }
  }
```

(*End definition for* `\l_galley_fixed_spacing_bool`. *This function 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.

```
\DeclareTemplateCode { justification } { single } { 0 }
  {
    end-skip           = \l_galley_par_end_skip         ,
    fixed-word-spacing = \l_galley_fixed_spacing_bool   ,
    indent-width       = \l_galley_par_indent_dim       ,
    left-skip          = \l_galley_line_left_skip       ,
    right-skip         = \l_galley_line_right_skip      ,
    start-skip         = \l_galley_par_begin_skip       ,
    stretch-last-line  = \l_galley_par_stretch_last_bool
  }
  {
    \tl_put_left:Nx \l_galley_justification_other_tl
      {
        \skip_set:Nn \exp_not:N \l_galley_par_end_skip
          { \skip_use:N \l_galley_par_end_skip }
        \bool_if:NTF \l_galley_fixed_spacing_bool
          { \bool_set_true:N  \exp_not:N \l_galley_fixed_spacing_bool }
          { \bool_set_false:N \exp_not:N \l_galley_fixed_spacing_bool }
        \galley_set_interword_spacing:N
          \exp_not:N \l_galley_fixed_spacing_bool
        \dim_set:Nn \exp_not:N \l_galley_par_indent_dim
          { \dim_use:N \l_galley_par_indent_dim  }
        \skip_set:Nn \l_galley_line_left_skip
          { \skip_use:N \l_galley_line_left_skip }
        \skip_set:Nn \exp_not:N \l_galley_line_right_skip
          { \skip_use:N \l_galley_line_right_skip }
        \skip_set:Nn \exp_not:N \l_galley_par_begin_skip
```

```
          { \skip_use:N \l_galley_par_begin_skip }
        \int_set:Nn \exp_not:N \l_galley_last_line_fit_int
          { \int_use:N \l_galley_last_line_fit_int }
        \skip_set:Nn \exp_not:N \@rightskip
          { \skip_use:N \l_galley_line_right_skip }
      }
    \tl_gput_right:Nn \g_galley_restore_running_tl
      { \l_galley_justification_other_tl }
    \AssignTemplateKeys
    \galley_set_interword_spacing:N \l_galley_fixed_spacing_bool
    \bool_if:NTF \l_galley_par_stretch_last_bool
      { \int_set_eq:NN \l_galley_last_line_fit_int \c_one_thousand }
      { \int_zero:N \l_galley_last_line_fit_int }
    \skip_set:Nn \@rightskip { \l_galley_line_right_skip }
  }
```

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.

```
\DeclareInstance { justification } { justified } { std }
  { indent-width = 15 pt }
\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.

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

The other standard justification schemes are for text which is either centred or ragged. The settings here are taken from the LaTeX 2_ε 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.

```
\DeclareInstance { justification } { centered } { std }
  {
    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
  }
\DeclareInstance { justification } { ragged-left } { std }
  {
    end-skip           = 0 pt            ,
```

```
    fixed-word-spacing = true            ,
    indent-width       = 0 pt            ,
    left-skip          = 0 pt plus 2 em  ,
    right-skip         = 0 pt
  }
\DeclareInstance { justification } { ragged-right } { std }
  {
    end-skip           = 0 pt plus 1 fil ,
    fixed-word-spacing = true            ,
    indent-width       = 0 pt            ,
    left-skip          = 0 pt            ,
    right-skip         = 0 pt plus 2 em
  }
```

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

```
\DeclareInstance { justification } { centering } { std }
  {
    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
  }
```

`\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.

```
\DeclareTemplateInterface { justification } { compound } { 0 }
  {
    first-paragraph  : instance { justification } ,
    other-paragraphs : instance { justification }
  }
\DeclareTemplateCode { justification } { compound } { 0 }
  {
    first-paragraph  = \galley_justification_first: ,
    other-paragraphs = \galley_justification_other:
  }
  {
    \AssignTemplateKeys
    \galley_justification_first:
    \tl_set:Nn \l_galley_justification_other_tl
      { \galley_justification_other: }
    \tl_gput_right:Nn \g_galley_restore_running_tl
      { \l_galley_justification_other_tl }
  }
```

(*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.

```
\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.

