Guile-GNOME: Pango

version 2.15.93, updated 2 September 2007

Owen Taylor
Behdad Esfahbod
many others

Short Contents

1 Overview

The Pango wrapper for Guile is a part of Guile-GNOME. Maybe write more here at some point.

2 Coverage Maps

Unicode character range coverage storage

2.1 Overview

It is often necessary in Pango to determine if a particular font can represent a particular character, and also how well it can represent that character. The `<pango-coverage>` is a data structure that is used to represent that information.

2.2 Usage

`pango-coverage-new` ⇒ (*ret* `<pango-coverage*>`) [Function]

Create a new `<pango-coverage>`

ret	the newly allocated `<pango-coverage>`, initialized to '`PANGO_COVERAGE_NONE`' with a reference count of one, which should be freed with `pango-coverage-unref`.

`pango-coverage-get` (*self* `<pango-coverage*>`) (*index_* `int`) ⇒ (*ret* `<pango-coverage-level>`) [Function]

Determine whether a particular index is covered by *coverage*

coverage	a `<pango-coverage>`
index	the index to check
ret	the coverage level of *coverage* for character *index*.

`pango-coverage-max` (*self* `<pango-coverage*>`) (*other* `<pango-coverage*>`) [Function]

Set the coverage for each index in *coverage* to be the max (better) value of the current coverage for the index and the coverage for the corresponding index in *other*.

coverage	a `<pango-coverage>`
other	another `<pango-coverage>`

`pango-coverage-set` (*self* `<pango-coverage*>`) (*index_* `int`) (*level* `<pango-coverage-level>`) [Function]

Modify a particular index within *coverage*

coverage	a `<pango-coverage>`
index	the index to modify
level	the new level for *index*

`pango-coverage-to-bytes` (*self* `<pango-coverage*>`) (*bytes* `<guchar**>`) ⇒ (*n_bytes* `int`) [Function]

Convert a `<pango-coverage>` structure into a flat binary format

coverage	a `<pango-coverage>`
bytes	location to store result (must be freed with `g-free`)
n-bytes	location to store size of result

`pango-coverage-from-bytes` (*bytes* `<guchar*>`) (*n_bytes* `int`) [Function]
⇒ (*ret* `<pango-coverage*>`)

Convert data generated from `pango-converage-to-bytes` back to a `<pango-coverage>`

bytes	binary data representing a `<pango-coverage>`
n-bytes	the size of *bytes* in bytes
ret	a newly allocated `<pango-coverage>`, or '`#f`' if the data was invalid.

3 Fonts

Structures representing abstract fonts

3.1 Overview

Pango supports a flexible architecture where a particular rendering architecture can supply an implementation of fonts. The `<pango-font>` structure represents an abstract rendering-system-independent font. Pango provides routines to list available fonts, and to load a font of a given description.

3.2 Usage

`<pango-font-description>` [Class]

`<pango-font-metrics>` [Class]

`<pango-font>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`<pango-font-family>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`<pango-font-face>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`<pango-font-map>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`pango-font-description-new` ⇒ (*ret* `<pango-font-description>`) [Function]

Creates a new font description structure with all fields unset.

ret	the newly allocated `<pango-font-description>`, which should be freed using `pango-font-description-free`.

`pango-font-description-hash` (*self* `<pango-font-description>`) ⇒ (*ret* `unsigned-int`) [Function]

Computes a hash of a `<pango-font-description>` structure suitable to be used, for example, as an argument to `g-hash-table-new`. The hash value is independent of *desc->mask*.

desc	a `<pango-font-description>`
ret	the hash value.

`pango-font-description-equal` (*self* `<pango-font-description>`) (*desc2* `<pango-font-description>`) ⇒ (*ret* `bool`) [Function]

Compares two font descriptions for equality. Two font descriptions are considered equal if the fonts they describe are provably identical. This means that their masks do not have to match, as long as other fields are all the same. (Two font descriptions may result in identical fonts being loaded, but still compare '`#f`'.)

desc1	a `<pango-font-description>`
desc2	another `<pango-font-description>`
ret	'`#t`' if the two font descriptions are identical, '`#f`' otherwise.

`pango-font-description-set-family` [Function]
(*self* `<pango-font-description>`) (*family* `mchars`)

Sets the family name field of a font description. The family name represents a family of related font styles, and will resolve to a particular `<pango-font-family>`. In some uses of `<pango-font-description>`, it is also possible to use a comma separated list of family names for this field.

desc	a `<pango-font-description>`.
family	a string representing the family name.

`pango-font-description-get-family` [Function]
(*self* `<pango-font-description>`) ⇒ (*ret* `mchars`)

Gets the family name field of a font description. See `pango-font-description-set-family`.

desc	a `<pango-font-description>`.
ret	the family name field for the font description, or '`#f`' if not previously set. This has the same life-time as the font description itself and should not be freed.

`pango-font-description-set-style` [Function]
(*self* `<pango-font-description>`) (*style* `<pango-style>`)

Sets the style field of a `<pango-font-description>`. The `<pango-style>` enumeration describes whether the font is slanted and the manner in which it is slanted; it can be either `<pango-style-normal>`, `<pango-style-italic>`, or `<pango-style-oblique>`. Most fonts will either have a italic style or an oblique style, but not both, and font matching in Pango will match italic specifications with oblique fonts and vice-versa if an exact match is not found.

desc	a `<pango-font-description>`
style	the style for the font description

`pango-font-description-get-style` [Function]
(*self* `<pango-font-description>`) ⇒ (*ret* `<pango-style>`)

Gets the style field of a `<pango-font-description>`. See `pango-font-description-set-style`.

desc	a `<pango-font-description>`
ret	the style field for the font description. Use `pango-font-description-get-set-fields` to find out if the field was explicitly set or not.

`pango-font-description-set-variant` [Function]
(*self* `<pango-font-description>`) (*variant* `<pango-variant>`)

Sets the variant field of a font description. The `<pango-variant>` can either be '`PANGO_VARIANT_NORMAL`' or '`PANGO_VARIANT_SMALL_CAPS`'.

desc	a `<pango-font-description>`
variant	the variant type for the font description.

`pango-font-description-get-variant` [Function]
(*self* `<pango-font-description>`) ⇒ (*ret* `<pango-variant>`)

Gets the variant field of a `<pango-font-description>`. See `pango-font-description-set-variant`.

desc	a `<pango-font-description>`.
ret	the variant field for the font description. Use `pango-font-description-get-set-fields` to find out if the field was explicitly set or not.

`pango-font-description-set-weight` [Function]
(*self* `<pango-font-description>`) (*weight* `<pango-weight>`)

Sets the weight field of a font description. The weight field specifies how bold or light the font should be. In addition to the values of the `<pango-weight>` enumeration, other intermediate numeric values are possible.

desc	a `<pango-font-description>`
weight	the weight for the font description.

`pango-font-description-get-weight` [Function]
(*self* `<pango-font-description>`) ⇒ (*ret* `<pango-weight>`)

Gets the weight field of a font description. See `pango-font-description-set-weight`.

desc	a `<pango-font-description>`
ret	the weight field for the font description. Use `pango-font-description-get-set-fields` to find out if the field was explicitly set or not.

`pango-font-description-set-stretch` [Function]
(*self* `<pango-font-description>`) (*stretch* `<pango-stretch>`)

Sets the stretch field of a font description. The stretch field specifies how narrow or wide the font should be.

desc	a `<pango-font-description>`
stretch	the stretch for the font description

`pango-font-description-get-stretch` [Function]
(*self* `<pango-font-description>`) ⇒ (*ret* `<pango-stretch>`)

Gets the stretch field of a font description. See `pango-font-description-set-stretch`.

desc	a `<pango-font-description>`.
ret	the stretch field for the font description. Use `pango-font-description-get-set-fields` to find out if the field was explicitly set or not.

`pango-font-description-set-size` [Function]
(*self* `<pango-font-description>`) (*size* `int`)

Sets the size field of a font description in fractional points. This is mutually exclusive with `pango-font-description-set-absolute-size`.

desc	a `<pango-font-description>`
size	the size of the font in points, scaled by PANGO_SCALE. (That is, a *size* value of 10 * PANGO_SCALE is a 10 point font. The conversion factor between points and device units depends on system configuration and the output device. For screen display, a logical DPI of 96 is common, in which case a 10 point font corresponds to a 10 * (96 / 72) = 13.3 pixel font. Use `pango-font-description-set-absolute-size` if you need a particular size in device units.

`pango-font-description-get-size` [Function]
(*self* `<pango-font-description>`) ⇒ (*ret* `int`)

Gets the size field of a font description. See `pango-font-description-set-size`.

desc	a `<pango-font-description>`
ret	the size field for the font description in points or device units. You must call `pango-font-description-get-size-is-absolute` to find out which is the case. Returns 0 if the size field has not previously been set or it has been set to 0 explicitly. Use `pango-font-description-get-set-fields` to find out if the field was explicitly set or not.

`pango-font-description-unset-fields` [Function]
(*self* `<pango-font-description>`) (*to_unset* `<pango-font-mask>`)

Unsets some of the fields in a `<pango-font-description>`. The unset fields will get back to their default values.

desc	a `<pango-font-description>`
to-unset	bitmask of fields in the *desc* to unset.

`pango-font-description-merge` (*self* `<pango-font-description>`) [Function]
(*desc_to_merge* `<pango-font-description>`) (*replace_existing* `bool`)

Merges the fields that are set in *desc-to-merge* into the fields in *desc*. If *replace-existing* is '`#f`', only fields in *desc* that are not already set are affected. If '`#t`', then fields that are already set will be replaced as well.

desc	a `<pango-font-description>`
desc-to-merge	the `<pango-font-description>` to merge from
replace-existing	if '`#t`', replace fields in *desc* with the corresponding values from *desc-to-merge*, even if they are already exist.

`pango-font-description-merge-static` [Function]
  (*self* `<pango-font-description>`)
  (*desc_to_merge* `<pango-font-description>`) (*replace_existing* `bool`)

Like `pango-font-description-merge`, but only a shallow copy is made of the family name and other allocated fields. *desc* can only be used until *desc-to-merge* is modified or freed. This is meant to be used when the merged font description is only needed temporarily.

desc	a `<pango-font-description>`
desc-to-merge	the `<pango-font-description>` to merge from
replace-existing	if '`#t`', replace fields in *desc* with the corresponding values from *desc-to-merge*, even if they are already exist.

`pango-font-description-better-match` [Function]
  (*self* `<pango-font-description>`)
  (*old_match* `<pango-font-description>`)
  (*new_match* `<pango-font-description>`) ⇒ (*ret* `bool`)

Determines if the style attributes of *new-match* are a closer match for *desc* than *old-match*, or if *old-match* is '`#f`', determines if *new-match* is a match at all. Approximate matching is done for weight and style; other attributes must match exactly.

desc	a `<pango-font-description>`
old-match	a `<pango-font-description>`, or '`#f`'
new-match	a `<pango-font-description>`
ret	'`#t`' if *new-match* is a better match

`pango-font-description-to-string` [Function]
  (*self* `<pango-font-description>`) ⇒ (*ret* `mchars`)

Creates a string representation of a font description. See `pango-font-description-from-string` for a description of the format of the string representation. The family list in the string description will only have a terminating comma if the last word of the list is a valid style option.

desc	a `<pango-font-description>`
ret	a new string that must be freed with `g-free`.

`pango-font-description-to-filename` [Function]
  (*self* `<pango-font-description>`) ⇒ (*ret* `mchars`)

Creates a filename representation of a font description. The filename is identical to the result from calling `pango-font-description-to-string`, but with underscores instead of characters that are untypical in filenames, and in lower case only.

desc	a `<pango-font-description>`
ret	a new string that must be freed with `g-free`.

