The LaTeX3 kernel: design-level 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. This module contains the design-level code to implement this scheme.

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

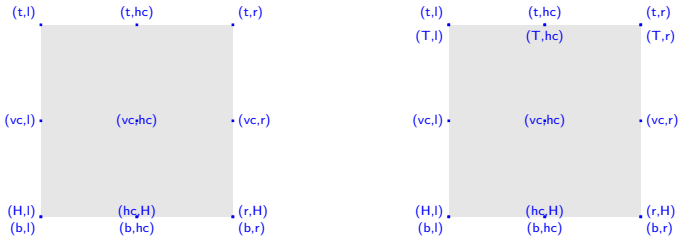

Figure 1: Standard coffin handles: left, horizontal coffin; right, vertical coffin

1 Introduction: the coffin concept

In LaTeX3 terminology, a 'coffin' is a box containing typeset material. Along with the box itself, the coffin structure includes information on the size and shape of the box, which makes it possible to align two or more coffins easily. This is achieved by providing a series of 'poles' for each coffin. These are horizontal and vertical lines through the coffin at defined positions, for example the top or horizontal centre. The points where these poles intersect are called 'handles'. Two coffins can then be aligned by describing the relationship between a handle on one coffin with a handle on the second. In words, an example might then read

> Align the top-left handle of coffin A with the bottom-right handle of coffin B.

The locations of coffin handles are much easier to understand visually. Figure 1 shows the standard handle positions for a coffin typeset in horizontal mode (left) and in vertical mode (right). Notice that the later case results in a greater number of handles being available. As illustrated, each handle results from the intersection of two poles. For example, the centre of the coffin is marked '(hc,vc), *i.e.* it is the point of intersection of the horizontal centre pole with the vertical centre pole. New handles are generated automatically when poles are added to a coffin: handles are 'dynamic' entities.

2 Creating and setting coffins

Before any alignment can take place, coffins must be created and their contents must be created. All coffin operations are local to the current TeX group with the exception

of coffin creation. Coffins are also 'colour safe': in contrast to the code-level `\box_...` functions there is no need to add additional grouping to coffins when dealing with colour.

`\NewCoffin` `\NewCoffin` ⟨*coffin*⟩
Before a ⟨*coffin*⟩ can be used, it must be allocated using `\NewCoffin`. The name of the ⟨*coffin*⟩ should be a control sequence (starting with the escape character, usually \), for example

```
\NewCoffin\MyCoffin
```

Coffins are allocated globally, and an error will be raised if the name of the ⟨*coffin*⟩ is not globally-unique.

`\SetHorizontalCoffin` `\SetHorizontalCoffin` ⟨*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.

`\SetVerticalCoffin` `\SetVerticalCoffin` ⟨*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.

3 Controlling coffin poles

A number of standard poles are automatically generated when the coffin is set or an alignment takes place. The standard poles for all coffins are:

- `l` a pole running along the left-hand edge of the bounding box of the coffin;
- `hc` a pole running vertically through the centre of the coffin half-way between the left- and right-hand edges of the bounding box (*i.e* the 'horizontal centre');
- `r` a pole running along the right-hand edge of the bounding box of the coffin;
- `b` a pole running along the bottom edge of the bounding box of the coffin;
- `vc` a pole running horizontally through the centre of the coffin half-way between the bottom and top edges of the bounding box (*i.e* the 'vertical centre');
- `t` a pole running along the top edge of the bounding box of the coffin;
- `H` a pole running along the baseline of the typeset material contained in the coffin.

In addition, coffins containing vertical-mode material also feature poles which reflect the richer nature of these systems:

B a pole running along the baseline of the material at the bottom of the coffin.

T a pole running along the baseline of the material at the top of the coffin.

`\SetHorizontalPole` `\SetHorizontalPole` ⟨*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 `\TotalHeight`, `\Height`, `\Depth` and `\Width`, which will evaluate to the appropriate dimensions of the ⟨*coffin*⟩. For example, to create a pole running horizontally through the coffin at one third of the distance from the base of the coffin to the top, the appropriate instruction would be

```
\SetHorizontalPole \MyCoffin {height/3} {\TotalHeight/3}
```

Note that poles which run *horizontally* are described in terms of their *vertical* location in the coffin. Also notice that the total height of the coffin is described by the sum of `\Height` and `\Depth`: these are both measured from the horizontal baseline of the material in the coffin.