```
\DeclareTemplateInterface { line-breaking } { std } { 0 }
  {
    badness                : integer = 1000   ,
    binop-penalty          : integer = 700    ,
    double-hyphen-demerits : integer = 10 000 ,
    emergency-stretch      : skip    = 0 pt   ,
    final-hyphen-demerits  : integer = 5000   ,
    fuzz                   : length  = 0.1 pt ,
    line-penalty           : integer = 10     ,
    mismatch-demerits      : integer = 10 000 ,
    pretolerance           : integer = 100    ,
    relation-penalty       : integer = 500    ,
    tolerance              : integer = 200
  }
\DeclareTemplateCode{ line-breaking } { std } { 0 }
  {
    badness                = \l_galley_linebreak_badness_int      ,
    binop-penalty          = \l_galley_binop_penalty_int          ,
    double-hyphen-demerits = \l_galley_double_hyphen_demerits_int ,
    emergency-stretch      = \l_galley_emergency_stretch_skip     ,
    final-hyphen-demerits  = \l_galley_final_hyphen_demerits_int  ,
    fuzz                   = \l_galley_linebreak_fuzz_dim         ,
    line-penalty           = \l_galley_linebreak_penalty_int      ,
    mismatch-demerits      = \l_galley_mismatch_demerits_int      ,
    pretolerance           = \l_galley_linebreak_pretolerance_int ,
    relation-penalty       = \l_galley_relation_penalty_int       ,
    tolerance              = \l_galley_linebreak_tolerance_int
  }
  { \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.

```
\DeclareInstance { line-breaking } { std } { std } { }
\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.

```
\DeclareObjectType { paragraph-breaking } { 0 }
\DeclareTemplateInterface { paragraph-breaking } { std } { 0 }
  {
    badness               : integer = 1000   ,
    broken-penalty        : integer = 100    ,
    club-penalty          : integer = 150    ,
    display-club-penalty  : integer = 150    ,
    display-widow-penalty : integer = 150    ,
    fuzz                  : length  = 0.1 pt ,
    interline-penalty     : integer = 0      ,
    post-display-penalty  : integer = 0      ,
    pre-display-penalty   : integer = 10 000 ,
    widow-penalty         : integer = 150
  }
\DeclareTemplateCode { paragraph-breaking } { std } { 0 }
  {
    badness               = \l_galley_parbreak_badness_int      ,
    broken-penalty        = \l_galley_broken_penalty_int        ,
    club-penalty          = \l_galley_club_penalty_int          ,
    display-club-penalty  = \l_galley_display_club_penalty_int  ,
    display-widow-penalty = \l_galley_display_widow_penalty_int ,
    fuzz                  = \l_galley_parbreak_fuzz_dim         ,
    interline-penalty     = \l_galley_interline_penalty_int     ,
    post-display-penalty  = \l_galley_post_display_penalty_int  ,
    pre-display-penalty   = \l_galley_pre_display_penalty_int   ,
    widow-penalty         = \l_galley_widow_penalty_int
  }
  {
    \AssignTemplateKeys
    \galley_set_club_penalties:V          \l_galley_club_penalty_int
    \galley_set_display_club_penalties:V  \l_galley_display_club_penalty_int
    \galley_set_display_widow_penalties:V \l_galley_display_widow_penalty_int
    \galley_set_interline_penalty:n       \l_galley_interline_penalty_int
    \galley_set_widow_penalties:V         \l_galley_widow_penalty_int
  }
```

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

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

```
\DeclareInstance { paragraph-breaking } { std } { std } { }
\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.