`pango-font-metrics-get-ascent` (*self* `<pango-font-metrics>`) [Function]
⇒ (*ret* `int`)

Gets the ascent from a font metrics structure. The ascent is the distance from the baseline to the logical top of a line of text. (The logical top may be above or below the top of the actual drawn ink. It is necessary to lay out the text to figure where the ink will be.)

metrics	a `<pango-font-metrics>` structure
ret	the ascent, in Pango units. (1 point == '`PANGO_SCALE`' Pango units.)

`pango-font-metrics-get-descent` (*self* `<pango-font-metrics>`) [Function]
⇒ (*ret* `int`)

Gets the descent from a font metrics structure. The descent is the distance from the baseline to the logical bottom of a line of text. (The logical bottom may be above or below the bottom of the actual drawn ink. It is necessary to lay out the text to figure where the ink will be.)

metrics	a `<pango-font-metrics>` structure
ret	the descent, in Pango units. (1 point == '`PANGO_SCALE`' Pango units.)

`pango-font-find-shaper` (*self* `<pango-font>`) [Function]
(*language* `<pango-language>`) (*ch* `unsigned-int32`)
⇒ (*ret* `<pango-engine-shape*>`)

`find-shaper` [Method]

Finds the best matching shaper for a font for a particular language tag and character point.

font	a `<pango-font>`
language	the language tag
ch	a Unicode character.
ret	the best matching shaper.

`pango-font-describe` (*self* `<pango-font>`) [Function]
⇒ (*ret* `<pango-font-description>`)

`describe` [Method]

Returns a description of the font, with font size set in points. Use `pango-font-describe-with-absolute-size` if you want the font size in device units.

font	a `<pango-font>`
ret	a newly-allocated `<pango-font-description>` object.

`pango-font-get-coverage` (*self* `<pango-font>`) [Function]
(*language* `<pango-language>`) ⇒ (*ret* `<pango-coverage*>`)

`get-coverage` [Method]

Computes the coverage map for a given font and language tag.

font	a `<pango-font>`
language	the language tag
ret	a newly-allocated `<pango-coverage>` object.

`pango-font-get-glyph-extents` (*self* `<pango-font>`) (*glyph* `unsigned-int32`) (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) [Function]
`get-glyph-extents` [Method]

Gets the logical and ink extents of a glyph within a font. The coordinate system for each rectangle has its origin at the base line and horizontal origin of the character with increasing coordinates extending to the right and down. The macros `pango-ascent`, `pango-descent`, `pango-lbearing`, and `pango-rbearing` can be used to convert from the extents rectangle to more traditional font metrics. The units of the rectangles are in 1/PANGO_SCALE of a device unit.

font	a `<pango-font>`
glyph	the glyph index
ink-rect	rectangle used to store the extents of the glyph as drawn or '`#f`' to indicate that the result is not needed.
logical-rect	rectangle used to store the logical extents of the glyph or '`#f`' to indicate that the result is not needed.

`pango-font-get-metrics` (*self* `<pango-font>`) (*language* `<pango-language>`) ⇒ (*ret* `<pango-font-metrics>`) [Function]
`get-metrics` [Method]

Gets overall metric information for a font. Since the metrics may be substantially different for different scripts, a language tag can be provided to indicate that the metrics should be retrieved that correspond to the script(s) used by that language.

font	a `<pango-font>`
language	language tag used to determine which script to get the metrics for, or '`#f`' to indicate to get the metrics for the entire font.
ret	a `<pango-font-metrics>` object. The caller must call `pango-font-metrics-unref` when finished using the object.

`pango-font-get-font-map` (*self* `<pango-font>`) ⇒ (*ret* `<pango-font-map>`) [Function]
`get-font-map` [Method]

Gets the font map for which the font was created.

font	a `<pango-font>`
ret	the `<pango-font-map>` for the font

Since 1.10

`pango-font-family-get-name` (*self* `<pango-font-family>`) ⇒ (*ret* `mchars`) [Function]
`get-name` [Method]

Gets the name of the family. The name is unique among all fonts for the font backend and can be used in a `<pango-font-description>` to specify that a face from this family is desired.

family a `<pango-font-family>`

ret the name of the family. This string is owned by the family object and must not be modified or freed.

`pango-font-family-is-monospace` (*self* `<pango-font-family>`) ⇒ (*ret* `bool`) [Function]

`is-monospace` [Method]

A monospace font is a font designed for text display where the the characters form a regular grid. For Western languages this would mean that the advance width of all characters are the same, but this categorization also includes Asian fonts which include double-width characters: characters that occupy two grid cells. `g-unichar-iswide` returns a result that indicates whether a character is typically double-width in a monospace font.

The best way to find out the grid-cell size is to call `pango-font-metrics-get-approximate-digit-width`, since the results of `pango-font-metrics-get-approximate-char-width` may be affected by double-width characters.

family a `<pango-font-family>`

ret '`#t`' if the family is monospace.

Since 1.4

`pango-font-family-list-faces` (*self* `<pango-font-family>`) (*faces* `<pango-font-face***>`) ⇒ (*n_faces* `int`) [Function]

`list-faces` [Method]

Lists the different font faces that make up *family*. The faces in a family share a common design, but differ in slant, weight, width and other aspects.

family a `<pango-font-family>`

faces location to store an array of pointers to `<pango-font-face>` objects, or '`#f`'. This array should be freed with `g-free` when it is no longer needed.

n-faces location to store number of elements in *faces*.

`pango-font-face-get-face-name` (*self* `<pango-font-face>`) ⇒ (*ret* `mchars`) [Function]

`get-face-name` [Method]

Gets a name representing the style of this face among the different faces in the `<pango-font-family>` for the face. This name is unique among all faces in the family and is suitable for displaying to users.

face a `<pango-font-face>`.

ret the face name for the face. This string is owned by the face object and must not be modified or freed.

`pango-font-face-list-sizes` (*self* `<pango-font-face>`) (*sizes* `<int**>`) ⇒ (*n_sizes* `int`) [Function]

`list-sizes` [Method]

List the available sizes for a font. This is only applicable to bitmap fonts. For scalable fonts, stores '`#f`' at the location pointed to by *sizes* and 0 at the location pointed to by *n-sizes*. The sizes returned are in Pango units and are sorted in ascending order.

face	a `<pango-font-face>`.
sizes	location to store a pointer to an array of int. This array should be freed with `g-free`.
n-sizes	location to store the number of elements in *sizes*

Since 1.4

`pango-font-face-describe` (*self* `<pango-font-face>`) ⇒ (*ret* `<pango-font-description>`) [Function]

`describe` [Method]

Returns the family, style, variant, weight and stretch of a `<pango-font-face>`. The size field of the resulting font description will be unset.

face	a `<pango-font-face>`
ret	a newly-created `<pango-font-description>` structure holding the description of the face. Use `pango-font-description-free` to free the result.

`pango-font-map-load-font` (*self* `<pango-font-map>`) (*context* `<pango-context*>`) (*desc* `<pango-font-description>`) ⇒ (*ret* `<pango-font>`) [Function]

`load-font` [Method]

Load the font in the fontmap that is the closest match for *desc*.

fontmap	a `<pango-font-map>`
context	the `<pango-context>` the font will be used with
desc	a `<pango-font-description>` describing the font to load
ret	the font loaded, or '`#f`' if no font matched.

`pango-font-map-load-fontset` (*self* `<pango-font-map>`) (*context* `<pango-context*>`) (*desc* `<pango-font-description>`) (*language* `<pango-language>`) ⇒ (*ret* `<pango-fontset*>`) [Function]

`load-fontset` [Method]

Load a set of fonts in the fontmap that can be used to render a font matching *desc*.

fontmap	a `<pango-font-map>`
context	the `<pango-context>` the font will be used with
desc	a `<pango-font-description>` describing the font to load
language	a `<pango-language>` the fonts will be used for
ret	the fontset, or '`#f`' if no font matched.

`pango-font-map-list-families` (*self* `<pango-font-map>`) (*families* `<pango-font-family***>`) ⇒ (*n_families* `int`) [Function]

`list-families` [Method]

List all families for a fontmap.

fontmap	a `<pango-font-map>`

families	location to store a pointer to an array of `<pango-font-family>` *. This array should be freed with `g-free`.
n-families	location to store the number of elements in *families*

4 Glyph Storage

Structures for storing information about glyphs

4.1 Overview

`pango-shape` produces a string of glyphs which can be measured or drawn to the screen. The following structures are used to store information about glyphs.

4.2 Usage

`<pango-glyph-string>` [Class]

`pango-matrix-translate` (*self* `<pango-matrix*>`) (*tx* `double`) (*ty* `double`) [Function]

Changes the transformation represented by *matrix* to be the transformation given by first translating by (*tx*, *ty*) then applying the original transformation.

matrix	a `<pango-matrix>`
tx	amount to translate in the X direction
ty	amount to translate in the Y direction

Since 1.6

`pango-matrix-scale` (*self* `<pango-matrix*>`) (*scale_x* `double`) (*scale_y* `double`) [Function]

Changes the transformation represented by *matrix* to be the transformation given by first scaling by *sx* in the X direction and *sy* in the Y direction then applying the original transformation.

matrix	a `<pango-matrix>`
scale-x	amount to scale by in X direction
scale-y	amount to scale by in Y direction

Since 1.6

`pango-matrix-rotate` (*self* `<pango-matrix*>`) (*degrees* `double`) [Function]

Changes the transformation represented by *matrix* to be the transformation given by first rotating by *degrees* degrees counter-clockwise then applying the original transformation.

matrix	a `<pango-matrix>`
degrees	degrees to rotate counter-clockwise

Since 1.6

`pango-matrix-concat` (*self* `<pango-matrix*>`) (*new_matrix* `<pango-matrix*>`) [Function]

Changes the transformation represented by *matrix* to be the transformation given by first applying transformation given by *new-matrix* then applying the original transformation.

matrix a `<pango-matrix>`

new-matrix
a `<pango-matrix>`

Since 1.6

`pango-matrix-get-font-scale-factor` (*self* `<pango-matrix*>`) ⇒ (*ret* `double`) [Function]

Returns the scale factor of a matrix on the height of the font. That is, the scale factor in the direction perpendicular to the vector that the X coordinate is mapped to.

matrix a `<pango-matrix>`, may be '`#f`'

ret the scale factor of *matrix* on the height of the font, or 1.0 if *matrix* is '`#f`'.

Since 1.12

`pango-glyph-string-new` ⇒ (*ret* `<pango-glyph-string>`) [Function]

Create a new `<pango-glyph-string>`.

ret the newly allocated `<pango-glyph-string>`, which should be freed with `pango-glyph-string-free`.

`pango-glyph-string-set-size` (*self* `<pango-glyph-string>`) (*new_len* `int`) [Function]

Resize a glyph string to the given length.

string a `<pango-glyph-string>`.

new-len the new length of the string.

`pango-glyph-string-extents` (*self* `<pango-glyph-string>`) (*font* `<pango-font>`) (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) [Function]

Compute the logical and ink extents of a glyph string. See the documentation for `pango-font-get-glyph-extents` for details about the interpretation of the rectangles.

glyphs a `<pango-glyph-string>`

font a `<pango-font>`

ink-rect rectangle used to store the extents of the glyph string as drawn or '`#f`' to indicate that the result is not needed.

logical-rect
rectangle used to store the logical extents of the glyph string or '`#f`' to indicate that the result is not needed.

`pango-glyph-string-extents-range` (*self* `<pango-glyph-string>`) (*start* `int`) (*end* `int`) (*font* `<pango-font>`) (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) [Function]

Computes the extents of a sub-portion of a glyph string. The extents are relative to the start of the glyph string range (the origin of their coordinate system is at the start of the range, not at the start of the entire glyph string).

glyphs a `<pango-glyph-string>`

start start index

end end index (the range is the set of bytes with indices such that start `<=` index `<` end)

font a `<pango-font>`

ink-rect rectangle used to store the extents of the glyph string range as drawn or '`#f`' to indicate that the result is not needed.

logical-rect
rectangle used to store the logical extents of the glyph string range or '`#f`' to indicate that the result is not needed.

