The LaTeX3 kernel: coffins*

The LaTeX3 Project†

Released 2011/03/23

Abstract

A LaTeX3 'coffin' is a design-level method for typesetting boxed material. The structure of coffins contains not only the boxed material itself but also information about the size of the box and potential alignment positions. This structure makes it is possible to build complex layouts rapidly by assembling coffins. The code here provides the low-level support system for coffins.

Contents

*This file describes v2209, last revised 2011/03/23.

†Frank Mittelbach, Denys Duchier, Chris Rowley, Rainer Schöpf, Johannes Braams, Michael Downes, David Carlisle, Alan Jeffrey, Morten Høgholm, Thomas Lotze, Javier Bezos, Will Robertson, Joseph Wright

1 Introduction

The material in this module provides the low-level support system for coffins. For details about the design concept of a coffin, see the xcoffins module.

2 Code-level functions

\coffin_new:N
\coffin_new:c
\coffin_new:N ⟨*coffin*⟩
Creates a new ⟨*coffin*⟩ or raises an error if the name is already taken. The declaration is global. The ⟨*coffin*⟩ will initially be empty.

\coffin_clear:N
\coffin_clear:c
\coffin_clear:N ⟨*coffin*⟩
Clears the content of the ⟨*coffin*⟩ within the current TEX group level.

\coffin_set_eq:NN
\coffin_set_eq:Nc
\coffin_set_eq:cN
\coffin_set_eq:cc
\coffin_set_eq:NN ⟨*coffin1*⟩ ⟨*coffin2*⟩
Sets both the content and poles of ⟨*coffin1*⟩ equal to those of ⟨*coffin2*⟩ within the current TEX group level.

\hcoffin_set:Nn
\hcoffin_set:cn
\hcoffin_set:Nn ⟨*coffin*⟩ {⟨*material*⟩}
Typesets the ⟨*material*⟩ in horizontal mode, storing the result in the ⟨*coffin*⟩. The standard poles for the ⟨*coffin*⟩ are then set up based on the size of the typeset material.

\vcoffin_set:Nnn
\vcoffin_set:cnn
\vcoffin_set:Nnn ⟨*coffin*⟩ {⟨*width*⟩} {⟨*material*⟩}
Typesets the ⟨*material*⟩ in vertical mode constrained to the given ⟨*width*⟩ and stores the result in the ⟨*coffin*⟩. The standard poles for the ⟨*coffin*⟩ are then set up based on the size of the typeset material.

\coffin_set_horizontal_pole:Nnn
\coffin_set_horizontal_pole:cnn
\coffin_set_horizontal_pole:Nnn ⟨*coffin*⟩ {⟨*pole*⟩} {⟨*offset*⟩}

Sets the ⟨*pole*⟩ to run horizontally through the ⟨*coffin*⟩. The ⟨*pole*⟩ will be located at the ⟨*offset*⟩ from the bottom edge of the bounding box of the ⟨*coffin*⟩. The ⟨*offset*⟩ should be given as a dimension expression; this may include the terms csTotalHeight, `\Height`, `\Depth` and `\Width`, which will evaluate to the appropriate dimensions of the ⟨*coffin*⟩.

`\coffin_set_vertical_pole:Nnn`
`\coffin_set_vertical_pole:cnn`

`\coffin_set_vertical_pole:Nnn` ⟨*coffin*⟩ {⟨*pole*⟩} {⟨*offset*⟩}

Sets the ⟨*pole*⟩ to run vertically through the ⟨*coffin*⟩. The ⟨*pole*⟩ will be located at the ⟨*offset*⟩ from the left-hand edge of the bounding box of the ⟨*coffin*⟩. The ⟨*offset*⟩ should be given as a dimension expression; this may include the terms `\TotalHeight`, `\Height`, `\Depth` and `\Width`, which will evaluate to the appropriate dimensions of the ⟨*coffin*⟩.

`\coffin_rotate:Nn`
`\coffin_rotate:cn`

`\coffin_rotate:Nn` ⟨*coffin*⟩ {⟨*angle*⟩}

Rotates the ⟨*coffin*⟩ by the given ⟨*angle*⟩ (given in degrees counter-clockwise). This process will rotate both the coffin content and poles. Multiple rotations will not result in the bounding box of the coffin growing unnecessarily.

`\coffin_attach:NnnNnnnn`
`\coffin_attach:cnnNnnnn`
`\coffin_attach:Nnncnnnn`
`\coffin_attach:cnncnnnn`

`\coffin_attach:NnnNnnnn`
⟨*coffin1*⟩ {⟨*coffin*$_1$-*pole*$_1$⟩} {⟨*coffin*$_1$-*pole*$_2$⟩}
⟨*coffin2*⟩ {⟨*coffin*$_2$-*pole*$_1$⟩} {⟨*coffin*$_2$-*pole*$_2$⟩}
{⟨*x-offset*⟩} {⟨*y-offset*⟩}

This function carries out alignment such that the bounding box of ⟨*coffin1*⟩ is not altered, *i.e.* ⟨*coffin2*⟩ can protrude outside of the bounding box of the coffin. The alignment is carried out by first calculating ⟨*handle1*⟩, the point of intersection of ⟨*coffin1-pole1*⟩ and ⟨*coffin1-pole2*⟩, and ⟨*handle2*⟩, the point of intersection of ⟨*coffin2-pole1*⟩ and ⟨*coffin2-pole2*⟩. ⟨*coffin2*⟩ is then attached to ⟨*coffin1*⟩ such that the relationship between ⟨*handle1*⟩ and ⟨*handle2*⟩ is described by the ⟨*x-offset*⟩ and ⟨*y-offset*⟩. The two offsets should be given as dimension expressions.

`\coffin_join:NnnNnnnn`
`\coffin_join:cnnNnnnn`
`\coffin_join:Nnncnnnn`
`\coffin_join:cnncnnnn`

`\coffin_join:NnnNnnnn`
⟨*coffin1*⟩ {⟨*coffin*$_1$-*pole*$_1$⟩} {⟨*coffin*$_1$-*pole*$_2$⟩}
⟨*coffin2*⟩ {⟨*coffin*$_2$-*pole*$_1$⟩} {⟨*coffin*$_2$-*pole*$_2$⟩}
{⟨*x-offset*⟩} {⟨*y-offset*⟩}

This function carries out alignment such that the bounding box of ⟨*coffin1*⟩ after the process will expand. The new bounding box will cover the area contain the bounding boxes of the two original coffins. The alignment is carried out by first calculating ⟨*handle1*⟩, the point of intersection of ⟨*coffin1-pole1*⟩ and ⟨*coffin1-pole2*⟩, and ⟨*handle2*⟩,

the point of intersection of ⟨*coffin2-pole1*⟩ and ⟨*coffin2-pole2*⟩. ⟨*coffin2*⟩ is then attached to ⟨*coffin1*⟩ such that the relationship between ⟨*handle1*⟩ and ⟨*handle2*⟩ is described by the ⟨*x-offset*⟩ and ⟨*y-offset*⟩. The two offsets should be given as dimension expressions.

`\coffin_typeset:Nnnnn`
`\coffin_typeset:cnnnn`
`\coffin_typeset:Nnnnn` ⟨*coffin*⟩ `{`⟨*pole*$_1$⟩`}` `{`⟨*pole*$_2$⟩`}` `{`⟨*x-offset*⟩`}` `{`⟨*y-offset*⟩`}`

Typesetting is carried out by first calculating ⟨*handle*⟩, the point of intersection of ⟨*pole1*⟩ and ⟨*pole2*⟩. The coffin is then typeset such that the relationship between the current reference point in the document and the ⟨*handle*⟩ is described by the ⟨*x-offset*⟩ and ⟨*y-offset*⟩. The two offsets should be given as dimension expressions. Typesetting a coffin is therefore analogous to carrying out an alignment where the 'parent' coffin is the current insertion point.

`\coffin_display_handles:cn`
`\coffin_display_handles:cn`
`\coffin_display_handles:Nn` ⟨*coffin*⟩ `{`⟨*colour*⟩`}`

This function first calculates the intersections between all of the ⟨*poles*⟩ of the ⟨*coffin*⟩ to give a set of ⟨*handles*⟩. It then prints the ⟨*coffin*⟩ at the current location in the source, with the position of the ⟨*handles*⟩ marked on the coffin. The ⟨*handles*⟩ will be labelled as part of this process: the locations of the ⟨*handles*⟩ and the labels are both printed in the ⟨*colour*⟩ specified.

`\coffin_mark_handle:Nnnn`
`\coffin_mark_handle:cnnn`
`\coffin_mark_handle:Nnnn` ⟨*coffin*⟩ `{`⟨*pole*$_1$⟩`}` `{`⟨*pole*$_2$⟩`}` `{`⟨*colour*⟩`}`

This function first calculates the ⟨*handle*⟩ for the ⟨*coffin*⟩ as defined by the intersection of ⟨*pole1*⟩ and ⟨*pole2*⟩. It then marks the position of the ⟨*handle*⟩ on the ⟨*coffin*⟩. The ⟨*handle*⟩ will be labelled as part of this process: the location of the ⟨*handle*⟩ and the label are both printed in the ⟨*colour*⟩ specified.

`\coffin_show_structure:N`
`\coffin_show_structure:c`
`\coffin_show_structure:N` ⟨*coffin*⟩

This function shows the structural information about the ⟨*coffin*⟩ in the terminal. The width, height and depth of the typeset material are given, along with the location of all of the poles of the coffin.

Notice that the poles of a coffin are defined by four values: the x and y co-ordinates of a point that the pole passes through and the x- and y-components of a vector denoting the direction of the pole. It is the ratio between the later, rather than the absolute values, which determines the direction of the pole.