`\SetVerticalPole` `\SetVerticalPole` ⟨*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*⟩. For example, to create a pole running vertically through the coffin at one third of the distance from the left-hand edge, the appropriate instruction would be

```
\SetVerticalPole \MyCoffin {width/3} {\Width/3}
```

Note that poles which run *vertically* are described in terms of their *horizontal* location in the coffin.

`\TotalHeight` `\TotalHeight`
Within the ⟨*offset*⟩ argument of `\SetHorizontalPole` and `\SetVerticalPole`, `\TotalHeight` will give the distance from the base to the top of the bounding box of the relevant coffin.

`\Height` `\Height`
Within the ⟨*offset*⟩ argument of `\SetHorizontalPole` and `\SetVerticalPole`, `\Height`

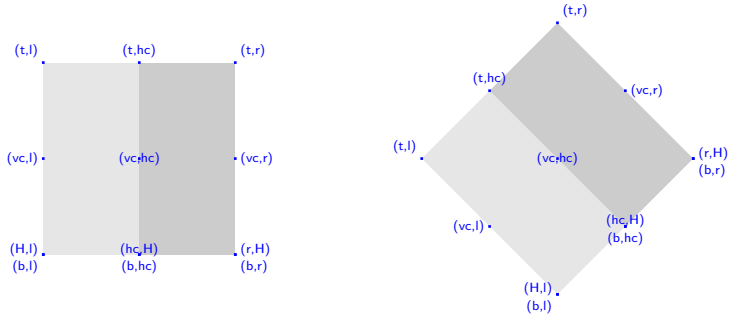

Figure 2: Coffin rotation: left, unrotated; right, rotated by 45°.

will give the distance from the baseline to the top of the bounding box of the relevant coffin.

`\Depth` `\Depth`
Within the ⟨*offset*⟩ argument of `\SetHorizontalPole` and `\SetVerticalPole`, `\Depth` will give the distance from the baseline to the bottom of the bounding box of the relevant coffin.

`\Width` `\Width`
Within the ⟨*offset*⟩ argument of `\SetHorizontalPole` and `\SetVerticalPole`, `\Width` will give the distance from the right edge to the left edge of the bounding box of the relevant coffin.

4 Rotating coffins

`\RotateCoffin` `\RotateCoffin` ⟨*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.

The effect of rotation on a coffin is illustrated in Figure 2. As is shown, the coffin handles will remain correctly positioned relative to the content of the coffin. The 'top' of a rotated coffin may of course no longer be the edge closest to the top of the physical page.

5 Resizing coffins

`\ResizeCoffin` `\RotateCoffin` ⟨*coffin*⟩ `{`⟨*width*⟩`}` `{`⟨*total-height*⟩`}`
Resized the ⟨*coffin*⟩ to ⟨*width*⟩ and ⟨*total-height*⟩, both of which should be given as di-

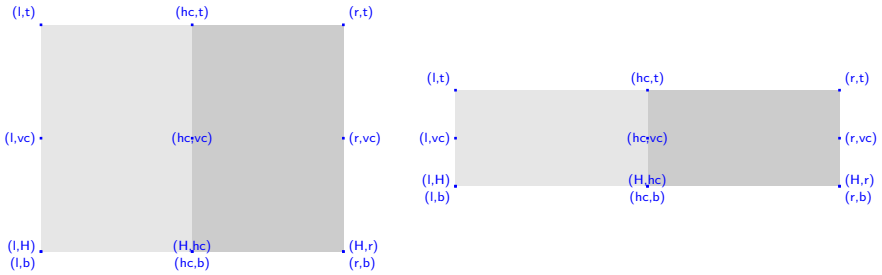

Figure 3: Coffin resizing: left, resized to exactly 4 cm by 6 cm; right, scaled a factors of 2 and 0.5 in x and y, respectively (example coffin as in Figure 2).

mension expressions. These may include the terms `\TotalHeight`, `\Height`, `\Depth` and `\Width`, which will evaluate to the appropriate dimensions of the ⟨*coffin*⟩.