`pango-glyph-string-get-width` (*self* `<pango-glyph-string>`) [Function]
⇒ (*ret* `int`)

Computes the logical width of the glyph string as can also be computed using `pango-glyph-string-extents`. However, since this only computes the width, it's much faster. This is in fact only a convenience function that computes the sum of geometry.width for each glyph in the *glyphs*.

glyphs a `<pango-glyph-string>`

ret the logical width of the glyph string.

Since 1.14

`pango-glyph-string-index-to-x` (*self* `<pango-glyph-string>`) [Function]
(*text* `mchars`) (*length* `int`) (*analysis* `<pango-analysis*>`) (*index_* `int`)
(*trailing* `bool`) ⇒ (*x_pos* `int`)

Converts from character position to x position. (X position is measured from the left edge of the run). Character positions are computed by dividing up each cluster into equal portions.

glyphs the glyphs return from `pango-shape`

text the text for the run

length the number of bytes (not characters) in *text*.

analysis the analysis information return from `pango-itemize`

index the byte index within *text*

trailing whether we should compute the result for the beginning or end of the character.

x-pos location to store result

`pango-glyph-string-x-to-index` (*self* `<pango-glyph-string>`) [Function]
(*text* `mchars`) (*length* `int`) (*analysis* `<pango-analysis*>`) (*x_pos* `int`)
⇒ (*index_* `int`) (*trailing* `int`)

Convert from x offset to character position. Character positions are computed by dividing up each cluster into equal portions. In scripts where positioning within a cluster is not allowed (such as Thai), the returned value may not be a valid cursor position; the caller must combine the result with the logical attributes for the text to compute the valid cursor position.

glyphs	the glyphs return from `pango-shape`
text	the text for the run
length	the number of bytes (not characters) in text.
analysis	the analysis information return from `pango-itemize`
x-pos	the x offset (in `<pango-glyph-unit>`)
index	location to store calculated byte index within *text*
trailing	location to store a integer indicating where whether the user clicked on the leading or trailing edge of the character.

`pango-glyph-item-apply-attrs` (*self* `<pango-glyph-item*>`) (*text* `mchars`) (*list* `<pango-attr-list>`) ⇒ (*ret* `gslist-of`) [Function]

Splits a shaped item (PangoGlyphItem) into multiple items based on an attribute list. The idea is that if you have attributes that don't affect shaping, such as color or underline, to avoid affecting shaping, you filter them out (`pango-attr-list-filter`), apply the shaping process and then reapply them to the result using this function.

All attributes that start or end inside a cluster are applied to that cluster; for instance, if half of a cluster is underlined and the other-half strikethrough, then the cluster will end up with both underline and strikethrough attributes. In these cases, it may happen that item->extra_attrs for some of the result items can have multiple attributes of the same type.

This function takes ownership of *glyph-item*; it will be reused as one of the elements in the list.

glyph-item	a shaped item
text	text that *list* applies to
list	a `<pango-attr-list>`
ret	a list of glyph items resulting from splitting *glyph-item*. Free the elements using `pango-glyph-item-free`, the list using `g-slist-free`.

Since 1.2

5 Layout Objects

High-level layout driver objects

5.1 Overview

While complete access to the layout capabilities of Pango is provided using the detailed interfaces for itemization and shaping, using that functionality directly involves writing a fairly large amount of code. The objects and functions in this section provide a high-level driver for formatting entire paragraphs of text at once.

5.2 Usage

`pango-layout-new` (*context* `<pango-context*>`) [Function]
⇒ (*ret* `<pango-layout*>`)

Create a new `<pango-layout>` object with attributes initialized to default values for a particular `<pango-context>`.

context	a `<pango-context>`
ret	the newly allocated `<pango-layout>`, with a reference count of one, which should be freed with `g-object-unref`.

`pango-layout-get-context` (*self* `<pango-layout*>`) [Function]
⇒ (*ret* `<pango-context*>`)

Retrieves the `<pango-context>` used for this layout.

layout	a `<pango-layout>`
ret	the `<pango-context>` for the layout. This does not have an additional refcount added, so if you want to keep a copy of this around, you must reference it yourself.

`pango-layout-context-changed` (*self* `<pango-layout*>`) [Function]

Forces recomputation of any state in the `<pango-layout>` that might depend on the layout's context. This function should be called if you make changes to the context subsequent to creating the layout.

layout	a `<pango-layout>`

`pango-layout-set-text` (*self* `<pango-layout*>`) (*text* `mchars`) [Function]
(*length* `int`)

Sets the text of the layout.

layout	a `<pango-layout>`
text	a valid UTF-8 string
length	maximum length of *text*, in bytes. -1 indicates that the string is nul-terminated and the length should be calculated. The text will also be truncated on encountering a nul-termination even when *length* is positive.

pango-layout-get-text (*self* `<pango-layout*>`) ⇒ (*ret* `mchars`) [Function]

Gets the text in the layout. The returned text should not be freed or modified.

layout	a `<pango-layout>`
ret	the text in the *layout*.

pango-layout-set-markup (*self* `<pango-layout*>`) (*markup* `mchars`) (*length* `int`) [Function]

Same as `pango-layout-set-markup-with-accel`, but the markup text isn't scanned for accelerators.

layout	a `<pango-layout>`
markup	marked-up text
length	length of marked-up text in bytes, or -1 if *markup* is nul-terminated

pango-layout-set-markup-with-accel (*self* `<pango-layout*>`) (*markup* `mchars`) (*length* `int`) (*accel_marker* `unsigned-int32`) ⇒ (*accel_char* `unsigned-int32`) [Function]

Sets the layout text and attribute list from marked-up text (see markup format). Replaces the current text and attribute list.

If *accel-marker* is nonzero, the given character will mark the character following it as an accelerator. For example, *accel-marker* might be an ampersand or underscore. All characters marked as an accelerator will receive a '`PANGO_UNDERLINE_LOW`' attribute, and the first character so marked will be returned in *accel-char*. Two *accel-marker* characters following each other produce a single literal *accel-marker* character.

layout	a `<pango-layout>`
markup	marked-up text (see markup format)
length	length of marked-up text in bytes, or -1 if *markup* is nul-terminated
accel-marker	marker for accelerators in the text
accel-char	return location for first located accelerator, or '`#f`'

pango-layout-set-attributes (*self* `<pango-layout*>`) (*attrs* `<pango-attr-list>`) [Function]

Sets the text attributes for a layout object.

layout	a `<pango-layout>`
attrs	a `<pango-attr-list>`

pango-layout-get-attributes (*self* `<pango-layout*>`) ⇒ (*ret* `<pango-attr-list>`) [Function]

Gets the attribute list for the layout, if any.

layout	a `<pango-layout>`
ret	a `<pango-attr-list>`.

`pango-layout-set-font-description` (*self* `<pango-layout*>`) (*desc* `<pango-font-description>`) [Function]

Sets the default font description for the layout. If no font description is set on the layout, the font description from the layout's context is used.

layout	a `<pango-layout>`
desc	the new `<pango-font-description>`, or '`#f`' to unset the current font description

`pango-layout-set-width` (*self* `<pango-layout*>`) (*width* `int`) [Function]

Sets the width to which the lines of the `<pango-layout>` should wrap.

layout	a `<pango-layout>`.
width	the desired width in Pango units, or -1 to indicate that no wrapping should be performed.

`pango-layout-get-width` (*self* `<pango-layout*>`) ⇒ (*ret* `int`) [Function]

Gets the width to which the lines of the `<pango-layout>` should wrap.

layout	a `<pango-layout>`
ret	the width, or -1 if no width set.

`pango-layout-set-wrap` (*self* `<pango-layout*>`) (*wrap* `<pango-wrap-mode>`) [Function]

Sets the wrap mode; the wrap mode only has effect if a width is set on the layout with `pango-layout-set-width`. To turn off wrapping, set the width to -1.

layout	a `<pango-layout>`
wrap	the wrap mode

`pango-layout-get-wrap` (*self* `<pango-layout*>`) ⇒ (*ret* `<pango-wrap-mode>`) [Function]

Gets the wrap mode for the layout.

layout	a `<pango-layout>`
ret	active wrap mode.

`pango-layout-set-ellipsize` (*self* `<pango-layout*>`) (*ellipsize* `<pango-ellipsize-mode>`) [Function]

Sets the type of ellipsization being performed for *layout*. Depending on the ellipsization mode *ellipsize* text is removed from the start, middle, or end of lines so they fit within the width of layout set with `pango-layout-set-width`.

If the layout contains characters such as newlines that force it to be layed out in multiple lines, then each line is ellipsized separately.

layout	a `<pango-layout>`
ellipsize	the new ellipsization mode for *layout*

Since 1.6

`pango-layout-get-ellipsize` (*self* `<pango-layout*>`) ⇒ (*ret* `<pango-ellipsize-mode>`) [Function]

Gets the type of ellipsization being performed for *layout*. See `pango-layout-set-ellipsize`

layout	a `<pango-layout>`
ret	the current ellipsization mode for *layout*.

Since 1.6

`pango-layout-set-indent` (*self* `<pango-layout*>`) (*indent* `int`) [Function]

Sets the width in Pango units to indent each paragraph. A negative value of *indent* will produce a hanging indentation. That is, the first line will have the full width, and subsequent lines will be indented by the absolute value of *indent*.

layout	a `<pango-layout>`.
indent	the amount by which to indent.

`pango-layout-get-indent` (*self* `<pango-layout*>`) ⇒ (*ret* `int`) [Function]

Gets the paragraph indent width in Pango units. A negative value indicates a hanging indentation.

layout	a `<pango-layout>`
ret	the indent.

`pango-layout-get-spacing` (*self* `<pango-layout*>`) ⇒ (*ret* `int`) [Function]

Gets the amount of spacing in `<pango-glyph-unit>` between the lines of the layout.

layout	a `<pango-layout>`
ret	the spacing.

`pango-layout-set-spacing` (*self* `<pango-layout*>`) (*spacing* `int`) [Function]

Sets the amount of spacing in `<pango-glyph-unit>` between the lines of the layout.

layout	a `<pango-layout>`.
spacing	the amount of spacing

`pango-layout-set-justify` (*self* `<pango-layout*>`) (*justify* `bool`) [Function]

Sets whether each complete line should be stretched to fill the entire width of the layout. This stretching is typically done by adding whitespace, but for some scripts (such as Arabic), the justification may be done in more complex ways, like extending the characters.

Note that as of Pango-1.16, this functionality is not yet implemented.

layout	a `<pango-layout>`
justify	whether the lines in the layout should be justified.

`pango-layout-get-justify` (*self* `<pango-layout*>`) ⇒ (*ret* `bool`) [Function]

Gets whether each complete line should be stretched to fill the entire width of the layout.

layout	a `<pango-layout>`
ret	the justify.

`pango-layout-set-auto-dir` (*self* `<pango-layout*>`) (*auto_dir* `bool`) [Function]

Sets whether to calculate the bidirectional base direction for the layout according to the contents of the layout; when this flag is on (the default), then paragraphs in *layout* that begin with strong right-to-left characters (Arabic and Hebrew principally), will have right-to-left layout, paragraphs with letters from other scripts will have left-to-right layout. Paragraphs with only neutral characters get their direction from the surrounding paragraphs.

When '`#f`', the choice between left-to-right and right-to-left layout is done according to the base direction of the layout's `<pango-context>`. (See `pango-context-set-base-dir`).

When the auto-computed direction of a paragraph differs from the base direction of the context, the interpretation of '`PANGO_ALIGN_LEFT`' and '`PANGO_ALIGN_RIGHT`' are swapped.

layout	a `<pango-layout>`
auto-dir	if '`#t`', compute the bidirectional base direction from the layout's contents.

Since 1.4

`pango-layout-get-auto-dir` (*self* `<pango-layout*>`) ⇒ (*ret* `bool`) [Function]

Gets whether to calculate the bidirectional base direction for the layout according to the contents of the layout. See `pango-layout-set-auto-dir`.

layout	a `<pango-layout>`
ret	'`#t`' if the bidirectional base direction is computed from the layout's contents, '`#f`' otherwise.