3 Implementation

```
1 ⟨*package⟩
```

```
2 \ProvidesExplPackage
3   {\filename}{\filedate}{\fileversion}{\filedescription}
```

3.1 Coffins: data structures and general variables

`\l_coffin_tmp_box`
`\l_coffin_tmp_dim`
`\l_coffin_tmp_fp`
`\l_coffin_tmp_tl`

Scratch variables.

```
4 \box_new:N \l_coffin_tmp_box
5 \dim_new:N \l_coffin_tmp_dim
6 \fp_new:N  \l_coffin_tmp_fp
7 \tl_new:N  \l_coffin_tmp_tl
```

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

`\c_coffin_corners_prop`

The 'corners;' of a coffin define the real content, as opposed to the TEX bounding box. They all start off in the same place, of course.

```
8 \prop_new:N \c_coffin_corners_prop
9 \tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } }
10 \prop_put:Nno \c_coffin_corners_prop { tl } { \l_coffin_tmp_tl }
11 \prop_put:Nno \c_coffin_corners_prop { tr } { \l_coffin_tmp_tl }
12 \prop_put:Nno \c_coffin_corners_prop { bl } { \l_coffin_tmp_tl }
13 \prop_put:Nno \c_coffin_corners_prop { br } { \l_coffin_tmp_tl }
```

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

`\c_coffin_poles_prop`

Pole positions are given for horizontal, vertical and reference-point based values.

```
14 \prop_new:N \c_coffin_poles_prop
15 \tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } { 0 pt } { 1000 pt } }
16 \prop_put:Nno \c_coffin_poles_prop { l }  { \l_coffin_tmp_tl }
17 \prop_put:Nno \c_coffin_poles_prop { hc } { \l_coffin_tmp_tl }
18 \prop_put:Nno \c_coffin_poles_prop { r }  { \l_coffin_tmp_tl }
19 \tl_set:Nn \l_coffin_tmp_tl { { 0 pt } { 0 pt } { 1000 pt } { 0 pt } }
20 \prop_put:Nno \c_coffin_poles_prop { b }  { \l_coffin_tmp_tl }
21 \prop_put:Nno \c_coffin_poles_prop { vc } { \l_coffin_tmp_tl }
22 \prop_put:Nno \c_coffin_poles_prop { t }  { \l_coffin_tmp_tl }
23 \prop_put:Nno \c_coffin_poles_prop { B }  { \l_coffin_tmp_tl }
24 \prop_put:Nno \c_coffin_poles_prop { H }  { \l_coffin_tmp_tl }
25 \prop_put:Nno \c_coffin_poles_prop { T }  { \l_coffin_tmp_tl }
```

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

`\l_coffin_calc_a_fp`
`\l_coffin_calc_b_fp`
`\l_coffin_calc_c_fp`
`\l_coffin_calc_d_fp`
`\l_coffin_calc_result_fp`

Used for calculations of intersections and in other internal places.

```
26 \fp_new:N \l_coffin_calc_a_fp
27 \fp_new:N \l_coffin_calc_b_fp
```

```
\fp_new:N \l_coffin_calc_c_fp
\fp_new:N \l_coffin_calc_d_fp
\fp_new:N \l_coffin_calc_result_fp
```

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

`\l_coffin_error_bool` For propagating errors so that parts of the code can work around them.

```
\bool_new:N \l_coffin_error_bool
```

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

`\l_coffin_offset_x_dim` `\l_coffin_offset_y_dim` The offset between two sets of coffin handles when typesetting. These values are corrected from those requested in an alignment for the positions of the handles.

```
\dim_new:N \l_coffin_offset_x_dim
\dim_new:N \l_coffin_offset_y_dim
```

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

`\l_coffin_pole_a_tl` `\l_coffin_pole_b_tl` Needed for finding the intersection of two poles.

```
\tl_new:N \l_coffin_pole_a_tl
\tl_new:N \l_coffin_pole_b_tl
```

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

`\l_coffin_sin_fp` `\l_coffin_cos_fp` Used for rotations to get the sine and cosine values.

```
\fp_new:N \l_coffin_sin_fp
\fp_new:N \l_coffin_cos_fp
```

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

`\l_coffin_x_dim` `\l_coffin_y_dim` `\l_coffin_x_prime_dim` `\l_coffin_y_prime_dim` For calculating intersections and so forth.

```
\dim_new:N \l_coffin_x_dim
\dim_new:N \l_coffin_y_dim
\dim_new:N \l_coffin_x_prime_dim
\dim_new:N \l_coffin_y_prime_dim
```

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

`\l_coffin_x_fp` `\l_coffin_y_fp` `\l_coffin_x_prime_fp` `\l_coffin_y_prime_fp` Used for calculations where there are clear x- and y-components, for example during vector rotation.

```
\fp_new:N \l_coffin_x_fp
\fp_new:N \l_coffin_y_fp
\fp_new:N \l_coffin_x_prime_fp
\fp_new:N \l_coffin_y_prime_fp
```

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

`\l_coffin_Depth_dim`
`\l_coffin_Height_dim`
`\l_coffin_TotalHeight_dim`
`\l_coffin_Width_dim`

Dimensions for the various parts of a coffin.

```
\dim_new:N \l_coffin_Depth_dim
\dim_new:N \l_coffin_Height_dim
\dim_new:N \l_coffin_TotalHeight_dim
\dim_new:N \l_coffin_Width_dim
```

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

`\coffin_saved_Depth:`
`\coffin_saved_Height:`
`\coffin_saved_TotalHeight:`
`\coffin_saved_Width:`

Used to save the meaning of `\Depth`, `\Height`, `\TotalHeight` and `\Width`.

```
\cs_new_nopar:Npn \coffin_saved_Depth:       { }
\cs_new_nopar:Npn \coffin_saved_Height:      { }
\cs_new_nopar:Npn \coffin_saved_TotalHeight: { }
\cs_new_nopar:Npn \coffin_saved_Width:       { }
```

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

3.2 Basic coffin functions

There are a number of basic functions needed for creating coffins and placing material in them. This all relies on the following data structures.

`\coffin_if_exist_execute:Nn`

Several of the higher-level coffin functions will give multiple errors if the coffin does not exist. A cleaner way to handle this is provided here: both the box and the coffin structure are checked.

```
\cs_new_protected:Npn \coffin_if_exist_execute:Nn #1#2 {
  \cs_if_exist:NTF #1
    {
      \cs_if_exist:cTF { l_coffin_poles_ \tex_number:D #1 _prop }
        { #2 }
        {
          \msg_kernel_error:nnx { coffin } { unknown-coffin }
            { \token_to_str:N #1 }
      }
    }
    {
      \msg_kernel_error:nnx { coffin } { unknown-coffin }
        { \token_to_str:N #1 }
    }
}
```

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

\coffin_clear:N
\coffin_clear:c

Clearing coffins means emptying the box and resetting all of the structures.

```
\cs_new_protected_nopar:Npn \coffin_clear:N #1 {
  \coffin_if_exist_execute:Nn #1
    {
      \box_clear:N #1
      \coffin_reset_structure:N #1
    }
}
\cs_generate_variant:Nn \coffin_clear:N { c }
```

(*End definition for* `\coffin_clear:N`, `\coffin_clear:c`, *and* . *These functions are documented on page* 2.)

\coffin_new:N
\coffin_new:c

Creating a new coffin means making the underlying box and adding the data structures. These are created globally, as there is a need to avoid any strange effects if the coffin is created inside a group. This means that the usual rule about `\l_...` variables has to be broken.

```
\cs_new_protected_nopar:Npn \coffin_new:N #1 {
  \box_new:N #1
  \cs_gundefine:c { l_coffin_corners_ \tex_number:D #1 _prop }
  \cs_gundefine:c { l_coffin_poles_ \tex_number:D #1 _prop }
  \prop_new:c { l_coffin_corners_ \tex_number:D #1 _prop }
  \prop_new:c { l_coffin_poles_ \tex_number:D #1 _prop }
  \prop_gset_eq:cN { l_coffin_corners_ \tex_number:D #1 _prop }
    \c_coffin_corners_prop
  \prop_gset_eq:cN { l_coffin_poles_ \tex_number:D #1 _prop }
    \c_coffin_poles_prop
}
\cs_generate_variant:Nn \coffin_new:N { c }
```

(*End definition for* `\coffin_new:N`, `\coffin_new:c`, *and* . *These functions are documented on page* 2.)

\hcoffin_set:Nn
\hcoffin_set:cn

Horizontal coffins are relatively easy: set the appropriate box, reset the structures then update the handle positions.

```
\cs_new_protected:Npn \hcoffin_set:Nn #1#2 {
  \coffin_if_exist_execute:Nn #1
    {
      \hbox_set:Nn #1
        {
          \group_begin:
            \set@color
            #2
          \group_end:
        }
      \coffin_reset_structure:N #1
      \coffin_update_poles:N #1
      \coffin_update_corners:N #1
```

```
    }
}
\cs_generate_variant:Nn \hcoffin_set:Nn { c }
```

(*End definition for* `\hcoffin_set:Nn`, `\hcoffin_set:cn`, *and* . *These functions are documented on page* 2.)

`\vcoffin_set:Nnn`
`\vcoffin_set:cnn`

Setting vertical coffins is more complex. First, the material is typeset with a given width. The default handles and poles are set as for a horizontal coffin, before finding the top baseline using a temporary box.

```
\cs_new_protected:Npn \vcoffin_set:Nnn #1#2#3 {
  \coffin_if_exist_execute:Nn #1
    {
      \vbox_set:Nn #1
        {
          \dim_set:Nn \tex_hsize:D {#2}
          \group_begin:
            \set@color
            #3
          \group_end:
        }
      \coffin_reset_structure:N #1
      \coffin_update_poles:N #1
      \coffin_update_corners:N #1
      \vbox_set_top:Nn \l_coffin_tmp_box { \vbox_unpack:N #1 }
      \dim_set:Nn \l_coffin_tmp_dim
        { \box_ht:N #1 - \box_ht:N \l_coffin_tmp_box }
      \coffin_set_pole:Nnx #1 { T }
        {
          { 0 pt } { \dim_use:N \l_coffin_tmp_dim } { 1000 pt } { 0 pt }
        }
    }
}
\cs_generate_variant:Nn \vcoffin_set:Nnn { c }
```

(*End definition for* `\vcoffin_set:Nnn`, `\vcoffin_set:cnn`, *and* . *These functions are documented on page* 2.)

`\coffin_set_eq:NN`
`\coffin_set_eq:Nc`
`\coffin_set_eq:cN`
`\coffin_set_eq:cc`

Setting two coffins equal is just a wrapper around other functions.

```
\cs_new_protected_nopar:Npn \coffin_set_eq:NN #1#2 {
  \coffin_if_exist_execute:Nn #1
    {
      \box_set_eq:NN #1 #2
      \coffin_set_eq_structure:NN #1 #2
    }
}
\cs_generate_variant:Nn \coffin_set_eq:NN { c , Nc , cc }
```

(*End definition for* `\coffin_set_eq:NN` *and others. These functions are documented on page* 2.)