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

```
    post-display-penalty = 10 000
  }
\DeclareInstance { paragraph-breaking } { nolone } { std }
  {
    club-penalty          = 10 000 ,
    display-club-penalty  = 10 000 ,
    display-widow-penalty = 10 000 ,
    widow-penalty         = 10 000
  }
```

\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

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.

```
\DeclareTemplateInterface { paragraph-breaking } { single } { 0 }
  {
    badness               : tokenlist ,
    broken-penalty        : tokenlist ,
    club-penalty          : tokenlist ,
    display-club-penalty  : tokenlist ,
    display-widow-penalty : tokenlist ,
    fuzz                  : tokenlist ,
    interline-penalty     : tokenlist ,
    post-display-penalty  : tokenlist ,
    pre-display-penalty   : tokenlist ,
    widow-penalty         : tokenlist
  }
\DeclareTemplateCode { paragraph-breaking } { single } { 0 }
  {
    badness               = \l_galley_parbreak_badness_tl        ,
    broken-penalty        = \l_galley_broken_penalty_tl          ,
    club-penalty          = \l_galley_club_penalties_tl          ,
    display-club-penalty  = \l_galley_display_club_penalties_tl  ,
    display-widow-penalty = \l_galley_display_widow_penalties_tl ,
    fuzz                  = \l_galley_parbreak_fuzz_tl           ,
    interline-penalty     = \l_galley_interline_penalty_tl       ,
    post-display-penalty  = \l_galley_post_display_penalty_tl    ,
    pre-display-penalty   = \l_galley_pre_display_penalty_tl     ,
    widow-penalty         = \l_galley_widow_penalties_tl
  }
  {
    \AssignTemplateKeys
```

The fuzz and interline penalties are handled explicitly as they have particular requirements.

```
    \tl_if_empty:NF \l_galley_interline_penalty_tl
      {
        \tl_gput_right:Nx \g_galley_par_after_hook_tl
          {
            \int_set:Nn \exp_not:N \l_galley_interline_penalty_int
              { \galley_interline_penalty: }
```

```
          }
        \int_set:Nn \l_galley_interline_penalty_int
          { \l_galley_interline_penalty_tl }
      }
    \tl_if_empty:NF \l_galley_parbreak_fuzz_tl
      {
        \tl_gput_right:Nx \g_galley_par_after_hook_tl
          {
            \dim_set:Nn \exp_not:N \l_galley_parbreak_fuzz_dim
              { \dim_use:N \l_galley_parbreak_fuzz_dim }
          }
        \dim_set:Nn \l_galley_parbreak_fuzz_dim { \l_galley_parbreak_fuzz_tl }
      }
```

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

```
    \seq_map_inline:Nn \c_galley_parbreak_single_seq
      {
        \tl_if_empty:cF { l_galley_ ##1 _tl }
          {
            \tl_gput_right:Nx \g_galley_par_after_hook_tl
              {
                \int_set:Nn \exp_not:c { l_galley_ ##1 _int }
                  { \int_use:c { l_galley_ ##1 _int } }
              }
            \int_set:cn { l_galley_ ##1 _int }
              { \tl_use:c { l_galley_ ##1 _tl } }
          }
      }
```

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

```
    \seq_map_inline:Nn \c_galley_parbreak_multi_seq
      {
        \tl_if_empty:cF { l_galley_ ##1 _tl }
          {
            \use:c { galley_save_ ##1 :N } \l_galley_tmpa_clist
            \tl_gput_right:Nx \g_galley_par_after_hook_tl
              {
                \exp_not:c { galley_set_ ##1 :n }
                  { \exp_not:o \l_galley_tmpa_clist }
              }
            \use:c { galley_set_ ##1 :v } { l_galley_ ##1 _tl }
          }
      }
  }
\seq_new:N \c_galley_parbreak_multi_seq
\seq_gput_right:Nn \c_galley_parbreak_multi_seq { club_penalties }
\seq_gput_right:Nn \c_galley_parbreak_multi_seq { display_club_penalties }
\seq_gput_right:Nn \c_galley_parbreak_multi_seq { display_widow_penalties }
\seq_gput_right:Nn \c_galley_parbreak_multi_seq { widow_penalties }
```

```
\seq_new:N \c_galley_parbreak_single_seq
\seq_gput_right:Nn \c_galley_parbreak_single_seq { parbreak_badness }
\seq_gput_right:Nn \c_galley_parbreak_single_seq { broken_penalty }
\seq_gput_right:Nn \c_galley_parbreak_single_seq { post_display_penalty }
\seq_gput_right:Nn \c_galley_parbreak_single_seq { pre_display_penalty }
```