Since 1.4

`pango-layout-set-alignment` (*self* `<pango-layout*>`) (*alignment* `<pango-alignment>`) [Function]

Sets the alignment for the layout: how partial lines are positioned within the horizontal space available.

layout	a `<pango-layout>`
alignment	the alignment

`pango-layout-get-alignment` (*self* `<pango-layout*>`) ⇒ (*ret* `<pango-alignment>`) [Function]

Gets the alignment for the layout: how partial lines are positioned within the horizontal space available.

layout a `<pango-layout>`

ret the alignment.

`pango-layout-set-tabs` (*self* `<pango-layout*>`) (*tabs* `<pango-tab-array>`) [Function]

Sets the tabs to use for *layout*, overriding the default tabs (by default, tabs are every 8 spaces). If *tabs* is '`#f`', the default tabs are reinstated. *tabs* is copied into the layout; you must free your copy of *tabs* yourself.

layout a `<pango-layout>`

tabs a `<pango-tab-array>`

`pango-layout-get-tabs` (*self* `<pango-layout*>`) ⇒ (*ret* `<pango-tab-array>`) [Function]

Gets the current `<pango-tab-array>` used by this layout. If no `<pango-tab-array>` has been set, then the default tabs are in use and '`#f`' is returned. Default tabs are every 8 spaces. The return value should be freed with `pango-tab-array-free`.

layout a `<pango-layout>`

ret a copy of the tabs for this layout, or '`#f`'.

`pango-layout-get-log-attrs` (*self* `<pango-layout*>`) (*attrs* `<pango-log-attr**>`) ⇒ (*n_attrs* `int`) [Function]

Retrieves an array of logical attributes for each character in the *layout*.

layout a `<pango-layout>`

attrs location to store a pointer to an array of logical attributes This value must be freed with `g-free`.

n-attrs location to store the number of the attributes in the array. (The stored value will be one more than the total number of characters in the layout, since there need to be attributes corresponding to both the position before the first character and the position after the last character.)

`pango-layout-index-to-pos` (*self* `<pango-layout*>`) (*index_* `int`) (*pos* `<pango-rectangle*>`) [Function]

Converts from an index within a `<pango-layout>` to the onscreen position corresponding to the grapheme at that index, which is represented as rectangle. Note that '`pos->x`' is always the leading edge of the grapheme and '`pos->x + pos->width`' the trailing edge of the grapheme. If the directionality of the grapheme is right-to-left, then '`pos->width`' will be negative.

layout a `<pango-layout>`

index byte index within *layout*

pos rectangle in which to store the position of the grapheme

`pango-layout-index-to-line-x` (*self* `<pango-layout*>`) (*index_* `int`) (*trailing* `bool`) ⇒ (*line* `int`) (*x_pos* `int`) [Function]

Converts from byte *index* within the *layout* to line and X position. (X position is measured from the left edge of the line)

layout	a `<pango-layout>`
index	the byte index of a grapheme within the layout.
trailing	an integer indicating the edge of the grapheme to retrieve the position of. If 0, the trailing edge of the grapheme, if `>` 0, the leading of the grapheme.
line	location to store resulting line index. (which will between 0 and pango_layout_get_line_count(layout) - 1)
x-pos	location to store resulting position within line (‘`PANGO_SCALE`’ units per device unit)

`pango-layout-xy-to-index` (*self* `<pango-layout*>`) (*x* `int`) (*y* `int`) [Function]
⇒ (*ret* `bool`) (*index_* `int`) (*trailing* `int`)

Converts from X and Y position within a layout to the byte index to the character at that logical position. If the Y position is not inside the layout, the closest position is chosen (the position will be clamped inside the layout). If the X position is not within the layout, then the start or the end of the line is chosen as described for `pango-layout-x-to-index`. If either the X or Y positions were not inside the layout, then the function returns ‘`#f`’; on an exact hit, it returns ‘`#t`’.

layout	a `<pango-layout>`
x	the X offset (in `<pango-glyph-unit>`) from the left edge of the layout.
y	the Y offset (in `<pango-glyph-unit>`) from the top edge of the layout
index	location to store calculated byte index
trailing	location to store a integer indicating where in the grapheme the user clicked. It will either be zero, or the number of characters in the grapheme. 0 represents the trailing edge of the grapheme.
ret	‘`#t`’ if the coordinates were inside text, ‘`#f`’ otherwise.

`pango-layout-get-cursor-pos` (*self* `<pango-layout*>`) (*index_* `int`) [Function]
(*strong_pos* `<pango-rectangle*>`) (*weak_pos* `<pango-rectangle*>`)

Given an index within a layout, determines the positions that of the strong and weak cursors if the insertion point is at that index. The position of each cursor is stored as a zero-width rectangle. The strong cursor location is the location where characters of the directionality equal to the base direction of the layout are inserted. The weak cursor location is the location where characters of the directionality opposite to the base direction of the layout are inserted.

layout	a `<pango-layout>`
index	the byte index of the cursor
strong-pos	location to store the strong cursor position (may be ‘`#f`’)
weak-pos	location to store the weak cursor position (may be ‘`#f`’)

`pango-layout-move-cursor-visually` (*self* `<pango-layout*>`) (*strong* `bool`) (*old_index* `int`) (*old_trailing* `int`) (*direction* `int`) ⇒ (*new_index* `int`) (*new_trailing* `int`) [Function]

Computes a new cursor position from an old position and a count of positions to move visually. If *count* is positive, then the new strong cursor position will be one position to the right of the old cursor position. If *count* is negative, then the new strong cursor position will be one position to the left of the old cursor position.

In the presence of bidirection text, the correspondence between logical and visual order will depend on the direction of the current run, and there may be jumps when the cursor is moved off of the end of a run.

Motion here is in cursor positions, not in characters, so a single call to `pango-layout-move-cursor-visually` may move the cursor over multiple characters when multiple characters combine to form a single grapheme.

layout	a `<pango-layout>`.
strong	whether the moving cursor is the strong cursor or the weak cursor. The strong cursor is the cursor corresponding to text insertion in the base direction for the layout.
old-index	the byte index of the grapheme for the old index
old-trailing	if 0, the cursor was at the trailing edge of the grapheme indicated by *old-index*, if `>` 0, the cursor was at the leading edge.
direction	direction to move cursor. A negative value indicates motion to the left.
new-index	location to store the new cursor byte index. A value of -1 indicates that the cursor has been moved off the beginning of the layout. A value of '`G_MAXINT`' indicates that the cursor has been moved off the end of the layout.
new-trailing	number of characters to move forward from the location returned for *new-index* to get the position where the cursor should be displayed. This allows distinguishing the position at the beginning of one line from the position at the end of the preceding line. *new-index* is always on the line where the cursor should be displayed.

`pango-layout-get-extents` (*self* `<pango-layout*>`) (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) [Function]

Computes the logical and ink extents of *layout*. Logical extents are usually what you want for positioning things. Note that both extents may have non-zero x and y. You may want to use those to offset where you render the layout. Not doing that is a very typical bug that shows up as right-to-left layouts not being correctly positioned in a layout with a set width.

The extents are given in layout coordinates and in Pango units; layout coordinates begin at the top left corner of the layout.

layout	a `<pango-layout>`

ink-rect rectangle used to store the extents of the layout as drawn or '`#f`' to indicate that the result is not needed.

logical-rect
rectangle used to store the logical extents of the layout or '`#f`' to indicate that the result is not needed.

`pango-layout-get-pixel-extents` (*self* `<pango-layout*>`) (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) [Function]

Computes the logical and ink extents of *layout* in device units. See `pango-layout-get-extents`; this function just calls `pango-layout-get-extents` and then converts the extents to device units using the '`PANGO_SCALE`' factor.

layout a `<pango-layout>`

ink-rect rectangle used to store the extents of the layout as drawn or '`#f`' to indicate that the result is not needed.

logical-rect
rectangle used to store the logical extents of the layout or '`#f`' to indicate that the result is not needed.

`pango-layout-get-size` (*self* `<pango-layout*>`) ⇒ (*width* `int`) (*height* `int`) [Function]

Determines the logical width and height of a `<pango-layout>` in Pango units. (device units scaled by '`PANGO_SCALE`'). This is simply a convenience function around `pango-layout-get-extents`.

layout a `<pango-layout>`

width location to store the logical width, or '`#f`'

height location to store the logical height, or '`#f`'

`pango-layout-get-pixel-size` (*self* `<pango-layout*>`) ⇒ (*width* `int`) (*height* `int`) [Function]

Determines the logical width and height of a `<pango-layout>` in device units. (`pango-layout-get-size` returns the width and height scaled by '`PANGO_SCALE`'.) This is simply a convenience function around `pango-layout-get-pixel-extents`.

layout a `<pango-layout>`

width location to store the logical width, or '`#f`'

height location to store the logical height, or '`#f`'

`pango-layout-get-line-count` (*self* `<pango-layout*>`) ⇒ (*ret* `int`) [Function]

Retrieves the count of lines for the *layout*.

layout `<pango-layout>`

ret the line count.

`pango-layout-get-line` (*self* `<pango-layout*>`) (*line* `int`) ⇒ (*ret* `<pango-layout-line*>`) [Function]

Retrieves a particular line from a `<pango-layout>`.

layout a `<pango-layout>`

line the index of a line, which must be between 0 and '`pango_layout_get_line_count(layout) - 1`', inclusive.

ret the requested `<pango-layout-line>`, or '`#f`' if the index is out of range. This layout line can be ref'ed and retained, but will become invalid if changes are made to the `<pango-layout>`.

`pango-layout-get-lines` (*self* `<pango-layout*>`) [Function]
⇒ (*ret* `gslist-of`)

Returns the lines of the *layout* as a list.

layout a `<pango-layout>`

ret a `<gs-list>` containing the lines in the layout. This points to internal data of the `<pango-layout>` and must be used with care. It will become invalid on any change to the layout's text or properties.

`pango-layout-get-iter` (*self* `<pango-layout*>`) [Function]
⇒ (*ret* `<pango-layout-iter*>`)

Returns an iterator to iterate over the visual extents of the layout.

layout a `<pango-layout>`

ret the new `<pango-layout-iter>` that should be freed using `pango-layout-iter-free`.

`pango-layout-iter-next-run` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `bool`)

Moves *iter* forward to the next run in visual order. If *iter* was already at the end of the layout, returns '`#f`'.

iter a `<pango-layout-iter>`

ret whether motion was possible.

`pango-layout-iter-next-char` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `bool`)

Moves *iter* forward to the next character in visual order. If *iter* was already at the end of the layout, returns '`#f`'.

iter a `<pango-layout-iter>`

ret whether motion was possible.

`pango-layout-iter-next-cluster` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `bool`)

Moves *iter* forward to the next cluster in visual order. If *iter* was already at the end of the layout, returns '`#f`'.

iter a `<pango-layout-iter>`

ret whether motion was possible.

`pango-layout-iter-next-line` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `bool`)

Moves *iter* forward to the start of the next line. If *iter* is already on the last line, returns '`#f`'.

iter	a `<pango-layout-iter>`
ret	whether motion was possible.

`pango-layout-iter-at-last-line` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `bool`)

Determines whether *iter* is on the last line of the layout.

iter	a `<pango-layout-iter>`
ret	'`#t`' if *iter* is on the last line.

`pango-layout-iter-get-index` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `int`)

Gets the current byte index. Note that iterating forward by char moves in visual order, not logical order, so indexes may not be sequential. Also, the index may be equal to the length of the text in the layout, if on the '`#f`' run (see `pango-layout-iter-get-run`).

iter	a `<pango-layout-iter>`
ret	current byte index.

`pango-layout-iter-get-baseline` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `int`)

Gets the Y position of the current line's baseline, in layout coordinates (origin at top left of the entire layout).

iter	a `<pango-layout-iter>`
ret	baseline of current line.