\c_empty_coffin
\l_coffin_aligned_coffin
\l_coffin_aligned_internal_coffin

Special coffins: these cannot be set up earlier as they need `\coffin_new:N`. The empty coffin is set as a box as the full coffin-setting system needs some material which is not yet available.

```
\coffin_new:N \c_empty_coffin
\hbox_set:Nn  \c_empty_coffin { }
\coffin_new:N \l_coffin_aligned_coffin
\coffin_new:N \l_coffin_aligned_internal_coffin
```

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

3.3 Coffins: handle and pole management

\coffin_get_pole:NnN

A simple wrapper around the recovery of a coffin pole, with some error checking and recovery built-in.

```
\cs_set_protected_nopar:Npn \coffin_get_pole:NnN #1#2#3 {
  \prop_get:cnN { l_coffin_poles_ \tex_number:D #1 _prop } {#2} #3
  \quark_if_no_value:NT #3
    {
      \msg_kernel_error:nnxx { coffin } { unknown-coffin-pole }
        {#2} { \token_to_str:N #1 }
      \tl_set:Nn #3 { { 0 pt } { 0 pt } { 0 pt } { 0 pt } }
    }
}
```

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

\coffin_reset_structure:N

Resetting the structure is a simple copy job.

```
\cs_new_protected_nopar:Npn \coffin_reset_structure:N #1 {
  \prop_set_eq:cN { l_coffin_corners_ \tex_number:D #1 _prop }
    \c_coffin_corners_prop
  \prop_set_eq:cN { l_coffin_poles_ \tex_number:D #1 _prop }
    \c_coffin_poles_prop
}
```

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

\coffin_set_eq_structure:NN

Setting coffin structures equal simply means copying the property list.

```
\cs_new_protected_nopar:Npn \coffin_set_eq_structure:NN #1#2 {
  \prop_set_eq:cc { l_coffin_corners_ \tex_number:D #1 _prop }
    { l_coffin_corners_ \tex_number:D #2 _prop }
  \prop_set_eq:cc { l_coffin_poles_ \tex_number:D #1 _prop }
    { l_coffin_poles_ \tex_number:D #2 _prop }
}
```

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

\coffin_set_user_dimensions:N
\coffin_end_user_dimensions:
\Depth
\Height
\TotalHeight
\Width

These make design-level names for the dimensions of a coffin easy to get at.

```
\cs_new_protected_nopar:Npn \coffin_set_user_dimensions:N #1 {
  \cs_set_eq:NN \coffin_saved_Height:      \Height
  \cs_set_eq:NN \coffin_saved_Depth:       \Depth
  \cs_set_eq:NN \coffin_saved_TotalHeight: \TotalHeight
  \cs_set_eq:NN \coffin_saved_Width:       \Width
  \cs_set_eq:NN \Height      \l_coffin_Height_dim
  \cs_set_eq:NN \Depth       \l_coffin_Depth_dim
  \cs_set_eq:NN \TotalHeight \l_coffin_TotalHeight_dim
  \cs_set_eq:NN \Width       \l_coffin_Width_dim
  \dim_set:Nn \Height      { \box_ht:N #1 }
  \dim_set:Nn \Depth       { \box_dp:N #1 }
  \dim_set:Nn \TotalHeight { \box_ht:N #1 - \box_dp:N #1 }
  \dim_set:Nn \Width       { \box_wd:N #1 }
}
\cs_new_protected_nopar:Npn \coffin_end_user_dimensions: {
  \cs_set_eq:NN \Height      \coffin_saved_Height:
  \cs_set_eq:NN \Depth       \coffin_saved_Depth:
  \cs_set_eq:NN \TotalHeight \coffin_saved_TotalHeight:
  \cs_set_eq:NN \Width       \coffin_saved_Width:
}
```

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

\coffin_set_horizontal_pole:Nnn
\coffin_set_horizontal_pole:cnn
\coffin_set_vertical_pole:Nnn
\coffin_set_vertical_pole:cnn
\coffin_set_pole:Nnn
\coffin_set_pole:Nnx

Setting the pole of a coffin at the user/designer level requires a bit more care. The idea here is to provide a reasonable interface to the system, then to do the setting with full expansion. The three-argument version is used internally to do a direct setting.

```
\cs_new_protected_nopar:Npn \coffin_set_horizontal_pole:Nnn #1#2#3 {
  \coffin_if_exist_execute:Nn #1
    {
      \coffin_set_user_dimensions:N #1
      \coffin_set_pole:Nnx #1 {#2}
        {
          { 0 pt } { \dim_eval:n {#3} }
          { 1000 pt } { 0 pt }
        }
      \coffin_end_user_dimensions:
    }
}
\cs_new_protected_nopar:Npn \coffin_set_vertical_pole:Nnn #1#2#3 {
  \coffin_if_exist_execute:Nn #1
    {
      \coffin_set_user_dimensions:N #1
      \coffin_set_pole:Nnx #1 {#2}
        {
          { \dim_eval:n {#3} } { 0 pt }
          { 0 pt } { 1000 pt }
        }
```

```
      \coffin_end_user_dimensions:
    }
}
\cs_new_protected_nopar:Npn \coffin_set_pole:Nnn #1#2#3 {
  \prop_put:cnn { l_coffin_poles_ \tex_number:D #1 _prop } {#2} {#3}
}
\cs_generate_variant:Nn \coffin_set_horizontal_pole:Nnn { c }
\cs_generate_variant:Nn \coffin_set_vertical_pole:Nnn { c }
\cs_generate_variant:Nn \coffin_set_pole:Nnn { Nnx }
```

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

`\coffin_update_corners:N` Updating the corners of a coffin is straight-forward as at this stage there can be no rotation. So the corners of the content are just those of the underlying TEX box.

```
\cs_new_protected_nopar:Npn \coffin_update_corners:N #1 {
  \prop_put:cnx { l_coffin_corners_ \tex_number:D #1 _prop } { tl }
    { { 0 pt } { \dim_use:N \box_ht:N #1 } }
  \prop_put:cnx { l_coffin_corners_ \tex_number:D #1 _prop } { tr }
    { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
  \dim_set:Nn \l_coffin_tmp_dim { - \dim_use:N \box_dp:N #1 }
  \prop_put:cnx { l_coffin_corners_ \tex_number:D #1 _prop } { bl }
    { { 0 pt } { \dim_use:N \l_coffin_tmp_dim } }
  \prop_put:cnx { l_coffin_corners_ \tex_number:D #1 _prop } { br }
    { { \dim_use:N \box_wd:N #1 } { \dim_use:N \l_coffin_tmp_dim } }
}
```

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

`\coffin_update_poles:N` This function is called when a coffin is set, and updates the poles to reflect the nature of size of the box. Thus this function only alters poles where the default position is dependent on the size of the box. It also does not set poles which are relevant only to vertical coffins.

```
\cs_new_protected_nopar:Npn \coffin_update_poles:N #1 {
  \dim_set:Nn \l_coffin_tmp_dim { 0.5 \box_wd:N #1 }
  \prop_put:cnx { l_coffin_poles_ \tex_number:D #1 _prop } { hc }
    { { \dim_use:N \l_coffin_tmp_dim } { 0 pt } { 0 pt } { 1000 pt } }
  \prop_put:cnx { l_coffin_poles_ \tex_number:D #1 _prop } { r }
    { { \dim_use:N \box_wd:N #1 } { 0 pt } { 0 pt } { 1000 pt } }
  \dim_set:Nn \l_coffin_tmp_dim { ( \box_ht:N #1 - \box_dp:N #1 ) / 2 }
  \prop_put:cnx { l_coffin_poles_ \tex_number:D #1 _prop } { vc }
    { { 0 pt } { \dim_use:N \l_coffin_tmp_dim } { 1000 pt } { 0 pt } }
  \prop_put:cnx { l_coffin_poles_ \tex_number:D #1 _prop } { t }
    { { 0 pt } { \dim_use:N \box_ht:N #1 } { 1000 pt } { 0 pt } }
  \dim_set:Nn \l_coffin_tmp_dim { \box_dp:N #1 }
  \dim_compare:nNnF { \l_coffin_tmp_dim } = { \c_zero_dim }
    { \dim_set:Nn \l_coffin_tmp_dim { -\l_coffin_tmp_dim } }
  \prop_put:cnx { l_coffin_poles_ \tex_number:D #1 _prop } { b }
```

```
    { { 0 pt } { \dim_use:N \l_coffin_tmp_dim } { 1000 pt } { 0 pt } }
}
```

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

3.4 Coffins: calculation of pole intersections

`\coffin_calculate_intersection:Nnn`
`\coffin_calculate_intersection:nnnnnnnn`
`\coffin_calculate_intersection_aux:nnnnnN`

The lead off in finding intersections is to recover the two poles and then hand off to the auxiliary for the actual calculation. There may of course not be an intersection, for which an error trap is needed.

```
\cs_new_protected_nopar:Npn \coffin_calculate_intersection:Nnn #1#2#3 {
  \coffin_get_pole:NnN #1 {#2} \l_coffin_pole_a_tl
  \coffin_get_pole:NnN #1 {#3} \l_coffin_pole_b_tl
  \bool_set_false:N \l_coffin_error_bool
  \exp_last_two_unbraced:Noo
    \coffin_calculate_intersection:nnnnnnnn
      \l_coffin_pole_a_tl \l_coffin_pole_b_tl
  \bool_if:NT \l_coffin_error_bool
    {
      \msg_kernel_error:nn { coffin } { no-pole-intersection }
      \dim_zero:N \l_coffin_x_dim
      \dim_zero:N \l_coffin_y_dim
    }
}
```

The two poles passed here each have four values (as dimensions), (a, b, c, d) and (a', b', c', d'). These are arguments 1–4 and 5–8, respectively. In both cases a and b are the co-ordinates of a point on the pole and c and d define the direction of the pole. Finding the intersection depends on the directions of the poles, which are given by d/c and d'/c'. However, if one of the poles is either horizontal or vertical then one or more of c, d, c' and d' will be zero and a special case is needed.

```
\cs_new_protected_nopar:Npn
  \coffin_calculate_intersection:nnnnnnnn #1#2#3#4#5#6#7#8 {
  \dim_compare:nNnTF {#3} = { \c_zero_dim }
```