`\ScaleCoffin` `\ScaleCoffin` ⟨*coffin*⟩ `{`⟨*x-scale*⟩`}` `{`⟨*y-scale*⟩`}`
Scales the ⟨*coffin*⟩ by a factors ⟨*x-scale*⟩ and ⟨*y-scale*⟩ in the horizontal and vertical directions, respectively. The two scale factors should be given as real numbers.

`\ResizeCoffin` and `\ScaleCoffin` can be used interchangeably: whether scale factors or absolute values are the best form for the resizing will depend upon the context (Figure 3).

6 Joining coffins

The key operation for coffins is joining coffins to each other. This is always carried out such that the first coffin is the 'parent', and is updated by the alignment. The second 'child' coffin is not altered by the alignment process.

`\JoinCoffins`
```
\JoinCoffins *
  ⟨coffin1⟩ [ ⟨coffin1-pole1⟩ , ⟨coffin1-pole2⟩ ]
  ⟨coffin2⟩ [ ⟨coffin2-pole1⟩ , ⟨coffin2-pole2⟩ ]
  ( ⟨x-offset⟩ , ⟨y-offset⟩ )
```
Joining of two coffins is carried out by the `\JoinCoffins` function, which takes two mandatory arguments: the 'parent' ⟨*coffin1*⟩ and the 'child' ⟨*coffin2*⟩. All of the other arguments shown are optional.

The standard `\JoinCoffins` functions joins ⟨*coffin2*⟩ to ⟨*coffin1*⟩ such that the bounding box of ⟨*coffin1*⟩ after the process will expand. The new bounding box will be the smallest rectangle covering the bounding boxes of the two input coffins. When the starred variant of `\JoinCoffins` is used, the bounding box of ⟨*coffin1*⟩ is not altered, *i.e.* ⟨*coffin2*⟩ may protrude outside of the bounding box of the updated ⟨*coffin1*⟩. The difference between the two forms of alignment is best illustrated using a visual example. In Figure 4, the two processes are contrasted. In both cases, the small red coffin has been aligned with the large grey coffin. In the left-hand illustration, the `\JoinCoffins` function was used,

Figure 4: Contrast between `\JoinCoffins` (left) and `\JoinCoffins*` (right); the bounding box of the coffin is show in black.

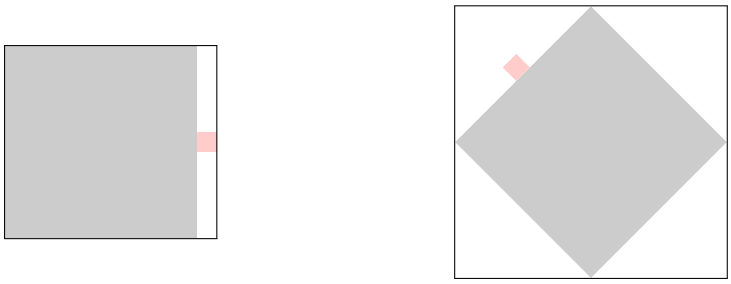

Figure 5: The effect of rotation of a joined coffin: the black line shows the coffin bounding box.

resulting in an expanded bounding box. In contrast, on the right `\AttachCoffin` was used, meaning that the bounding box does not include the area of the smaller 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*⟩. If the two ⟨*poles*⟩ are not specified, `\JoinCoffins` will use the default value `(H,l)`, *i.e.* the reference point used by TEX for the underlying box. Once the two ⟨*handles*⟩ have been located, ⟨*coffin2*⟩ is then attached to ⟨*coffin1*⟩ such that the relationship between ⟨*handle1*⟩ and ⟨*handle2*⟩ is described by the ⟨*x-offset*⟩ and ⟨*y-offset*⟩. This ⟨*offset*⟩ is an optional argument, and if it is not given then `(0 pt, 0 pt)` is used.

Notice that when `\JoinCoffins` is used the new bounding box is the smallest rectangle containing the bounding boxes of the two input coffins. As a result, it will include additional white space unless one coffin entirely overlaps the other (Figure 5, left). Rotation of coffins will take account of the extent of the material after rotation when re-calculating the bounding box. This means that no *unnecessary* white space will be added on rotation (Figure 5, right).