`pango-layout-iter-get-run` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `<pango-layout-run*>`)

Gets the current run. When iterating by run, at the end of each line, there's a position with a '`#f`' run, so this function can return '`#f`'. The '`#f`' run at the end of each line ensures that all lines have at least one run, even lines consisting of only a newline.

iter	a `<pango-layout-iter>`
ret	the current run.

`pango-layout-iter-get-line` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*ret* `<pango-layout-line*>`)

Gets the current line.

iter	a `<pango-layout-iter>`
ret	the current line.

`pango-layout-iter-get-char-extents` [Function]
(*self* `<pango-layout-iter*>`) (*logical_rect* `<pango-rectangle*>`)

Gets the extents of the current character, in layout coordinates (origin is the top left of the entire layout). Only logical extents can sensibly be obtained for characters; ink extents make sense only down to the level of clusters.

iter	a `<pango-layout-iter>`
logical-rect	rectangle to fill with logical extents

`pango-layout-iter-get-run-extents` (*self* `<pango-layout-iter*>`) [Function]
(*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`)

Gets the extents of the current run in layout coordinates (origin is the top left of the entire layout).

iter	a `<pango-layout-iter>`
ink-rect	rectangle to fill with ink extents, or '`#f`'
logical-rect	rectangle to fill with logical extents, or '`#f`'

`pango-layout-iter-get-line-yrange` (*self* `<pango-layout-iter*>`) [Function]
⇒ (*y0_* `int`) (*y1_* `int`)

Divides the vertical space in the `<pango-layout>` being iterated over between the lines in the layout, and returns the space belonging to the current line. A line's range includes the line's logical extents, plus half of the spacing above and below the line, if `pango-layout-set-spacing` has been called to set layout spacing. The Y positions are in layout coordinates (origin at top left of the entire layout).

iter	a `<pango-layout-iter>`
y0	start of line
y1	end of line

`pango-layout-iter-get-line-extents` [Function]
(*self* `<pango-layout-iter*>`) (*ink_rect* `<pango-rectangle*>`)
(*logical_rect* `<pango-rectangle*>`)

Obtains the extents of the current line. *ink-rect* or *logical-rect* can be NULL if you aren't interested in them. Extents are in layout coordinates (origin is the top-left corner of the entire `<pango-layout>`). Thus the extents returned by this function will be the same width/height but not at the same x/y as the extents returned from `pango-layout-line-get-extents`.

iter	a `<pango-layout-iter>`
ink-rect	rectangle to fill with ink extents, or '`#f`'
logical-rect	rectangle to fill with logical extents, or '`#f`'

pango-layout-line-get-extents (*self* `<pango-layout-line*>`) (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) [Function]

Computes the logical and ink extents of a layout line. See `pango-font-get-glyph-extents` for details about the interpretation of the rectangles.

line a `<pango-layout-line>`

ink-rect rectangle used to store the extents of the glyph string as drawn, or '`#f`'

logical-rect
rectangle used to store the logical extents of the glyph string, or '`#f`'

pango-layout-line-get-pixel-extents (*self* `<pango-layout-line*>`) (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) [Function]

Computes the logical and ink extents of a layout line. See `pango-font-get-glyph-extents` for details about the interpretation of the rectangles. The returned rectangles are in device units, as opposed to `pango-layout-line-get-extents`, which returns the extents in `<pango-glyph-unit>`.

layout-line
a `<pango-layout-line>`

ink-rect rectangle used to store the extents of the glyph string as drawn, or '`#f`'

logical-rect
rectangle used to store the logical extents of the glyph string, or '`#f`'

pango-layout-line-index-to-x (*self* `<pango-layout-line*>`) (*index_* `int`) (*trailing* `bool`) ⇒ (*x_pos* `int`) [Function]

Converts an index within a line to a X position.

line a `<pango-layout-line>`

index byte offset of a grapheme within the layout

trailing an integer indicating the edge of the grapheme to retrieve the position of. If 0, the trailing edge of the grapheme, if > 0, the leading of the grapheme.

x-pos location to store the x_offset (in `<pango-glyph-unit>`)

pango-layout-line-x-to-index (*self* `<pango-layout-line*>`) (*x_pos* `int`) ⇒ (*ret* `bool`) (*index_* `int`) (*trailing* `int`) [Function]

Converts from x offset to the byte index of the corresponding character within the text of the layout. If *x-pos* is outside the line, *index* and *trailing* will point to the very first or very last position in the line. This determination is based on the resolved direction of the paragraph; for example, if the resolved direction is right-to-left, then an X position to the right of the line (after it) results in 0 being stored in *index* and *trailing*. An X position to the left of the line results in *index* pointing to the (logical) last grapheme in the line and *trailing* being set to the number of characters in that grapheme. The reverse is true for a left-to-right line.

line a `<pango-layout-line>`

x-pos the X offset (in `<pango-glyph-unit>`) from the left edge of the line.

index location to store calculated byte index for the grapheme in which the user clicked.

trailing location to store a integer indicating where in the grapheme the user clicked. It will either be zero, or the number of characters in the grapheme. 0 represents the trailing edge of the grapheme.

ret '`#f`' if *x-pos* was outside the line, '`#t`' if inside

`pango-layout-line-get-x-ranges` (*self* `<pango-layout-line*>`) (*start_index* `int`) (*end_index* `int`) (*ranges* `<int**>`) ⇒ (*n_ranges* `int`) [Function]

Gets a list of visual ranges corresponding to a given logical range. This list is not necessarily minimal - there may be consecutive ranges which are adjacent. The ranges will be sorted from left to right. The ranges are with respect to the left edge of the entire layout, not with respect to the line.

line a `<pango-layout-line>`

start-index
Start byte index of the logical range. If this value is less than the start index for the line, then the first range will extend all the way to the leading edge of the layout. Otherwise it will start at the leading edge of the first character.

end-index Ending byte index of the logical range. If this value is greater than the end index for the line, then the last range will extend all the way to the trailing edge of the layout. Otherwise, it will end at the trailing edge of the last character.

ranges location to store a pointer to an array of ranges. The array will be of length '`2*n_ranges`', with each range starting at '`(*ranges)[2*n]`' and of width '`(*ranges)[2*n + 1] - (*ranges)[2*n]`'. This array must be freed with `g-free`. The coordinates are relative to the layout and are in `<pango-glyph-unit>`.

n-ranges The number of ranges stored in *ranges*.

6 Rendering

Functions to run the rendering pipeline

6.1 Overview

The Pango rendering pipeline takes a string of Unicode characters and converts it into glyphs. The functions described in this section accomplish various steps of this process.

6.2 Usage

`pango-itemize` (*context* `<pango-context*>`) (*text* `mchars`) (*start_index* `int`) (*length* `int`) (*attrs* `<pango-attr-list>`) (*cached_iter* `<pango-attr-iterator*>`) ⇒ (*ret* `glist-of`) [Function]

Breaks a piece of text into segments with consistent directional level and shaping engine. Each byte of *text* will be contained in exactly one of the items in the returned list; the generated list of items will be in logical order (the start offsets of the items are ascending).

cached-iter should be an iterator over *attrs* currently positioned at a range before or containing *start-index*; *cached-iter* will be advanced to the range covering the position just after *start-index* + *length*. (i.e. if itemizing in a loop, just keep passing in the same *cached-iter*).

context	a structure holding information that affects the itemization process.
text	the text to itemize.
start-index	first byte in *text* to process
length	the number of bytes (not characters) to process after *start-index*. This must be >= 0.
attrs	the set of attributes that apply to *text*.
cached-iter	Cached attribute iterator, or '`#f`'
ret	a `<g-list>` of `<pango-item>` structures.

`pango-itemize-with-base-dir` (*context* `<pango-context*>`) (*base_dir* `<pango-direction>`) (*text* `mchars`) (*start_index* `int`) (*length* `int`) (*attrs* `<pango-attr-list>`) (*cached_iter* `<pango-attr-iterator*>`) ⇒ (*ret* `glist-of`) [Function]

Like `pango-itemize`, but the base direction to use when computing bidirectional levels (see `pango-context-set-base-dir`), is specified explicitly rather than gotten from the `<pango-context>`.

context	a structure holding information that affects the itemization process.
base-dir	base direction to use for bidirectional processing
text	the text to itemize.

start-index
: first byte in *text* to process

length
: the number of bytes (not characters) to process after *start-index*. This must be >= 0.

attrs
: the set of attributes that apply to *text*.

cached-iter
: Cached attribute iterator, or '`#f`'

ret
: a `<g-list>` of `<pango-item>` structures.

Since 1.4

`pango-item-new` ⇒ (*ret* `<pango-item*>`) [Function]

Creates a new `<pango-item>` structure initialized to default values.

ret
: the newly allocated `<pango-item>`, which should be freed with `pango-item-free`.

`pango-item-split` (*self* `<pango-item*>`) (*split_index* `int`) (*split_offset* `int`) ⇒ (*ret* `<pango-item*>`) [Function]

Modifies *orig* to cover only the text after *split-index*, and returns a new item that covers the text before *split-index* that used to be in *orig*. You can think of *split-index* as the length of the returned item. *split-index* may not be 0, and it may not be greater than or equal to the length of *orig* (that is, there must be at least one byte assigned to each item, you can't create a zero-length item). *split-offset* is the length of the first item in chars, and must be provided because the text used to generate the item isn't available, so `pango-item-split` can't count the char length of the split items itself.

orig
: a `<pango-item>`

split-index
: byte index of position to split item, relative to the start of the item

split-offset
: number of chars between start of *orig* and *split-index*

ret
: new item representing text before *split-index*, which should be freed with `pango-item-free`.

`pango-reorder-items` (*logical_items* `glist-of`) ⇒ (*ret* `glist-of`) [Function]

From a list of items in logical order and the associated directional levels, produce a list in visual order. The original list is unmodified.

logical-items
: a `<g-list>` of `<pango-item>` in logical order.

ret
: a `<g-list>` of `<pango-item>` structures in visual order. (Please open a bug if you use this function. It is not a particularly convenient interface, and the code is duplicated elsewhere in Pango for that reason.)

`pango-context-get-font-map` (*self* `<pango-context*>`) [Function]
⇒ (*ret* `<pango-font-map>`)

Gets the `<pango-fontmap>` used to look up fonts for this context.

context	a `<pango-context>`
ret	the font map for the `<pango-context>`. This value is owned by Pango and should not be unreferenced.

Since 1.6

`pango-context-set-font-description` (*self* `<pango-context*>`) [Function]
(*desc* `<pango-font-description>`)

Set the default font description for the context

context	a `<pango-context>`
desc	the new pango font description

`pango-context-get-language` (*self* `<pango-context*>`) [Function]
⇒ (*ret* `<pango-language>`)

Retrieves the global language tag for the context.

context	a `<pango-context>`
ret	the global language tag.

`pango-context-set-language` (*self* `<pango-context*>`) [Function]
(*language* `<pango-language>`)

Sets the global language tag for the context.

context	a `<pango-context>`
language	the new language tag.

`pango-context-get-base-dir` (*self* `<pango-context*>`) [Function]
⇒ (*ret* `<pango-direction>`)

Retrieves the base direction for the context. See `pango-context-set-base-dir`.

context	a `<pango-context>`
ret	the base direction for the context.

`pango-context-set-base-dir` (*self* `<pango-context*>`) [Function]
(*direction* `<pango-direction>`)

Sets the base direction for the context.

The base direction is used in applying the Unicode bidirectional algorithm; if the *direction* is ‘`PANGO_DIRECTION_LTR`’ or ‘`PANGO_DIRECTION_RTL`’, then the value will be used as the paragraph direction in the Unicode bidirectional algorithm. A value of ‘`PANGO_DIRECTION_WEAK_LTR`’ or ‘`PANGO_DIRECTION_WEAK_RTL`’ is used only for paragraphs that do not contain any strong characters themselves.

context	a `<pango-context>`
direction	the new base direction

`pango-context-get-matrix` (*self* `<pango-context*>`) [Function]
⇒ (*ret* `<pango-matrix*>`)

Gets the transformation matrix that will be applied when rendering with this context. See `pango-context-set-matrix`.

context	a `<pango-context>`
ret	the matrix, or '`#f`' if no matrix has been set (which is the same as the identity matrix). The returned matrix is owned by Pango and must not be modified or freed.