The case where the first pole is vertical. So the x-component of the interaction will be at a. There is then a test on the second pole: if it is also vertical then there is an error.

```
    {
      \dim_set:Nn \l_coffin_x_dim {#1}
      \dim_compare:nNnTF {#7} = { \c_zero_dim }
        { \bool_set_true:N \l_coffin_error_bool }
```

The second pole may still be horizontal, in which case the y-component of the intersection will be b'. If not,

$$y = \frac{d'}{c'}\,(x - a') + b'$$

with the x-component already known to be `#1`. This calculation is done as a generalised auxiliary.

```
      {
        \dim_compare:nNnTF {#8} = { \c_zero_dim }
          { \dim_set:Nn \l_coffin_y_dim {#6} }
          {
            \coffin_calculate_intersection_aux:nnnnnN
              {#1} {#5} {#6} {#7} {#8} \l_coffin_y_dim
          }
      }
  }
```

If the first pole is not vertical then it may be horizontal. If so, then the procedure is essentially the same as that already done but with the x- and y-components interchanged.

```
  {
    \dim_compare:nNnTF {#4} = { \c_zero_dim }
      {
        \dim_set:Nn \l_coffin_y_dim {#2}
        \dim_compare:nNnTF {#8} = { \c_zero_dim }
          { \bool_set_true:N \l_coffin_error_bool }
          {
            \dim_compare:nNnTF {#7} = { \c_zero_dim }
              { \dim_set:Nn \l_coffin_x_dim {#5} }
```

The formula for the case where the second pole is neither horizontal nor vertical is

$$x = \frac{c'}{d'}\,(y - b') + a'$$

which is again handled by the same auxiliary.

```
            {
              \coffin_calculate_intersection_aux:nnnnnN
                {#2} {#6} {#5} {#8} {#7} \l_coffin_x_dim
            }
        }
    }
```

The first pole is neither horizontal nor vertical. This still leaves the second pole, which may be a special case. For those possibilities, the calculations are the same as above with the first and second poles interchanged.

```
    {
      \dim_compare:nNnTF {#7} = { \c_zero_dim }
        {
          \dim_set:Nn \l_coffin_x_dim {#5}
          \coffin_calculate_intersection_aux:nnnnnN
            {#5} {#1} {#2} {#3} {#4} \l_coffin_y_dim
        }
```

```
    {
      \dim_compare:nNnTF {#8} = { \c_zero_dim }
        {
          \dim_set:Nn \l_coffin_x_dim {#6}
          \coffin_calculate_intersection_aux:nnnnnN
            {#6} {#2} {#1} {#4} {#3} \l_coffin_x_dim
        }
```

If none of the special cases apply then there is still a need to check that there is a unique intersection between the two pole. This is the case if they have different slopes.

```
        {
          \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#3}
          \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#4}
          \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#7}
          \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#8}
          \fp_div:Nn \l_coffin_calc_b_fp \l_coffin_calc_a_fp
          \fp_div:Nn \l_coffin_calc_d_fp \l_coffin_calc_c_fp
          \fp_compare:nNnTF
            { \l_coffin_calc_b_fp } = { \l_coffin_calc_d_fp }
            { \bool_set_true:N \l_coffin_error_bool }
```

All of the tests pass, so there is the full complexity of the calculation:

$$x = \frac{a(d/c) - a'(d'/c') - b + b'}{(d/c) - (d'/c')}$$

and noting that the two ratios are already worked out from the test just performed. There is quite a bit of shuffling from dimensions to floating points in order to do the work. The y-values is then worked out using the standard auxiliary starting from the x-position.

```
          {
            \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#6}
            \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
            \fp_sub:Nn \l_coffin_calc_result_fp
              { \l_coffin_calc_a_fp }
            \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#1}
            \fp_mul:Nn \l_coffin_calc_a_fp
              { \l_coffin_calc_b_fp }
            \fp_add:Nn \l_coffin_calc_result_fp
              { \l_coffin_calc_a_fp }
            \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#5}
            \fp_mul:Nn \l_coffin_calc_a_fp
              { \l_coffin_calc_d_fp }
            \fp_sub:Nn \l_coffin_calc_result_fp
              { \l_coffin_calc_a_fp }
            \fp_sub:Nn \l_coffin_calc_b_fp
              { \l_coffin_calc_d_fp }
            \fp_div:Nn \l_coffin_calc_result_fp
```

```
                    { \l_coffin_calc_b_fp }
                  \dim_set:Nn \l_coffin_x_dim
                    { \fp_to_dim:N \l_coffin_calc_result_fp }
                  \coffin_calculate_intersection_aux:nnnnnN
                    { \l_coffin_x_dim }
                    {#5} {#6} {#8} {#7} \l_coffin_y_dim
                }
            }
        }
    }
}
```

The formula for finding the intersection point is in most cases the same. The formula here is

$$\#6 = \frac{\#5}{\#4}(\#1 - \#2) + \#3$$

Thus `#4` and `#5` should be the directions of the pole while `#2` and `#3` are co-ordinates.

```
\cs_new_protected_nopar:Npn
  \coffin_calculate_intersection_aux:nnnnnN #1#2#3#4#5#6 {
    \fp_set_from_dim:Nn \l_coffin_calc_result_fp {#1}
    \fp_set_from_dim:Nn \l_coffin_calc_a_fp {#2}
    \fp_set_from_dim:Nn \l_coffin_calc_b_fp {#3}
    \fp_set_from_dim:Nn \l_coffin_calc_c_fp {#4}
    \fp_set_from_dim:Nn \l_coffin_calc_d_fp {#5}
    \fp_sub:Nn \l_coffin_calc_result_fp { \l_coffin_calc_a_fp }
    \fp_div:Nn \l_coffin_calc_result_fp { \l_coffin_calc_d_fp }
    \fp_mul:Nn \l_coffin_calc_result_fp { \l_coffin_calc_c_fp }
    \fp_add:Nn \l_coffin_calc_result_fp { \l_coffin_calc_b_fp }
    \dim_set:Nn #6 { \fp_to_dim:N \l_coffin_calc_result_fp }
}
```

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

3.5 Aligning and typesetting of coffins

`\coffin_join:NnnNnnnn`
`\coffin_join:cnnNnnnn`
`\coffin_join:Nnncnnnn`
`\coffin_join:cnncnnnn`

This command joins two coffins, using a horizontal and vertical pole from each coffin and making an offset between the two. The result is stored as the as a third coffin, which will have all of its handles reset to standard values. First, the more basic alignment function is used to get things started.

```
\cs_new_protected_nopar:Npn \coffin_join:NnnNnnnn #1#2#3#4#5#6#7#8 {
  \coffin_align:NnnNnnnnN
    #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
```

Correct the placement of the reference point. If the x-offset is negative then the reference point of the second box is to the left of that of the first, which is corrected using a kern.

On the right side the first box might stick out, which will show up if it is wider than the sum of the x-offset and the width of the second box. So a second kern may be needed.

```
\hbox_set:Nn \l_coffin_aligned_coffin
  {
    \dim_compare:nNnT { \l_coffin_offset_x_dim } < { \c_zero_dim }
      { \tex_kern:D -\l_coffin_offset_x_dim }
    \hbox_unpack:N \l_coffin_aligned_coffin
    \dim_set:Nn \l_coffin_tmp_dim
      { \l_coffin_offset_x_dim - \box_wd:N #1 + \box_wd:N #4 }
    \dim_compare:nNnT { \l_coffin_tmp_dim } < { \c_zero_dim }
      { \tex_kern:D -\l_coffin_tmp_dim }
  }
```

The coffin structure is reset, and the corners are cleared: only those from the two parent coffins are needed.

```
\coffin_reset_structure:N \l_coffin_aligned_coffin
\prop_clear:c
  {
    l_coffin_corners_ \tex_number:D \l_coffin_aligned_coffin _ prop
  }
\coffin_update_poles:N \l_coffin_aligned_coffin
```

The structures of the parent coffins are now transferred to the new coffin, which requires that the appropriate offsets are applied. That will then depend on whether any shift was needed.

```
\dim_compare:nNnTF { \l_coffin_offset_x_dim } < { \c_zero_dim }
  {
    \coffin_offset_poles:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
    \coffin_offset_poles:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
    \coffin_offset_corners:Nnn #1 { -\l_coffin_offset_x_dim } { 0 pt }
    \coffin_offset_corners:Nnn #4 { 0 pt } { \l_coffin_offset_y_dim }
  }
  {
    \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
    \coffin_offset_poles:Nnn #4
      { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
    \coffin_offset_corners:Nnn #1 { 0 pt } { 0 pt }
    \coffin_offset_corners:Nnn #4
      { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
  }
\coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
\coffin_set_eq:NN #1 \l_coffin_aligned_coffin
}
\cs_generate_variant:Nn \coffin_join:NnnNnnnn { c , Nnnc , cnnc }
```

(*End definition for* `\coffin_join:NnnNnnnn` *and others. These functions are documented on page* 3.)

\coffin_attach:NnnNnnnn
\coffin_attach:cnnNnnnn
\coffin_attach:Nnncnnnn
\coffin_attach:cnncnnnn
\coffin_attach_mark:NnnNnnnn

A more simple version of the above, as it simply uses the size of the first coffin for the new one. This means that the work here is rather simplified compared to the above code. The function used when marking a position is hear also as it is similar but without the structure updates.

```
\cs_new_protected_nopar:Npn \coffin_attach:NnnNnnnn #1#2#3#4#5#6#7#8 {
  \coffin_align:NnnNnnnnN
    #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
  \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
  \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
  \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
  \coffin_reset_structure:N \l_coffin_aligned_coffin
  \prop_set_eq:cc
    { l_coffin_corners_ \tex_number:D \l_coffin_aligned_coffin _prop }
    { l_coffin_corners_ \tex_number:D #1 _prop }
  \coffin_update_poles:N    \l_coffin_aligned_coffin
  \coffin_offset_poles:Nnn #1 { 0 pt } { 0 pt }
  \coffin_offset_poles:Nnn #4
    { \l_coffin_offset_x_dim } { \l_coffin_offset_y_dim }
  \coffin_update_vertical_poles:NNN #1 #4 \l_coffin_aligned_coffin
  \coffin_set_eq:NN #1 \l_coffin_aligned_coffin
}
\cs_new_protected_nopar:Npn
  \coffin_attach_mark:NnnNnnnn #1#2#3#4#5#6#7#8 {
  \coffin_align:NnnNnnnnN
    #1 {#2} {#3} #4 {#5} {#6} {#7} {#8} \l_coffin_aligned_coffin
  \box_set_ht:Nn \l_coffin_aligned_coffin { \box_ht:N #1 }
  \box_set_dp:Nn \l_coffin_aligned_coffin { \box_dp:N #1 }
  \box_set_wd:Nn \l_coffin_aligned_coffin { \box_wd:N #1 }
  \box_set_eq:NN #1 \l_coffin_aligned_coffin
}
\cs_generate_variant:Nn \coffin_attach:NnnNnnnn { c , Nnnc , cnnc }
```