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

```
\DeclareInstance { paragraph-breaking } { single-std } { single } { }
\DeclareInstance { paragraph-breaking } { single-nobreak } { single }
  {
    interline-penalty    = 10 000 ,
    post-display-penalty = 10 000
  }
\DeclareInstance { paragraph-breaking } { single-noclub } { single }
  {
    club-penalty         = 10 000 ,
    display-club-penalty = 10 000
  }
\DeclareInstance { paragraph-breaking } { single-nolone } { single }
  {
    club-penalty          = 10 000 ,
    display-club-penalty  = 10 000 ,
    display-widow-penalty = 10 000 ,
    widow-penalty         = 10 000
  }
\DeclareInstance { paragraph-breaking } { single-nowidow } { single }
  {
    display-widow-penalty = 10 000 ,
    widow-penalty         = 10 000
  }
```

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.

```
\DeclareObjectType { display-begin } { 0 }
\DeclareObjectType { display-end }   { 0 }
\DeclareTemplateInterface { display-begin } { std } { 0 }
  {
    par-penalty : tokenlist ,
    par-space   : tokenlist ,
    penalty     : tokenlist ,
    space       : tokenlist
  }
\DeclareTemplateInterface { display-end } { std } { 0 }
```

```
  {
    par-penalty : tokenlist ,
    par-space   : tokenlist ,
    penalty     : tokenlist ,
    space       : tokenlist
  }
\DeclareTemplateCode { display-begin } { std } { 0 }
  {
    par-penalty = \l_galley_display_begin_par_vpenalty_tl ,
    par-space   = \l_galley_display_begin_par_vspace_tl   ,
    penalty     = \l_galley_display_begin_vpenalty_tl     ,
    space       = \l_galley_display_begin_vspace_tl
  }
  {
   \AssignTemplateKeys
    \tl_if_empty:NF \l_galley_display_begin_par_vpenalty_tl
      {
        \tl_set:Nx \l_galley_display_begin_par_vpenalty_tl
          { \int_eval:n { \l_galley_display_begin_par_vpenalty_tl } }
      }
    \tl_if_empty:NF \l_galley_display_begin_par_vspace_tl
      {
        \tl_set:Nx \l_galley_display_begin_par_vspace_tl
          { \skip_eval:n { \l_galley_display_begin_par_vspace_tl } }
      }
    \tl_if_empty:NF \l_galley_display_begin_vpenalty_tl
      {
        \tl_set:Nx \l_galley_display_begin_vpenalty_tl
          { \int_eval:n { \l_galley_display_begin_vpenalty_tl } }
      }
    \tl_if_empty:NF \l_galley_display_begin_vspace_tl
      {
        \tl_set:Nx \l_galley_display_begin_vspace_tl
          { \skip_eval:n { \l_galley_display_begin_vspace_tl } }
      }
  }
\DeclareTemplateCode { display-end } { std } { 0 }
  {
    par-penalty = \l_galley_display_end_par_vpenalty_tl ,
    par-space   = \l_galley_display_end_par_vspace_tl   ,
    penalty     = \l_galley_display_end_vpenalty_tl     ,
    space       = \l_galley_display_end_vspace_tl
  }
  {
    \AssignTemplateKeys
    \tl_if_empty:NF \l_galley_display_end_par_vpenalty_tl
      {
        \tl_set:Nx \l_galley_display_end_par_vpenalty_tl
          { \int_eval:n { \l_galley_display_end_par_vpenalty_tl } }
      }
```

```
    \tl_if_empty:NF \l_galley_display_end_par_vspace_tl
      {
        \tl_set:Nx \l_galley_display_end_par_vspace_tl
          { \skip_eval:n { \l_galley_display_end_par_vspace_tl } }
      }
    \tl_if_empty:NF \l_galley_display_end_vpenalty_tl
      {
        \tl_set:Nx \l_galley_display_end_vpenalty_tl
          { \int_eval:n { \l_galley_display_end_vpenalty_tl } }
      }
    \tl_if_empty:NF \l_galley_display_end_vspace_tl
      {
        \tl_set:Nx \l_galley_display_end_vspace_tl
          { \skip_eval:n { \l_galley_display_end_vspace_tl } }
      }
  }
⟨/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.

G

I

L

P

R

S

T

U