Since 1.6

`pango-context-set-matrix` (*self* `<pango-context*>`) [Function]
(*matrix* `<pango-matrix*>`)

Sets the transformation matrix that will be applied when rendering with this context. Note that reported metrics are in the user space coordinates before the application of the matrix, not device-space coordinates after the application of the matrix. So, they don't scale with the matrix, though they may change slightly for different matrices, depending on how the text is fit to the pixel grid.

context	a `<pango-context>`
matrix	a `<pango-matrix>`, or '`#f`' to unset any existing matrix. (No matrix set is the same as setting the identity matrix.)

Since 1.6

`pango-context-load-font` (*self* `<pango-context*>`) [Function]
(*desc* `<pango-font-description>`) ⇒ (*ret* `<pango-font>`)

Loads the font in one of the fontmaps in the context that is the closest match for *desc*.

context	a `<pango-context>`
desc	a `<pango-font-description>` describing the font to load
ret	the font loaded, or '`#f`' if no font matched.

`pango-context-load-fontset` (*self* `<pango-context*>`) [Function]
(*desc* `<pango-font-description>`) (*language* `<pango-language>`)
⇒ (*ret* `<pango-fontset*>`)

Load a set of fonts in the context that can be used to render a font matching *desc*.

context	a `<pango-context>`
desc	a `<pango-font-description>` describing the fonts to load
language	a `<pango-language>` the fonts will be used for
ret	the fontset, or '`#f`' if no font matched.

`pango-context-get-metrics` (*self* `<pango-context*>`) [Function]
(*desc* `<pango-font-description>`) (*language* `<pango-language>`)
⇒ (*ret* `<pango-font-metrics>`)

Get overall metric information for a particular font description. Since the metrics may be substantially different for different scripts, a language tag can be provided to indicate that the metrics should be retrieved that correspond to the script(s) used by that language.

The `<pango-font-description>` is interpreted in the same way as by `pango-itemize`, and the family name may be a comma separated list of figures. If characters from multiple of these families would be used to render the string, then the returned fonts would be a composite of the metrics for the fonts loaded for the individual families.

context a `<pango-context>`

desc a `<pango-font-description>` structure

language language tag used to determine which script to get the metrics for. '`#f`' means that the language tag from the context will be used. If no language tag is set on the context, metrics large enough to cover a range of languages will be returned. The process of determining such metrics is slow, so it is best to always make sure some real language tag will be used.

ret a `<pango-font-metrics>` object. The caller must call `pango-font-metrics-unref` when finished using the object.

`pango-context-list-families` (*self* `<pango-context*>`) [Function]
(*families* `<pango-font-family***>`) ⇒ (*n_families* `int`)

List all families for a context.

context a `<pango-context>`

families location to store a pointer to an array of `<pango-font-family>` *. This array should be freed with `g-free`.

n-families location to store the number of elements in *descs*

`pango-get-mirror-char` (*ch* `unsigned-int32`) ⇒ (*ret* `bool`) [Function]
(*mirrored_ch* `unsigned-int32`)

'`pango_get_mirror_char`' is deprecated and should not be used in newly-written code.

If *ch* has the Unicode mirrored property and there is another Unicode character that typically has a glyph that is the mirror image of *ch*'s glyph, puts that character in the address pointed to by *mirrored-ch*.

Use `g-unichar-get-mirror-char` instead; the docs for that function provide full details.

ch a Unicode character

mirrored-ch
location to store the mirrored character

ret '`#t`' if *ch* has a mirrored character and *mirrored-ch* is filled in, '`#f`' otherwise

`pango-unichar-direction` (*ch* `unsigned-int32`) [Function]
⇒ (*ret* `<pango-direction>`)

Determines the direction of a character; either '`PANGO_DIRECTION_LTR`', '`PANGO_DIRECTION_RTL`', or '`PANGO_DIRECTION_NEUTRAL`'.

ch	a Unicode character
ret	the direction of the character, as used in the Unicode bidirectional algorithm.

`pango-find-base-dir` (*text* `mchars`) (*length* `int`) [Function]
⇒ (*ret* `<pango-direction>`)

Searches a string the first character that has a strong direction, according to the Unicode bidirectional algorithm.

text	the text to process
length	length of *text* in bytes (may be -1 if *text* is nul-terminated)
ret	The direction corresponding to the first strong character. If no such character is found, then '`PANGO_DIRECTION_NEUTRAL`' is returned.

Since 1.4

`pango-break` (*text* `mchars`) (*length* `int`) [Function]
(*analysis* `<pango-analysis*>`) (*attrs* `<pango-log-attr*>`) (*attrs_len* `int`)

Determines possible line, word, and character breaks for a string of Unicode text with a single analysis. For most purposes you may want to use `pango-get-log-attrs`.

text	the text to process
length	length of *text* in bytes (may be -1 if *text* is nul-terminated)
analysis	`<pango-analysis>` structure from `pango-itemize`
attrs	an array to store character information in
attrs-len	size of the array passed as *attrs*

`pango-get-log-attrs` (*text* `mchars`) (*length* `int`) (*level* `int`) [Function]
(*language* `<pango-language>`) (*log_attrs* `<pango-log-attr*>`)
(*attrs_len* `int`)

Computes a `<pango-log-attr>` for each character in *text*. The *log-attrs* array must have one `<pango-log-attr>` for each position in *text*; if *text* contains N characters, it has N+1 positions, including the last position at the end of the text. *text* should be an entire paragraph; logical attributes can't be computed without context (for example you need to see spaces on either side of a word to know the word is a word).

text	text to process
length	length in bytes of *text*
level	embedding level, or -1 if unknown
language	language tag

log-attrs array with one `<pango-log-attr>` per character in *text*, plus one extra, to be filled in

attrs-len length of *log-attrs* array

`pango-find-paragraph-boundary` (*text* `mchars`) (*length* `int`) [Function]
⇒ (*paragraph_delimiter_index* `int`) (*next_paragraph_start* `int`)

Locates a paragraph boundary in *text*. A boundary is caused by delimiter characters, such as a newline, carriage return, carriage return-newline pair, or Unicode paragraph separator character. The index of the run of delimiters is returned in *paragraph-delimiter-index*. The index of the start of the paragraph (index after all delimiters) is stored in *next-paragraph-start*.

If no delimiters are found, both *paragraph-delimiter-index* and *next-paragraph-start* are filled with the length of *text* (an index one off the end).

text UTF-8 text

length length of *text* in bytes, or -1 if nul-terminated

paragraph-delimiter-index
return location for index of delimiter

next-paragraph-start
return location for start of next paragraph

`pango-shape` (*text* `mchars`) (*length* `int`) [Function]
(*analysis* `<pango-analysis*>`) (*glyphs* `<pango-glyph-string>`)

Given a segment of text and the corresponding `<pango-analysis>` structure returned from `pango-itemize`, convert the characters into glyphs. You may also pass in only a substring of the item from `pango-itemize`.

text the text to process

length the length (in bytes) of *text*

analysis `<pango-analysis>` structure from `pango-itemize`

glyphs glyph string in which to store results

7 PangoRenderer

Rendering driver base class

7.1 Overview

`<pango-renderer>` is a base class that contains the necessary logic for rendering a `<pango-layout>` or `<pango-layout-line>`. By subclassing `<pango-renderer>` and overriding operations such as *draw-glyphs* and *draw-rectangle*, renderers for particular font backends and destinations can be created.

7.2 Usage

`<pango-renderer>` [Class]

This `<gobject>` class defines no properties, other than those defined by its superclasses.

`pango-renderer-draw-layout` (*self* `<pango-renderer>`) (*layout* `<pango-layout*>`) (*x* `int`) (*y* `int`) [Function]

`draw-layout` [Method]

Draws *layout* with the specified `<pango-renderer>`.

renderer	a `<pango-renderer>`
layout	a `<pango-layout>`
x	X position of left edge of baseline, in user space coordinates in Pango units.
y	Y position of left edge of baseline, in user space coordinates in Pango units.

Since 1.8

`pango-renderer-draw-layout-line` (*self* `<pango-renderer>`) (*line* `<pango-layout-line*>`) (*x* `int`) (*y* `int`) [Function]

`draw-layout-line` [Method]

Draws *line* with the specified `<pango-renderer>`.

renderer	a `<pango-renderer>`
line	a `<pango-layout-line>`
x	X position of left edge of baseline, in user space coordinates in Pango units.
y	Y position of left edge of baseline, in user space coordinates in Pango units.

Since 1.8

`pango-renderer-draw-glyphs` (*self* `<pango-renderer>`) (*font* `<pango-font>`) (*glyphs* `<pango-glyph-string>`) (*x* `int`) (*y* `int`) [Function]

`draw-glyphs` [Method]

Draws the glyphs in *glyphs* with the specified `<pango-renderer>`.

renderer	a `<pango-renderer>`
font	a `<pango-font>`
glyphs	a `<pango-glyph-string>`
x	X position of left edge of baseline, in user space coordinates in Pango units.
y	Y position of left edge of baseline, in user space coordinates in Pango units.

Since 1.8

`pango-renderer-draw-rectangle` (*self* `<pango-renderer>`) (*part* `<pango-render-part>`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) [Function]

`draw-rectangle` [Method]

Draws an axis-aligned rectangle in user space coordinates with the specified `<pango-renderer>`.

This should be called while *renderer* is already active. Use `pango-renderer-activate` to activate a renderer.

renderer	a `<pango-renderer>`
part	type of object this rectangle is part of
x	X position at which to draw rectangle, in user space coordinates in Pango units
y	Y position at which to draw rectangle, in user space coordinates in Pango units
width	width of rectangle in Pango units in user space coordinates
height	height of rectangle in Pango units in user space coordinates

Since 1.8

`pango-renderer-draw-error-underline` (*self* `<pango-renderer>`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) [Function]

`draw-error-underline` [Method]

Draw a squiggly line that approximately covers the given rectangle in the style of an underline used to indicate a spelling error. (The width of the underline is rounded to an integer number of up/down segments and the resulting rectangle is centered in the original rectangle)

This should be called while *renderer* is already active. Use `pango-renderer-activate` to activate a renderer.

renderer	a `<pango-renderer>`
x	X coordinate of underline, in Pango units in user coordinate system
y	Y coordinate of underline, in Pango units in user coordinate system
width	width of underline, in Pango units in user coordinate system
height	height of underline, in Pango units in user coordinate system

Since 1.8

`pango-renderer-draw-trapezoid` (*self* `<pango-renderer>`) (*part* `<pango-render-part>`) (*y1_* `double`) (*x11* `double`) (*x21* `double`) (*y2* `double`) (*x12* `double`) (*x22* `double`) [Function]
`draw-trapezoid` [Method]

Draws a trapezoid with the parallel sides aligned with the X axis using the given `<pango-renderer>`; coordinates are in device space.

renderer	a `<pango-renderer>`
part	type of object this trapezoid is part of
y1	Y coordinate of top of trapezoid
x11	X coordinate of left end of top of trapezoid
x21	X coordinate of right end of top of trapezoid
y2	Y coordinate of bottom of trapezoid
x12	X coordinate of left end of bottom of trapezoid
x22	X coordinate of right end of bottom of trapezoid

Since 1.8

`pango-renderer-draw-glyph` (*self* `<pango-renderer>`) (*font* `<pango-font>`) (*glyph* `unsigned-int32`) (*x* `double`) (*y* `double`) [Function]
`draw-glyph` [Method]

Draws a single glyph with coordinates in device space.

renderer	a `<pango-renderer>`
font	a `<pango-font>`
glyph	the glyph index of a single glyph
x	X coordinate of left edge of baseline of glyph
y	Y coordinate of left edge of baseline of glyph

Since 1.8

`pango-renderer-activate` (*self* `<pango-renderer>`) [Function]
`activate` [Method]

Does initial setup before rendering operations on *renderer*. `pango-renderer-deactivate` should be called when done drawing. Calls such as `pango-renderer-draw-layout` automatically activate the layout before drawing on it. Calls to `pango-renderer-activate` and `pango-renderer-deactivate` can be nested and the renderer will only be initialized and deinitialized once.

renderer	a `<pango-renderer>`

Since 1.8

`pango-renderer-deactivate` (*self* `<pango-renderer>`) [Function]
`deactivate` [Method]

Cleans up after rendering operations on *renderer*. See docs for `pango-renderer-activate`.

renderer	a `<pango-renderer>`

Since 1.8

`pango-renderer-part-changed` (*self* `<pango-renderer>`) (*part* `<pango-render-part>`) [Function]

`part-changed` [Method]

Informs Pango that the way that the rendering is done for *part* has changed in a way that would prevent multiple pieces being joined together into one drawing call. For instance, if a subclass of `<pango-renderer>` was to add a stipple option for drawing underlines, it needs to call

```
pango_renderer_part_changed (render, PANGO_RENDER_PART_UNDERLINE);
```

When the stipple changes or underlines with different stipples might be joined together. Pango automatically calls this for changes to colors. (See `pango-renderer-set-color`)

renderer	a `<pango-renderer>`
part	the part for which rendering has changed.