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

\coffin_align:NnnNnnnnN

The internal function aligns the two coffins into a third one, but performs no corrections on the resulting coffin poles. The process begins by finding the points of intersection for the poles for each of the input coffins. Those for the first coffin are worked out after those for the second coffin, as this allows the 'primed' storage area to be used for the second coffin. The 'real' box offsets are then calculated, before using these to re-box the input coffins. The default poles are then set up, but the final result will depend on how the bounding box is being handled.

```
\cs_new_protected_nopar:Npn
  \coffin_align:NnnNnnnnN #1#2#3#4#5#6#7#8#9 {
  \coffin_calculate_intersection:Nnn #4 {#5} {#6}
  \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
  \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
  \coffin_calculate_intersection:Nnn #1 {#2} {#3}
```

```
  \dim_set:Nn \l_coffin_offset_x_dim
    { \l_coffin_x_dim - \l_coffin_x_prime_dim + #7 }
  \dim_set:Nn \l_coffin_offset_y_dim
    { \l_coffin_y_dim - \l_coffin_y_prime_dim + #8 }
  \hbox_set:Nn \l_coffin_aligned_internal_coffin
    {
      \box_use:N #1
      \tex_kern:D -\box_wd:N #1
      \tex_kern:D \l_coffin_offset_x_dim
      \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #4 }
    }
  \coffin_set_eq:NN #9 \l_coffin_aligned_internal_coffin
}
```

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

`\coffin_offset_poles:Nnn`
`\coffin_offset_pole:Nnnnnnn`

Transferring structures from one coffin to another requires that the positions are updated by the offset between the two coffins. This is done by mapping to the property list of the source coffins, moving as appropriate and saving to the new coffin data structures. The test for a `-` means that the structures from the parent coffins are uniquely labelled and do not depend on the order of alignment. The pay off for this is that `-` should not be used in coffin pole or handle names, and that multiple alignments do not result in a whole set of values.

```
\cs_new_protected_nopar:Npn \coffin_offset_poles:Nnn #1#2#3 {
  \prop_map_inline:cn { l_coffin_poles_ \tex_number:D #1 _prop }
    { \coffin_offset_pole:Nnnnnnn #1 {##1} ##2 {#2} {#3} }
}
\cs_new_protected_nopar:Npn
  \coffin_offset_pole:Nnnnnnn #1#2#3#4#5#6#7#8 {
  \dim_set:Nn \l_coffin_x_dim { #3 + #7 }
  \dim_set:Nn \l_coffin_y_dim { #4 + #8 }
  \tl_if_in:nnTF {#2} { - }
    { \tl_set:Nn \l_coffin_tmp_tl { {#2} } }
    { \tl_set:Nn \l_coffin_tmp_tl { { #1 - #2 } } }
  \exp_last_unbraced:NNo \coffin_set_pole:Nnx \l_coffin_aligned_coffin
    { \l_coffin_tmp_tl }
    {
      { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
      {#5} {#6}
    }
}
```

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

`\coffin_offset_corners:Nnn`
`\coffin_offset_corners:Nnnnn`

Saving the offset corners of a coffin is very similar, except that there is no need to worry about naming: every corner can be saved here as order is unimportant.

```
\cs_new_protected_nopar:Npn \coffin_offset_corners:Nnn #1#2#3 {
  \prop_map_inline:cn { l_coffin_corners_ \tex_number:D #1 _prop }
```

```
    { \coffin_offset_corner:Nnnnn #1 {##1} ##2 {#2} {#3} }
}
\cs_new_protected_nopar:Npn \coffin_offset_corner:Nnnnn #1#2#3#4#5#6 {
  \prop_put:cnx
    { l_coffin_corners_ \tex_number:D \l_coffin_aligned_coffin _prop }
    { #1 - #2 }
    {
      { \dim_eval:n { #3 + #5 } }
      { \dim_eval:n { #4 + #6 } }
    }
}
```

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

`\coffin_update_vertical_poles:NNN`
`\coffin_update_T:nnnnnnnnN`
`\coffin_update_B:nnnnnnnnN`

The `T` and `B` poles will need to be recalculated after alignment. These functions find the larger absolute value for the poles, but this is of course only logical when the poles are horizontal.

```
\cs_new_protected_nopar:Npn \coffin_update_vertical_poles:NNN #1#2#3 {
  \coffin_get_pole:NnN #3 { #1 -T } \l_coffin_pole_a_tl
  \coffin_get_pole:NnN #3 { #2 -T } \l_coffin_pole_b_tl
  \exp_last_two_unbraced:Noo \coffin_update_T:nnnnnnnnN
    \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
  \coffin_get_pole:NnN #3 { #1 -B } \l_coffin_pole_a_tl
  \coffin_get_pole:NnN #3 { #2 -B } \l_coffin_pole_b_tl
  \exp_last_two_unbraced:Noo \coffin_update_B:nnnnnnnnN
    \l_coffin_pole_a_tl \l_coffin_pole_b_tl #3
}
\cs_new_protected_nopar:Npn
  \coffin_update_T:nnnnnnnnN #1#2#3#4#5#6#7#8#9 {
  \dim_compare:nNnTF {#2} < {#6}
    {
      \coffin_set_pole:Nnx #9 { T }
        { { 0 pt } {#6} { 1000 pt } { 0 pt } }
    }
    {
      \coffin_set_pole:Nnx #9 { T }
        { { 0 pt } {#2} { 1000 pt } { 0 pt } }
    }
}
\cs_new_protected_nopar:Npn
  \coffin_update_B:nnnnnnnnN #1#2#3#4#5#6#7#8#9 {
  \dim_compare:nNnTF {#2} < {#6}
    {
      \coffin_set_pole:Nnx #9 { B }
        { { 0 pt } {#2}  { 1000 pt } { 0 pt } }
    }
    {
      \coffin_set_pole:Nnx #9 { B }
        { { 0 pt } {#6} { 1000 pt } { 0 pt } }
```

```
    }
}
```

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

`\coffin_typeset:Nnnnn`
`\coffin_typeset:cnnnn`

Typesetting a coffin means aligning it with the current position, which is done using a coffin with no content at all. This is done using the same approach as `\coffin_align:NnnNnnnnN` but without the offset corrections (which would be thrown away). The same is true for overlaying coffins, which uses the known size of an empty box!

```
\cs_new_protected_nopar:Npn \coffin_typeset:Nnnnn #1#2#3#4#5 {
  \coffin_align:NnnNnnnnN \c_empty_coffin { H } { l }
    #1 {#2} {#3} {#4} {#5} \l_coffin_aligned_coffin
  \hbox_set:Nn \l_coffin_aligned_coffin
    {
      \dim_compare:nNnT { \l_coffin_offset_x_dim } < { \c_zero_dim }
        { \tex_kern:D -\l_coffin_offset_x_dim }
      \hbox_unpack:N \l_coffin_aligned_coffin
      \dim_set:Nn \l_coffin_tmp_dim
        { \l_coffin_offset_x_dim + \box_wd:N #1 }
      \dim_compare:nNnT { \l_coffin_tmp_dim } < { \c_zero_dim }
        { \tex_kern:D -\l_coffin_tmp_dim }
    }
  \hbox_unpack:N \c_empty_box
  \box_use:N \l_coffin_aligned_coffin
}
\cs_generate_variant:Nn \coffin_typeset:Nnnnn { c }
```

(*End definition for* `\coffin_typeset:Nnnnn`*,* `\coffin_typeset:cnnnn`*, and . These functions are documented on page* 4.)

3.6 Rotating coffins

`\l_coffin_bounding_prop`

A property list for the bounding box of a coffin. This is only needed during the rotation, so there is just the one.

```
\prop_new:N \l_coffin_bounding_prop
```

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

`\l_coffin_bounding_shift_dim`

The shift of the bounding box of a coffin from the real content.

```
\dim_new:N \l_coffin_bounding_shift_dim
```

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

`\l_coffin_left_corner_dim`
`\l_coffin_right_corner_dim`
`\l_coffin_bottom_corner_dim`
`\l_coffin_top_corner_dim`

These are used to hold maxima for the various corner values: these thus define the minimum size of the bounding box after rotation.

```
\dim_new:N \l_coffin_left_corner_dim
\dim_new:N \l_coffin_right_corner_dim
\dim_new:N \l_coffin_bottom_corner_dim
\dim_new:N \l_coffin_top_corner_dim
```

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

`\coffin_rotate:Nn`
`\coffin_rotate:cn`

Rotating a coffin requires several steps which can be conveniently run together. The first step is to convert the angle given in degrees to one in radians. This is then used to set `\l_coffin_sin_fp` and `\l_coffin_cos_fp`, which are carried through unchanged for the rest of the procedure.

```
\cs_new_protected_nopar:Npn \coffin_rotate:Nn #1#2 {
  \fp_set:Nn \l_coffin_tmp_fp {#2}
  \fp_div:Nn \l_coffin_tmp_fp { 180 }
  \fp_mul:Nn \l_coffin_tmp_fp { \c_pi_fp }
  \fp_sin:Nn \l_coffin_sin_fp { \l_coffin_tmp_fp }
  \fp_cos:Nn \l_coffin_cos_fp { \l_coffin_tmp_fp }
```

The corners and poles of the coffin can now be rotated around the origin. This is best achieved using mapping functions.

```
  \prop_map_inline:cn { l_coffin_corners_ \tex_number:D #1 _prop }
    { \coffin_rotate_corner:Nnnn #1 {##1} ##2 }
  \prop_map_inline:cn { l_coffin_poles_ \tex_number:D #1 _prop }
    { \coffin_rotate_pole:Nnnnnn #1 {##1} ##2 }
```

The bounding box of the coffin needs to be rotated, and to do this the corners have to be found first. They are then rotated in the same way as the corners of the coffin material itself.

```
  \coffin_set_bounding:N #1
  \prop_map_inline:Nn \l_coffin_bounding_prop
    { \coffin_rotate_bounding:nnn {##1} ##2 }
```