Since 1.8

`pango-renderer-set-color` (*self* `<pango-renderer>`) (*part* `<pango-render-part>`) (*color* `<pango-color>`) [Function]

`set-color` [Method]

Sets the color for part of the rendering.

renderer	a `<pango-renderer>`
part	the part to change the color of
color	the new color or '`#f`' to unset the current color

Since 1.8

`pango-renderer-get-color` (*self* `<pango-renderer>`) (*part* `<pango-render-part>`) ⇒ (*ret* `<pango-color>`) [Function]

`get-color` [Method]

Gets the current rendering color for the specified part.

renderer	a `<pango-renderer>`
part	the part to get the color for
ret	the color for the specified part, or '`#f`' if it hasn't been set and should be inherited from the environment.

Since 1.8

`pango-renderer-set-matrix` (*self* `<pango-renderer>`) (*matrix* `<pango-matrix*>`) [Function]

`set-matrix` [Method]

Sets the transformation matrix that will be applied when rendering.

renderer	a `<pango-renderer>`
matrix	a `<pango-matrix>`, or '`#f`' to unset any existing matrix. (No matrix set is the same as setting the identity matrix.)

Since 1.8

`pango-renderer-get-matrix` (*self* `<pango-renderer>`) [Function]
⇒ (*ret* `<pango-matrix*>`)
`get-matrix` [Method]

Gets the transformation matrix that will be applied when rendering. See `pango-renderer-set-matrix`.

renderer	a `<pango-renderer>`
ret	the matrix, or '`#f`' if no matrix has been set (which is the same as the identity matrix). The returned matrix is owned by Pango and must not be modified or freed.

Since 1.8

8 Scripts

Identifying writing systems

8.1 Overview

The functions in this section are used to identify the writing system, or *script* of individual characters and of ranges within a larger text string.

8.2 Usage

`pango-script-for-unichar` (*ch* `unsigned-int32`) [Function]
⇒ (*ret* `<pango-script>`)

Looks up the `<pango-script>` for a particular character (as defined by Unicode Standard Annex `<24>`). No check is made for *ch* being a valid Unicode character; if you pass in invalid character, the result is undefined.

ch	a Unicode character
ret	the `<pango-script>` for the character.

Since 1.4

`pango-script-get-sample-language` (*script* `<pango-script>`) [Function]
⇒ (*ret* `<pango-language>`)

Given a script, finds a language tag that is reasonably representative of that script. This will usually be the most widely spoken or used language written in that script: for instance, the sample language for '`PANGO_SCRIPT_CYRILLIC`' is '`ru`' (Russian), the sample language for '`PANGO_SCRIPT_ARABIC`' is '`ar`'.

For some scripts, no sample language will be returned because there is no language that is sufficiently representative. The best example of this is '`PANGO_SCRIPT_HAN`', where various different variants of written Chinese, Japanese, and Korean all use significantly different sets of Han characters and forms of shared characters. No sample language can be provided for many historical scripts as well.

script	a `<pango-script>`
ret	a `<pango-language>` that is representative of the script, or '`#f`' if no such language exists.

Since 1.4

`pango-language-includes-script` (*self* `<pango-language>`) [Function]
(*script* `<pango-script>`) ⇒ (*ret* `bool`)

Determines if *script* is one of the scripts used to write *language*. The returned value is conservative; if nothing is known about the language tag *language*, '`#t`' will be returned, since, as far as Pango knows, *script* might be used to write *language*.

This routine is used in Pango's itemization process when determining if a supplied language tag is relevant to a particular section of text. It probably is not useful for applications in most circumstances.

language	a `<pango-language>`
script	a `<pango-script>`
ret	'`#t`' if *script* is one of the scripts used to write *language*, or if nothing is known about *language*.

Since 1.4

`pango-script-iter-new` (*text* `mchars`) (*length* `int`) [Function]
⇒ (*ret* `<pango-script-iter*>`)

Create a new `<pango-script-iter>`, used to break a string of Unicode into runs by text. No copy is made of *text*, so the caller needs to make sure it remains valid until the iterator is freed with `pango-script-iter-free`.x

text	a UTF-8 string
length	length of *text*, or -1 if *text* is nul-terminated.
ret	the new script iterator, initialized to point at the first range in the text, which should be freed with `pango-script-iter-free`. If the string is empty, it will point at an empty range.

Since 1.4

`pango-script-iter-get-range` (*self* `<pango-script-iter*>`) [Function]
(*start* `<char**>`) (*end* `<char**>`) (*script* `<pango-script*>`)

Gets information about the range to which *iter* currently points. The range is the set of locations p where *start <= p < *end. (That is, it doesn't include the character stored at *end)

iter	a `<pango-script-iter>`
start	location to store start position of the range, or '`#f`'
end	location to store end position of the range, or '`#f`'
script	location to store script for range, or '`#f`'

Since 1.4

`pango-script-iter-next` (*self* `<pango-script-iter*>`) [Function]
⇒ (*ret* `bool`)

Advances a `<pango-script-iter>` to the next range. If *iter* is already at the end, it is left unchanged and '`#f`' is returned.

iter	a `<pango-script-iter>`
ret	'`#t`' if *iter* was successfully advanced.

Since 1.4

9 Tab Stops

Structures for storing tab stops

9.1 Overview

Functions in this section are used to deal with `<pango-tab-array>` objects that can be used to set tab stop positions in a `<pango-layout>`.

9.2 Usage

`<pango-tab-array>` [Class]

`pango-tab-array-new` (*initial_size* `int`) (*positions_in_pixels* `bool`) [Function]
⇒ (*ret* `<pango-tab-array>`)

Creates an array of *initial-size* tab stops. Tab stops are specified in pixel units if *positions-in-pixels* is '`#t`', otherwise in Pango units. All stops are initially at position 0.

initial-size	Initial number of tab stops to allocate, can be 0
positions-in-pixels	whether positions are in pixel units
ret	the newly allocated `<pango-tab-array>`, which should be freed with `pango-tab-array-free`.

`pango-tab-array-get-size` (*self* `<pango-tab-array>`) ⇒ (*ret* `int`) [Function]

Gets the number of tab stops in *tab-array*.

tab-array	a `<pango-tab-array>`
ret	the number of tab stops in the array.

`pango-tab-array-resize` (*self* `<pango-tab-array>`) (*new_size* `int`) [Function]

Resizes a tab array. You must subsequently initialize any tabs that were added as a result of growing the array.

tab-array	a `<pango-tab-array>`
new-size	new size of the array

`pango-tab-array-set-tab` (*self* `<pango-tab-array>`) (*tab_index* `int`) [Function]
(*alignment* `<pango-tab-align>`) (*location* `int`)

Sets the alignment and location of a tab stop. *alignment* must always be `<pango-tab-left>` in the current implementation.

tab-array	a `<pango-tab-array>`
tab-index	the index of a tab stop
alignment	tab alignment
location	tab location in Pango units

`pango-tab-array-get-tab` (*self* `<pango-tab-array>`) (*tab_index* `int`) (*alignment* `<pango-tab-align*>`) ⇒ (*location* `int`) [Function]

Gets the alignment and position of a tab stop.

tab-array	a `<pango-tab-array>`
tab-index	tab stop index
alignment	location to store alignment, or '`#f`'
location	location to store tab position, or '`#f`'

`pango-tab-array-get-tabs` (*self* `<pango-tab-array>`) (*alignments* `<pango-tab-align**>`) (*locations* `<gint**>`) [Function]

If non-'`#f`', *alignments* and *locations* are filled with allocated arrays of length `pango-tab-array-get-size`. You must free the returned array.

tab-array	a `<pango-tab-array>`
alignments	location to store an array of tab stop alignments, or '`#f`'
locations	location to store an array of tab positions, or '`#f`'

10 Text Attributes

Font and other attributes for annotating text

10.1 Overview

Attributed text is used in a number of places in Pango. It is used as the input to the itemization process and also when creating a `<pango-layout>`. The data types and functions in this section are used to represent and manipulate sets of attributes applied to a portion of text.

10.2 Usage

`<pango-color>` [Class]

`<pango-language>` [Class]

`<pango-attr-list>` [Class]

`pango-parse-markup` (*markup_text* `mchars`) (*length* `int`) (*accel_marker* `unsigned-int32`) (*attr_list* `<pango-attr-list**>`) (*text* `<char**>`) ⇒ (*ret* `bool`) (*accel_char* `unsigned-int32`) [Function]

Parses marked-up text (see markup format) to create a plain-text string and an attribute list.

If *accel-marker* is nonzero, the given character will mark the character following it as an accelerator. For example, *accel-marker* might be an ampersand or underscore. All characters marked as an accelerator will receive a '`PANGO_UNDERLINE_LOW`' attribute, and the first character so marked will be returned in *accel-char*. Two *accel-marker* characters following each other produce a single literal *accel-marker* character.

markup-text	markup to parse (see markup format)
length	length of *markup-text*, or -1 if nul-terminated
accel-marker	character that precedes an accelerator, or 0 for none
attr-list	address of return location for a `<pango-attr-list>`, or '`#f`'
text	address of return location for text with tags stripped, or '`#f`'
accel-char	address of return location for accelerator char, or '`#f`'
error	address of return location for errors, or '`#f`'
ret	'`#f`' if *error* is set, otherwise '`#t`'

`pango-attr-type-register` (*name* `mchars`) ⇒ (*ret* `<pango-attr-type>`) [Function]

Allocate a new attribute type ID.

name	an identifier for the type (currently unused.)
ret	the new type ID.

`pango-attribute-equal` (*self* `<pango-attribute*>`) (*attr2* `<pango-attribute*>`) ⇒ (*ret* `bool`) [Function]

Compare two attributes for equality. This compares only the actual value of the two attributes and not the ranges that the attributes apply to.

attr1	a `<pango-attribute>`
attr2	another `<pango-attribute>`
ret	'`#t`' if the two attributes have the same value.

`pango-attribute-destroy` (*self* `<pango-attribute*>`) [Function]

Destroy a `<pango-attribute>` and free all associated memory.

attr	a `<pango-attribute>`.

`pango-attr-language-new` (*language* `<pango-language>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new language tag attribute.

language	language tag
ret	the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-family-new` (*family* `mchars`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font family attribute.

family	the family or comma separated list of families
ret	the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-style-new` (*style* `<pango-style>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font slant style attribute.

style	the slant style
ret	the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-variant-new` (*variant* `<pango-variant>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font variant attribute (normal or small caps)

variant	the variant
ret	the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-stretch-new` (*stretch* `<pango-stretch>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font stretch attribute

stretch the stretch

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-weight-new` (*weight* `<pango-weight>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font weight attribute.

weight the weight

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-size-new` (*size* `int`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font-size attribute in fractional points.

size the font size, in '`PANGO_SCALE`'ths of a point.