At this stage, there needs to be a calculation to find where the corners of the content and the box itself will end up.

```
  \coffin_find_corner_maxima:N #1
  \coffin_find_bounding_shift:
  \hbox_set:Nn #1 { \rotatebox {#2} { \box_use:N #1 } }
```

The correction of the box position itself takes place here. The idea is that the bounding box for a coffin is tight up to the content, and has the reference point at the bottom-left. The x-direction is handled by moving the content by the difference in the positions of the bounding box and the content left edge. The y-direction is dealt with by moving the box down by any depth it has acquired.

```
  \hbox_set:Nn #1
    {
```

```
      \tex_kern:D \l_coffin_bounding_shift_dim
      \tex_kern:D -\l_coffin_left_corner_dim
      \box_move_down:nn { \l_coffin_bottom_corner_dim }
        { \box_use:N #1 }
    }
```

If there have been any previous rotations then the size of the bounding box will be bigger than the contents. This can be corrected easily by setting the size of the box to the height and width of the content.

```
  \box_set_ht:Nn #1
    { \l_coffin_top_corner_dim - \l_coffin_bottom_corner_dim }
  \box_set_dp:Nn #1 { 0 pt }
  \box_set_wd:Nn #1
    { \l_coffin_right_corner_dim - \l_coffin_left_corner_dim }
```

The final task is to move the poles and corners such that they are back in alignment with the box reference point.

```
  \prop_map_inline:cn { l_coffin_corners_ \tex_number:D #1 _prop }
    { \coffin_shift_corner:Nnnn #1 {##1} ##2 }
  \prop_map_inline:cn { l_coffin_poles_ \tex_number:D #1 _prop }
    { \coffin_shift_pole:Nnnnnn #1 {##1} ##2 }
}
\cs_generate_variant:Nn \coffin_rotate:Nn { c }
```

(*End definition for* `\coffin_rotate:Nn`*,* `\coffin_rotate:cn`*, and . These functions are documented on page 3.*)

`\coffin_set_bounding:N` The bounding box corners for a coffin are easy enough to find: this is the same code as for the corners of the material itself, but using a dedicated property list.

```
\cs_new_protected_nopar:Npn \coffin_set_bounding:N #1 {
  \prop_put:Nnx \l_coffin_bounding_prop { tl }
    { { 0 pt } { \dim_use:N \box_ht:N #1 } }
  \prop_put:Nnx \l_coffin_bounding_prop { tr }
    { { \dim_use:N \box_wd:N #1 } { \dim_use:N \box_ht:N #1 } }
  \dim_set:Nn \l_coffin_tmp_dim { - \dim_use:N \box_dp:N #1 }
  \prop_put:Nnx \l_coffin_bounding_prop { bl }
    { { 0 pt } { \dim_use:N \l_coffin_tmp_dim } }
  \prop_put:Nnx \l_coffin_bounding_prop { br }
    { { \dim_use:N \box_wd:N #1 } { \dim_use:N \l_coffin_tmp_dim } }
}
```

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

`\coffin_rotate_bounding:nnn` `\coffin_rotate_corner:Nnnn` Rotating the position of the corner of the coffin is just a case of treating this as a vector from the reference point. The same treatment is used for the corners of the material itself and the bounding box.

```
\cs_new_protected_nopar:Npn \coffin_rotate_bounding:nnn #1#2#3 {
  \coffin_rotate_vector:nnNN {#2} {#3} \l_coffin_x_dim \l_coffin_y_dim
  \prop_put:Nnx \l_coffin_bounding_prop {#1}
    { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
}
\cs_new_protected_nopar:Npn \coffin_rotate_corner:Nnnn #1#2#3#4 {
  \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
  \prop_put:cnx { l_coffin_corners_ \tex_number:D #1 _prop } {#2}
    { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
}
```

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

`\coffin_rotate_pole:Nnnnnn` Rotating a single pole simply means shifting the co-ordinate of the pole and its direction. The rotation here is about the bottom-left corner of the coffin.

```
\cs_new_protected_nopar:Npn \coffin_rotate_pole:Nnnnnn #1#2#3#4#5#6 {
  \coffin_rotate_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
  \coffin_rotate_vector:nnNN {#5} {#6}
    \l_coffin_x_prime_dim \l_coffin_y_prime_dim
  \coffin_set_pole:Nnx #1 {#2}
    {
      { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
      { \dim_use:N \l_coffin_x_prime_dim }
      { \dim_use:N \l_coffin_y_prime_dim }
    }
}
```

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

`\coffin_rotate_vector:nnNN` A rotation function, which needs only an input vector (as dimensions) and an output space. The values `\l_coffin_cos_fp` and `\l_coffin_sin_fp` should previously have been set up correctly. Working this way means that the floating point work is kept to a minimum: for any given rotation the sin and cosine values do no change, after all.

```
\cs_new_protected_nopar:Npn \coffin_rotate_vector:nnNN #1#2#3#4 {
  \fp_set_from_dim:Nn \l_coffin_x_fp {#1}
  \fp_set_from_dim:Nn \l_coffin_y_fp {#2}
  \fp_set_eq:NN \l_coffin_x_prime_fp \l_coffin_x_fp
  \fp_set_eq:NN \l_coffin_tmp_fp     \l_coffin_y_fp
  \fp_mul:Nn \l_coffin_x_prime_fp { \l_coffin_cos_fp }
  \fp_mul:Nn \l_coffin_tmp_fp     { \l_coffin_sin_fp }
  \fp_sub:Nn \l_coffin_x_prime_fp { \l_coffin_tmp_fp }
  \fp_set_eq:NN \l_coffin_y_prime_fp \l_coffin_y_fp
  \fp_set_eq:NN \l_coffin_tmp_fp     \l_coffin_x_fp
  \fp_mul:Nn \l_coffin_y_prime_fp { \l_coffin_cos_fp }
  \fp_mul:Nn \l_coffin_tmp_fp     { \l_coffin_sin_fp }
  \fp_add:Nn \l_coffin_y_prime_fp { \l_coffin_tmp_fp }
  \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_x_prime_fp }
  \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_y_prime_fp }
}
```

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

`\coffin_find_corner_maxima:N`
`\coffin_find_corner_maxima_aux:nn`

The idea here is to find the extremities of the content of the coffin. This is done by looking for the smallest values for the bottom and left corners, and the largest values for the top and right corners. The values start at the maximum dimensions so that the case where all are positive or all are negative works out correctly.

```
\cs_new_protected_nopar:Npn \coffin_find_corner_maxima:N #1 {
  \dim_set:Nn \l_coffin_top_corner_dim   { -\c_max_dim }
  \dim_set:Nn \l_coffin_right_corner_dim { -\c_max_dim }
  \dim_set:Nn \l_coffin_bottom_corner_dim { \c_max_dim }
  \dim_set:Nn \l_coffin_left_corner_dim   { \c_max_dim }
  \prop_map_inline:cn { l_coffin_corners_ \tex_number:D #1 _prop }
    { \coffin_find_corner_maxima_aux:nn ##2 }
}
\cs_new_protected_nopar:Npn \coffin_find_corner_maxima_aux:nn #1#2 {
  \dim_set_min:Nn \l_coffin_left_corner_dim   {#1}
  \dim_set_max:Nn \l_coffin_right_corner_dim  {#1}
  \dim_set_min:Nn \l_coffin_bottom_corner_dim {#2}
  \dim_set_max:Nn \l_coffin_top_corner_dim    {#2}
}
```

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

`\coffin_find_bounding_shift:`
`\coffin_find_bounding_shift_aux:nn`

The approach to finding the shift for the bounding box is similar to that for the corners. However, there is only one value needed here and a fixed input property list, so things are a bit clearer.

```
\cs_new_protected_nopar:Npn \coffin_find_bounding_shift: {
  \dim_set:Nn \l_coffin_bounding_shift_dim { \c_max_dim }
  \prop_map_inline:Nn \l_coffin_bounding_prop
    { \coffin_find_bounding_shift_aux:nn ##2 }
}
\cs_new_protected_nopar:Npn \coffin_find_bounding_shift_aux:nn #1#2 {
  \dim_set_min:Nn \l_coffin_bounding_shift_dim {#1}
}
```

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

`\coffin_shift_corner:Nnnn`
`\coffin_shift_pole:Nnnnnn`

Shifting the corners and poles of a coffin means subtracting the appropriate values from the x- and y-components. For the poles, this means that the direction vector is unchanged.

```
\cs_new_protected_nopar:Npn \coffin_shift_corner:Nnnn #1#2#3#4 {
  \prop_put:cnx { l_coffin_corners_ \tex_number:D #1 _ prop } {#2}
    {
      { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
      { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
    }
```

```
}
\cs_new_protected_nopar:Npn \coffin_shift_pole:Nnnnnn #1#2#3#4#5#6 {
  \prop_put:cnx { l_coffin_poles_ \tex_number:D #1 _ prop } {#2}
    {
      { \dim_eval:n { #3 - \l_coffin_left_corner_dim } }
      { \dim_eval:n { #4 - \l_coffin_bottom_corner_dim } }
      {#5} {#6}
    }
}
```

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

3.7 Resizing coffins

`\l_coffin_scale_x_fp`
`\l_coffin_scale_y_fp`

Storage for the scaling factors in x and y, respectively.

```
\fp_new:N \l_coffin_scale_x_fp
\fp_new:N \l_coffin_scale_y_fp
```

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

`\l_coffin_scaled_total_height_dim`
`\l_coffin_scaled_width_dim`

When scaling, the values given have to be turned into absolute values.

```
\dim_new:N \l_coffin_scaled_total_height_dim
\dim_new:N \l_coffin_scaled_width_dim
```

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

`\coffin_resize:Nnn`
`\coffin_resize:cnn`

Resizing a coffin begins by setting up the user-friendly names for the dimensions of the coffin box. The new sizes are then turned into scale factor. This is the same operation as takes place for the underlying box, but that operation is grouped and so the same calculation is done here.

```
\cs_new_protected_nopar:Npn \coffin_resize:Nnn #1#2#3 {
  \coffin_set_user_dimensions:N #1
  \fp_set_from_dim:Nn \l_coffin_scale_x_fp {#2}
  \fp_set_from_dim:Nn \l_coffin_tmp_fp { \Width }
  \fp_div:Nn \l_coffin_scale_x_fp { \l_coffin_tmp_fp }
  \fp_set_from_dim:Nn \l_coffin_scale_y_fp {#3}
  \fp_set_from_dim:Nn \l_coffin_tmp_fp { \TotalHeight }
  \fp_div:Nn \l_coffin_scale_y_fp { \l_coffin_tmp_fp }
  \hbox_set:Nn #1 { \resizebox * {#2} {#3} { \box_use:N #1 } }
  \coffin_resize_common:Nnn #1 {#2} {#3}
}
\cs_generate_variant:Nn \coffin_resize:Nnn { c }
```

(*End definition for* `\coffin_resize:Nnn`*,* `\coffin_resize:cnn`*, and . These functions are documented on page* ??.)

\coffin_resize_common:Nnn The poles and corners of the coffin are scaled to the appropriate places before actually resizing the underlying box.

```
\cs_new_protected_nopar:Npn \coffin_resize_common:Nnn #1#2#3 {
  \prop_map_inline:cn { l_coffin_corners_ \tex_number:D #1 _prop }
    { \coffin_scale_corner:Nnnn #1 {##1} ##2 }
  \prop_map_inline:cn { l_coffin_poles_ \tex_number:D #1 _prop }
    { \coffin_scale_pole:Nnnnnn #1 {##1} ##2 }
```

Negative x-scaling values will place the poles in the wrong location: this is corrected here.

```
  \fp_compare:NNNT \l_coffin_scale_x_fp < \c_zero_fp
    {
      \prop_map_inline:cn { l_coffin_corners_ \tex_number:D #1 _prop }
        { \coffin_x_shift_corner:Nnnn #1 {##1} ##2 }
      \prop_map_inline:cn { l_coffin_poles_ \tex_number:D #1 _prop }
        { \coffin_x_shift_pole:Nnnnnn #1 {##1} ##2 }
    }
  \coffin_end_user_dimensions:
}
```

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

\coffin_scale:Nnn
\coffin_scale:cnn For scaling, the opposite calculation is done to find the new dimensions for the coffin. Only the total height is needed, as this is the shift required for corners and poles. The scaling is done the TEX way as this works properly with floating point values without needing to use the `fp` module.

```
\cs_new_protected_nopar:Npn \coffin_scale:Nnn #1#2#3 {
  \coffin_set_user_dimensions:N #1
  \fp_set:Nn \l_coffin_scale_x_fp {#2}
  \fp_set:Nn \l_coffin_scale_y_fp {#3}
  \fp_compare:NNNTF \l_coffin_scale_y_fp > \c_zero_fp
    { \l_coffin_scaled_total_height_dim #3 \TotalHeight }
    { \l_coffin_scaled_total_height_dim -#3 \TotalHeight }
  \fp_compare:NNNTF \l_coffin_scale_x_fp > \c_zero_fp
    { \l_coffin_scaled_width_dim -#2 \Width }
    { \l_coffin_scaled_width_dim #2 \Width }
  \hbox_set:Nn #1 { \scalebox  {#2} [#3] { \box_use:N #1 } }
  \coffin_resize_common:Nnn #1
    { \l_coffin_scaled_width_dim } { \l_coffin_scaled_total_height_dim }
}
\cs_generate_variant:Nn \coffin_scale:Nnn { c }
```

(*End definition for* `\coffin_scale:Nnn`*,* `\coffin_scale:cnn`*, and . These functions are documented on page* ??.)

\coffin_scale_vector:nnNN This functions scales a vector from the origin using the pre-set scale factors in x and y. This is a much less complex operation than rotation, and as a result the code is a lot clearer.

```
\cs_new_protected_nopar:Npn \coffin_scale_vector:nnNN #1#2#3#4 {
  \fp_set_from_dim:Nn \l_coffin_tmp_fp {#1}
  \fp_mul:Nn \l_coffin_tmp_fp { \l_coffin_scale_x_fp }
  \dim_set:Nn #3 { \fp_to_dim:N \l_coffin_tmp_fp }
  \fp_set_from_dim:Nn \l_coffin_tmp_fp {#2}
  \fp_mul:Nn \l_coffin_tmp_fp { \l_coffin_scale_y_fp }
  \dim_set:Nn #4 { \fp_to_dim:N \l_coffin_tmp_fp }
}
```

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

`\coffin_scale_corner:Nnnn`
`\coffin_scale_pole:Nnnnnn`

Scaling both corners and poles is a simple calculation using the preceding vector scaling.

```
\cs_new_protected_nopar:Npn \coffin_scale_corner:Nnnn #1#2#3#4 {
  \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
  \prop_put:cnx { l_coffin_corners_ \tex_number:D #1 _prop } {#2}
    { { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim } }
}
\cs_new_protected_nopar:Npn \coffin_scale_pole:Nnnnnn #1#2#3#4#5#6 {
  \coffin_scale_vector:nnNN {#3} {#4} \l_coffin_x_dim \l_coffin_y_dim
  \coffin_set_pole:Nnx #1 {#2}
    {
      { \dim_use:N \l_coffin_x_dim } { \dim_use:N \l_coffin_y_dim }
      {#5} {#6}
    }
}
```

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

`\coffin_x_shift_corner:Nnnn`
`\coffin_x_shift_pole:Nnnnnn`

These functions correct for the x displacement that takes place with a negative horizontal scaling.

```
\cs_new_protected_nopar:Npn \coffin_x_shift_corner:Nnnn #1#2#3#4 {
  \prop_put:cnx { l_coffin_corners_ \tex_number:D #1 _prop } {#2}
    {
      { \the_tex:D \etex_dimexpr:D #3 + \box_wd:N #1 \scan_stop: } {#4}
    }
}
\cs_new_protected_nopar:Npn \coffin_x_shift_pole:Nnnnnn #1#2#3#4#5#6 {
  \prop_put:cnx { l_coffin_poles_ \tex_number:D #1 _prop } {#2}
    {
      { \the_tex:D \etex_dimexpr:D #3 + \box_wd:N #1 \scan_stop: } {#4}
      {#5} {#6}
    }
}
```

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

3.8 Coffin diagnostics

\l_coffin_display_coffin
\l_coffin_display_coord_coffin
\l_coffin_display_pole_coffin

Used for printing coffins with data structures attached.

```
\coffin_new:N \l_coffin_display_coffin
\coffin_new:N \l_coffin_display_coord_coffin
\coffin_new:N \l_coffin_display_pole_coffin
```

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

\l_coffin_display_handles_prop

This property list is used to print coffin handles at suitable positions. The offsets are expressed as multiples of the basic offset value, which therefore acts as a scale-factor.

```
\prop_new:N \l_coffin_display_handles_prop
\prop_put:Nnn \l_coffin_display_handles_prop { tl }
  { { b } { r } { -1 } { 1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { thc }
  { { b } { hc } { 0 } { 1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { tr }
  { { b } { l } { 1 } { 1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { vcl }
  { { vc } { r } { -1 } { 0 } }
\prop_put:Nnn \l_coffin_display_handles_prop { vchc }
  { { vc } { hc } { 0 } { 0 } }
\prop_put:Nnn \l_coffin_display_handles_prop { vcr }
  { { vc } { l } { 1 } { 0 } }
\prop_put:Nnn \l_coffin_display_handles_prop { bl }
  { { t } { r } { -1 } { -1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { bhc }
  { { t } { hc } { 0 } { -1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { br }
  { { t } { l } { 1 } { -1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Tl }
  { { t } { r } { -1 } { -1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Thc }
  { { t } { hc } { 0 } { -1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Tr }
  { { t } { l } { 1 } { -1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Hl }
  { { vc } { r } { -1 } { 1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Hhc }
  { { vc } { hc } { 0 } { 1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Hr }
  { { vc } { l } { 1 } { 1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Bl }
  { { b } { r } { -1 } { -1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Bhc }
  { { b } { hc } { 0 } { -1 } }
\prop_put:Nnn \l_coffin_display_handles_prop { Br }
  { { b } { l } { 1 } { -1 } }
```

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

`\l_coffin_display_offset_dim` The standard offset for the label from the handle position when displaying handles.

```
\dim_new:N  \l_coffin_display_offset_dim
\dim_set:Nn \l_coffin_display_offset_dim { 2 pt }
```

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

`\l_coffin_display_x_dim`
`\l_coffin_display_y_dim` As the intersections of poles have to be calculated to find which ones to print, there is a need to avoid repetition. This is done by saving the intersection into two dedicated values.

```
\dim_new:N \l_coffin_display_x_dim
\dim_new:N \l_coffin_display_y_dim
```

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

`\l_coffin_display_poles_prop` A property list for printing poles: various things need to be deleted from this to get a 'nice' output.

```
\prop_new:N \l_coffin_display_poles_prop
```

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

`\l_coffin_display_font_tl` Stores the settings used to print coffin data: this keeps things flexible.

```
\tl_new:N  \l_coffin_display_font_tl
\tl_set:Nn \l_coffin_display_font_tl { \sffamily \tiny }
```

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

`\l_coffin_show_toks` For the package the LaTeX 2_ε code can be used.

```
\cs_new_eq:NN \l_coffin_show_toks \toks@
```

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

`\l_coffin_handles_tmp_prop` Used for displaying coffins, as the handles need to be stored in this case, at least temporarily.

```
\prop_new:N \l_coffin_handles_tmp_prop
```

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

`\coffin_mark_handle:Nnnn`
`\coffin_mark_handle:cnnn`
`\coffin_mark_handle_aux:nnnnNnn` Marking a single handle is relatively easy. The standard attachment function is used, meaning that there are two calculations for the location. However, this is likely to be okay given the load expected. Contrast with the more optimised version for showing all handles which comes next.

```
\cs_new_protected_nopar:Npn \coffin_mark_handle:Nnnn #1#2#3#4 {
  \hcoffin_set:Nn \l_coffin_display_pole_coffin
    {
      \color {#4}
      \rule { 1 pt } { 1 pt }
      \hbox:n { \tex_vrule:D width 1 pt height 1 pt \scan_stop: } % TEMP
    }
  \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
    \l_coffin_display_pole_coffin { hc } { vc } { 0 pt } { 0 pt }
  \hcoffin_set:Nn \l_coffin_display_coord_coffin
    {
      \color {#4}
      \l_coffin_display_font_tl
      ( \tl_to_str:n { #2 , #3 } )
    }
  \prop_get:NnN \l_coffin_display_handles_prop
    { #2 #3 } \l_coffin_tmp_tl
  \quark_if_no_value:NTF \l_coffin_tmp_tl
    {
      \prop_get:NnN \l_coffin_display_handles_prop
        { #3 #2 } \l_coffin_tmp_tl
      \quark_if_no_value:NTF \l_coffin_tmp_tl
        {
          \coffin_attach_mark:NnnNnnnn #1 {#2} {#3}
            \l_coffin_display_coord_coffin { l } { vc }
              { 1 pt } { 0 pt }
        }
        {
          \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
            \l_coffin_tmp_tl #1 {#2} {#3}
        }
    }
    {
      \exp_last_unbraced:No \coffin_mark_handle_aux:nnnnNnn
        \l_coffin_tmp_tl #1 {#2} {#3}
    }
}
\cs_new_protected_nopar:Npn
  \coffin_mark_handle_aux:nnnnNnn #1#2#3#4#5#6#7 {
  \coffin_attach_mark:NnnNnnnn #5 {#6} {#7}
    \l_coffin_display_coord_coffin {#1} {#2}
    { #3 \l_coffin_display_offset_dim }
    { #4 \l_coffin_display_offset_dim }
}
\cs_generate_variant:Nn \coffin_mark_handle:Nnnn { c }
```

(*End definition for* `\coffin_mark_handle:Nnnn`, `\coffin_mark_handle:cnnn`, *and* . *These functions are documented on page* 4.)