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-size-new-absolute` (*size* `int`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font-size attribute in device units.

size the font size, in '`PANGO_SCALE`'ths of a device unit.

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

Since 1.8

`pango-attr-font-desc-new` (*desc* `<pango-font-description>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font description attribute. This attribute allows setting family, style, weight, variant, stretch, and size simultaneously.

desc the font description

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-foreground-new` (*red* `unsigned-int16`) (*green* `unsigned-int16`) (*blue* `unsigned-int16`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new foreground color attribute.

red the red value (ranging from 0 to 65535)

green the green value

blue the blue value

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-background-new` (*red* `unsigned-int16`) (*green* `unsigned-int16`) (*blue* `unsigned-int16`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new background color attribute.

red the red value (ranging from 0 to 65535)

green the green value

blue the blue value

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-strikethrough-new` (*strikethrough* `bool`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new strike-through attribute.

strikethrough
‘`#t`’ if the text should be struck-through.

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-strikethrough-color-new` (*red* `unsigned-int16`) (*green* `unsigned-int16`) (*blue* `unsigned-int16`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new strikethrough color attribute. This attribute modifies the color of strikethrough lines. If not set, strikethrough lines will use the foreground color.

red the red value (ranging from 0 to 65535)

green the green value

blue the blue value

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

Since 1.8

`pango-attr-underline-new` (*underline* `<pango-underline>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new underline-style attribute.

underline the underline style.

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-underline-color-new` (*red* `unsigned-int16`) (*green* `unsigned-int16`) (*blue* `unsigned-int16`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new underline color attribute. This attribute modifies the color of underlines. If not set, underlines will use the foreground color.

red	the red value (ranging from 0 to 65535)
green	the green value
blue	the blue value
ret	the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

Since 1.8

`pango-attr-shape-new` (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new shape attribute. A shape is used to impose a particular ink and logical rectangle on the result of shaping a particular glyph. This might be used, for instance, for embedding a picture or a widget inside a `<pango-layout>`.

ink-rect	ink rectangle to assign to each character
logical-rect	logical rectangle to assign to each character
ret	the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-shape-new-with-data` (*ink_rect* `<pango-rectangle*>`) (*logical_rect* `<pango-rectangle*>`) (*data* `<gpointer>`) (*copy_func* `<pango-attr-data-copy-func>`) (*destroy_func* `<g-destroy-notify>`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Like `pango-attr-shape-new`, but a user data pointer is also provided; this pointer can be accessed when later rendering the glyph.

ink-rect	ink rectangle to assign to each character
logical-rect	logical rectangle to assign to each character
data	user data pointer
copy-func	function to copy *data* when the attribute is copied. If '`#f`', *data* is simply copied as a pointer.
destroy-func	function to free *data* when the attribute is freed, or '`#f`'
ret	the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

Since 1.8

`pango-attr-scale-new` (*scale_factor* `double`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font size scale attribute. The base font for the affected text will have its size multiplied by *scale-factor*.

scale-factor	factor to scale the font

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-fallback-new` (*enable_fallback* `bool`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new font fallback attribute.

If fallback is disabled, characters will only be used from the closest matching font on the system. No fallback will be done to other fonts on the system that might contain the characters in the text.

enable-fallback
: '`#t`' if we should fall back on other fonts for characters the active font is missing.

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

Since 1.4

`pango-attr-rise-new` (*rise* `int`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new baseline displacement attribute.

rise the amount that the text should be displaced vertically, in Pango units. Positive values displace the text upwards.

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

`pango-attr-letter-spacing-new` (*letter_spacing* `int`) ⇒ (*ret* `<pango-attribute*>`) [Function]

Create a new letter-spacing attribute.

letter-spacing
: amount of extra space to add between graphemes of the text, in Pango units.

ret the newly allocated `<pango-attribute>`, which should be freed with `pango-attribute-destroy`.

Since 1.6

`pango-color-parse` (*self* `<pango-color>`) (*spec* `mchars`) ⇒ (*ret* `bool`) [Function]

Fill in the fields of a color from a string specification. The string can either one of a large set of standard names. (Taken from the X11 '`rgb.txt`' file), or it can be a hex value in the form '&#x0023;rgb' '&#x0023;rrggbb' '&#x0023;rrrgggbbb' or '&#x0023;rrrrggggbbbb' where 'r', 'g' and 'b' are hex digits of the red, green, and blue components of the color, respectively. (White in the four forms is '&#x0023;fff' '&#x0023;ffffff' '&#x0023;fffffffff' and '&#x0023;ffffffffffff')

color a `<pango-color>` structure in which to store the result

spec a string specifying the new color

ret '`#t`' if parsing of the specifier succeeded, otherwise false.

`pango-language-from-string` (*language* `mchars`) [Function]
⇒ (*ret* `<pango-language>`)

Take a RFC-3066 format language tag as a string and convert it to a `<pango-language>` pointer that can be efficiently copied (copy the pointer) and compared with other language tags (compare the pointer.)

This function first canonicalizes the string by converting it to lowercase, mapping '_' to '-', and stripping all characters other than letters and '-'.

language	a string representing a language tag
ret	an opaque pointer to a `<pango-language>` structure. this will be valid forever after.

`pango-language-matches` (*self* `<pango-language>`) [Function]
(*range_list* `mchars`) ⇒ (*ret* `bool`)

Checks if a language tag matches one of the elements in a list of language ranges. A language tag is considered to match a range in the list if the range is '*', the range is exactly the tag, or the range is a prefix of the tag, and the character after it in the tag is '-'.

language	a language tag (see `pango-language-from-string`), '`#f`' is allowed and matches nothing but '*'
range-list	a list of language ranges, separated by ';', ':', ',', or space characters. Each element must either be '*', or a RFC 3066 language range canonicalized as by `pango-language-from-string`
ret	'`#t`' if a match was found.

`pango-attr-list-new` ⇒ (*ret* `<pango-attr-list>`) [Function]

Create a new empty attribute list with a reference count of one.

ret	the newly allocated `<pango-attr-list>`, which should be freed with `pango-attr-list-unref`.

`pango-attr-list-insert` (*self* `<pango-attr-list>`) [Function]
(*attr* `<pango-attribute*>`)

Insert the given attribute into the `<pango-attr-list>`. It will be inserted after all other attributes with a matching *start-index*.

list	a `<pango-attr-list>`
attr	the attribute to insert. Ownership of this value is assumed by the list.

`pango-attr-list-insert-before` (*self* `<pango-attr-list>`) [Function]
(*attr* `<pango-attribute*>`)

Insert the given attribute into the `<pango-attr-list>`. It will be inserted before all other attributes with a matching *start-index*.

list	a `<pango-attr-list>`
attr	the attribute to insert. Ownership of this value is assumed by the list.

`pango-attr-list-change` (*self* `<pango-attr-list>`) [Function]
(*attr* `<pango-attribute*>`)

Insert the given attribute into the `<pango-attr-list>`. It will replace any attributes of the same type on that segment and be merged with any adjoining attributes that are identical.

This function is slower than `pango-attr-list-insert` for creating a attribute list in order (potentially much slower for large lists). However, `pango-attr-list-insert` is not suitable for continually changing a set of attributes since it never removes or combines existing attributes.

list	a `<pango-attr-list>`
attr	the attribute to insert. Ownership of this value is assumed by the list.

`pango-attr-list-splice` (*self* `<pango-attr-list>`) [Function]
(*other* `<pango-attr-list>`) (*pos* `int`) (*len* `int`)

This function opens up a hole in *list*, fills it in with attributes from the left, and then merges *other* on top of the hole.

This operation is equivalent to stretching every attribute that applies at position *pos* in *list* by an amount *len*, and then calling `pango-attr-list-change` with a copy of each attribute in *other* in sequence (offset in position by *pos*).

This operation proves useful for, for instance, inserting a pre-edit string in the middle of an edit buffer.

list	a `<pango-attr-list>`
other	another `<pango-attr-list>`
pos	the position in *list* at which to insert *other*
len	the length of the spliced segment. (Note that this must be specified since the attributes in *other* may only be present at some subsection of this range)

`pango-attr-list-filter` (*self* `<pango-attr-list>`) [Function]
(*func* `<pango-attr-filter-func>`) (*data* `<gpointer>`)
⇒ (*ret* `<pango-attr-list>`)

Given a `<pango-attr-list>` and callback function, removes any elements of *list* for which *func* returns '`#t`' and inserts them into a new list.

list	a `<pango-attr-list>`
func	callback function; returns '`#t`' if an attribute should be filtered out.
data	Data to be passed to *func*
ret	the new `<pango-attr-list>` or '`#f`' if no attributes of the given types were found.

Since 1.2

`pango-attr-list-get-iterator` (*self* `<pango-attr-list>`) [Function]
⇒ (*ret* `<pango-attr-iterator*>`)

Create a iterator initialized to the beginning of the list. *list* must not be modified until this iterator is freed.

list	a `<pango-attr-list>`
ret	the newly allocated `<pango-attr-iterator>`, which should be freed with `pango-attr-iterator-destroy`.

pango-attr-iterator-next (*self* `<pango-attr-iterator*>`) [Function]
⇒ (*ret* `bool`)

Advance the iterator until the next change of style.

iterator	a `<pango-attr-iterator>`
ret	'`#f`' if the iterator is at the end of the list, otherwise '`#t`'

pango-attr-iterator-range (*self* `<pango-attr-iterator*>`) [Function]
⇒ (*start* `int`) (*end* `int`)

Get the range of the current segment. Note that the stored return values are signed, not unsigned like the values in `<pango-attribute>`. To deal with this API oversight, stored return values that wouldn't fit into a signed integer are clamped to '`G_MAXINT`'.

iterator	a `<pango-attr-iterator>`
start	location to store the start of the range
end	location to store the end of the range

pango-attr-iterator-get (*self* `<pango-attr-iterator*>`) [Function]
(*type* `<pango-attr-type>`) ⇒ (*ret* `<pango-attribute*>`)

Find the current attribute of a particular type at the iterator location. When multiple attributes of the same type overlap, the attribute whose range starts closest to the current location is used.

iterator	a `<pango-attr-iterator>`
type	the type of attribute to find.
ret	the current attribute of the given type, or '`#f`' if no attribute of that type applies to the current location.

pango-attr-iterator-get-font (*self* `<pango-attr-iterator*>`) [Function]
(*desc* `<pango-font-description>`) (*language* `<pango-language**>`)
(*extra_attrs* `<gs-list**>`)

Get the font and other attributes at the current iterator position.

iterator	a `<pango-attr-iterator>`
desc	a `<pango-font-description>` to fill in with the current values. The family name in this structure will be set using `pango-font-description-set-family-static` using values from an attribute in the `<pango-attr-list>` associated with the iterator, so if you plan to keep it around, you must call: '`pango_font_description_set_family (desc, pango_font_description_get_family (desc))`'.
language	if non-'`#f`', location to store language tag for item, or '`#f`' if none is found.

extra-attrs
: if non-'`#f`', location in which to store a list of non-font attributes at the the current position; only the highest priority value of each attribute will be added to this list. In order to free this value, you must call `pango-attribute-destroy` on each member.

`pango-attr-iterator-get-attrs` (*self* `<pango-attr-iterator*>`) ⇒ (*ret* `gslist-of`) [Function]

Gets a list of all attributes at the current position of the iterator.

iterator a `<pango-attr-iterator>`

ret a list of all attributes for the current range. To free this value, call `pango-attribute-destroy` on each value and `g-slist-free` on the list.

Since 1.2

`pango-attr-iterator-destroy` (*self* `<pango-attr-iterator*>`) [Function]

Destroy a `<pango-attr-iterator>` and free all associated memory.

iterator a `<pango-attr-iterator>`.

Function Index