`\coffin_display_handles:Nn`
`\coffin_display_handles:cn`
`\coffin_display_handles_aux:nnnnnn`
`\coffin_display_handles_aux:nnnn`
`\coffin_display_attach:Nnnnn`

Printing the poles starts by removing any duplicates, for which the `H` poles is used as

the definitive version for the baseline and bottom. Two loops are then used to find the combinations of handles for all of these poles. This is done such that poles are removed during the loops to avoid duplication.

```
\cs_new_protected_nopar:Npn \coffin_display_handles:Nn #1#2 {
  \hcoffin_set:Nn \l_coffin_display_pole_coffin
    {
      \color {#2}
      \rule { 1 pt } { 1 pt }
    }
  \prop_set_eq:Nc \l_coffin_display_poles_prop
    { l_coffin_poles_ \tex_number:D #1 _prop }
  \coffin_get_pole:NnN #1 { H } \l_coffin_pole_a_tl
  \coffin_get_pole:NnN #1 { T } \l_coffin_pole_b_tl
  \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
    { \prop_del:Nn \l_coffin_display_poles_prop { T } }
  \coffin_get_pole:NnN #1 { B } \l_coffin_pole_b_tl
  \tl_if_eq:NNT \l_coffin_pole_a_tl \l_coffin_pole_b_tl
    { \prop_del:Nn \l_coffin_display_poles_prop { B } }
  \coffin_set_eq:NN \l_coffin_display_coffin #1
  \prop_clear:N \l_coffin_handles_tmp_prop
  \prop_map_inline:Nn \l_coffin_display_poles_prop
    {
      \prop_del:Nn \l_coffin_display_poles_prop {##1}
      \coffin_display_handles_aux:nnnnnn {##1} ##2 {#2}
    }
  \box_use:N \l_coffin_display_coffin
}
```

For each pole there is a check for an intersection, which here does not give an error if none is found. The successful values are stored and used to align the pole coffin with the main coffin for output. The positions are recovered from the preset list if available.

```
\cs_new_protected_nopar:Npn
  \coffin_display_handles_aux:nnnnnn #1#2#3#4#5#6 {
  \prop_map_inline:Nn \l_coffin_display_poles_prop
    {
      \bool_set_false:N \l_coffin_error_bool
      \coffin_calculate_intersection:nnnnnnnn {#2} {#3} {#4} {#5} ##2
      \bool_if:NF \l_coffin_error_bool
        {
          \dim_set:Nn \l_coffin_display_x_dim { \l_coffin_x_dim }
          \dim_set:Nn \l_coffin_display_y_dim { \l_coffin_y_dim }
          \coffin_display_attach:Nnnnn
            \l_coffin_display_pole_coffin { hc } { vc }
            { 0 pt } { 0 pt }
          \hcoffin_set:Nn \l_coffin_display_coord_coffin
            {
              \color {#6}
              \l_coffin_display_font_tl
```

```
              ( \tl_to_str:n { #1 , ##1 } )
            }
          \prop_get:NnN \l_coffin_display_handles_prop
            { #1 ##1 } \l_coffin_tmp_tl
          \quark_if_no_value:NTF \l_coffin_tmp_tl
            {
              \prop_get:NnN \l_coffin_display_handles_prop
                { ##1 #1 } \l_coffin_tmp_tl
              \quark_if_no_value:NTF \l_coffin_tmp_tl
                {
                  \coffin_display_attach:Nnnnn
                    \l_coffin_display_coord_coffin { l } { vc }
                    { 1 pt } { 0 pt }
                }
                {
                  \exp_last_unbraced:No
                    \coffin_display_handles_aux:nnnn
                    \l_coffin_tmp_tl
                }
            }
            {
              \exp_last_unbraced:No \coffin_display_handles_aux:nnnn
                \l_coffin_tmp_tl
            }
        }
    }
}
\cs_new_protected_nopar:Npn \coffin_display_handles_aux:nnnn #1#2#3#4 {
  \coffin_display_attach:Nnnnn
    \l_coffin_display_coord_coffin {#1} {#2}
    { #3 \l_coffin_display_offset_dim }
    { #4 \l_coffin_display_offset_dim }
}
\cs_generate_variant:Nn \coffin_display_handles:Nn { c }
```

This is a dedicated version of `\coffin_attach:NnnNnnnn` with a hard-wired first coffin. As the intersection is already known and stored for the display coffin the code simply uses it directly, with no calculation.

```
\cs_new_protected_nopar:Npn \coffin_display_attach:Nnnnn #1#2#3#4#5 {
  \coffin_calculate_intersection:Nnn #1 {#2} {#3}
  \dim_set:Nn \l_coffin_x_prime_dim { \l_coffin_x_dim }
  \dim_set:Nn \l_coffin_y_prime_dim { \l_coffin_y_dim }
  \dim_set:Nn \l_coffin_offset_x_dim
    { \l_coffin_display_x_dim - \l_coffin_x_prime_dim + #4 }
  \dim_set:Nn \l_coffin_offset_y_dim
    { \l_coffin_display_y_dim - \l_coffin_y_prime_dim + #5 }
  \hbox_set:Nn \l_coffin_aligned_coffin
    {
      \box_use:N \l_coffin_display_coffin
```

```
      \tex_kern:D -\box_wd:N \l_coffin_display_coffin
      \tex_kern:D \l_coffin_offset_x_dim
      \box_move_up:nn { \l_coffin_offset_y_dim } { \box_use:N #1 }
    }
  \box_set_ht:Nn \l_coffin_aligned_coffin
    { \box_ht:N \l_coffin_display_coffin }
  \box_set_dp:Nn \l_coffin_aligned_coffin
    { \box_dp:N \l_coffin_display_coffin }
  \box_set_wd:Nn \l_coffin_aligned_coffin
    { \box_wd:N \l_coffin_display_coffin }
  \box_set_eq:NN \l_coffin_display_coffin \l_coffin_aligned_coffin
}
```

(*End definition for* `\coffin_display_handles:Nn`, `\coffin_display_handles:cn`, *and* . *These functions are documented on page* 4.)

`\coffin_show_structure:N`
`\coffin_show_structure:c`

For showing the various internal structures attached to a coffin in a way that keeps things relatively readable. If there is no apparent structure then the code complains.

```
\cs_new_protected_nopar:Npn \coffin_show_structure:N #1 {
  \toks_clear:N \l_coffin_show_toks
  \cs_if_exist:cTF { l_coffin_poles_ \tex_number:D #1 _prop }
    {
      \iow_term:x
        {
          \iow_newline:
          Size~of~coffin~\token_to_str:N #1 : \iow_newline:
          > ~ ht~=~\dim_use:N \box_ht:N #1 \iow_newline:
          > ~ dp~=~\dim_use:N \box_dp:N #1 \iow_newline:
          > ~ wd~=~\dim_use:N \box_wd:N #1 \iow_newline:
        }
      \iow_term:x { Poles~of~coffin~\token_to_str:N #1 : }
      \prop_map_inline:cn { l_coffin_poles_ \tex_number:D #1 _prop }
        {
          \toks_if_empty:NF \l_coffin_show_toks
            {
              \toks_put_right:Nx \l_coffin_show_toks
                { \iow_newline: > ~ }
            }
          \toks_put_right:Nx \l_coffin_show_toks
            { ~ \exp_not:n {##1} \c_space_tl => ~ \exp_not:n {##2} }
        }
    }
    {
      \iow_term:x { ----No~poles~found~--- }
      \toks_put_right:Nn \l_coffin_show_toks
        { Is~this~really~a~coffin? }
    }
  \toks_show:N \l_coffin_show_toks
}
```

```
\cs_generate_variant:Nn \coffin_show_structure:N { c }
```

(*End definition for* `\coffin_show_structure:N`*. This function is documented on page* 4.)

3.9 Messages

```
\msg_kernel_new:nnnn { coffin } { no-pole-intersection }
  { No~intersection~between~coffin~poles. }
  {
    \l_msg_coding_error_text_tl
    LaTeX~was~asked~to~find~the~intersection~between~two~poles,~
    but~they~do~not~have~a~unique~meeting~point:~
    the~value~(0~pt,~0~pt)~will~be~used.
  }
\msg_kernel_new:nnnn { coffin } { unknown-coffin }
  { Unknown~coffin~'#1'. }
  { The~coffin~'#1'~was~never~defined. }
\msg_kernel_new:nnnn { coffin } { unknown-coffin-pole }
  { Pole~'#1'~unknown~for~coffin~'#2'. }
  {
    \l_msg_coding_error_text_tl
    LaTeX~was~asked~to~find~a~typesetting~pole~for~a~coffin,~
    but~either~the~coffin~does~not~exist~or~the~pole~name~is~wrong.
  }
```

`\exp_two_last_unbraced:Noo`

```
\cs_new:Npn \exp_last_two_unbraced:Noo #1 #2 #3{
  \exp_after:wN \exp_after:wN \exp_after:wN #1 \exp_after:wN #2 #3
}
```

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

⟨/package⟩

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

C

D

E

F

G

H

I

L

M

P

Q

R

S

T

V

W