As part of the joining procedure, the poles of the two input coffins are preserved within the structure of the updated coffin. In this way it is possible to carry out complex alignment procedures. The poles of a coffin after alignment may therefore be divided into three groups:

1. The 'native' poles of the updated coffin, such as `l`, `r`, `hc`, *etc.*

Figure 6: Aligning coffins using poles from previous operations.

2. Poles derived from ⟨*coffin1*⟩, such as ⟨*coffin1*⟩`-l`, ⟨*coffin1*⟩`-r`, ⟨*coffin1*⟩`-hc`, *etc.*
3. Poles derived from ⟨*coffin2*⟩, such as ⟨*coffin2*⟩`-l`, ⟨*coffin2*⟩`-r`, ⟨*coffin2*⟩`-hc`, *etc.*

Applying this ability allows a series of joining operations to take place, as illustrated in Figure 6. In this example, the scheme used for alignment was as follows:

```
\SetHorizontalCoffin\OutputCoffin{}
\SetHorizontalCoffin\RedCoffin
  {\color{red!20!white}\rule{0.2 in}{0.2 in}}
\JoinCoffins\OutputCoffin[vc,hc]\RedCoffin[vc,hc]
\SetHorizontalCoffin\BlueCoffin
  {\color{blue!20!white}\rule{0.2 in}{0.2 in}}
\JoinCoffins\OutputCoffin[\RedCoffin-vc,\RedCoffin-hc]
  \BlueCoffin[b,l]
\SetHorizontalCoffin\GreenCoffin
  {\color{green!20!white}\rule{0.2 in}{0.2 in}}
\JoinCoffins\OutputCoffin[\BlueCoffin-vc,\BlueCoffin-hc]
  \GreenCoffin[b,l]
\SetHorizontalCoffin\YellowCoffin
  {\color{yellow!20!white}\rule{0.2 in}{0.2 in}}
\JoinCoffins\OutputCoffin[\GreenCoffin-vc,\GreenCoffin-hc]
  \YellowCoffin[b,l]
\SetHorizontalCoffin \OrangeCoffin
  {\color{orange!20!white}\rule{0.2 in}{0.2 in}}
\JoinCoffins\OutputCoffin[\BlueCoffin-t,\BlueCoffin-l]
  \OrangeCoffin[b,r]
\TypesetCoffin\OutputCoffin
```

This process begins by setting up `\OutputCoffin` to hold the joined output. Each join then takes place placing the new addition relative to the previous one. As each coffin joined has a unique name it is possible to align relative to each one of the component parts of the assembly. This is illustrated by the addition of the final `\OrangeCoffin` based on the earlier placement of the `\BlueCoffin`.

7 Typesetting coffins

\TypesetCoffin
⟨*coffin*⟩ [⟨*pole1*⟩ , ⟨*pole2*⟩]
(⟨*x-offset*⟩ , ⟨*y-offset*⟩)

Typesetting is carried out by first calculating ⟨*handle*⟩, the point of intersection of ⟨*pole1*⟩ and ⟨*pole2*⟩. This is an optional argument, and if not given then `(H, l)`, the TEX reference point of the underlying box, is used. 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*⟩. This ⟨*offset*⟩ is optional, and if not given `(0 pt, 0 pt)` is used. Typesetting a coffin is therefore analogous to carrying out an alignment where the 'parent' coffin is the current insertion point.

8 Diagnostic functions

Diagnostic data for following the coffin-building process is available both graphically and at the terminal. This reflects the fact that coffins are visual constructs.

\DisplayCoffinHandles `\DisplayCoffinHandles` ⟨*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.

\MarkCoffinHandle `\MarkCoffinHandle` ⟨*coffin*⟩
[⟨*pole1*⟩ , ⟨*pole2*⟩] {⟨*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. If no ⟨*poles*⟩ are give, the default `(H,l)` is used.

\ShowCoffinStructure `\ShowCoffinStructure` ⟨*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. For example, for the rotated coffin in Figure 2, the output of `\ShowCoffinStructure` is:

```
Size of coffin \ExampleCoffin:
```

```
> ht = 102.2052pt
> dp = 0.0pt
> wd = 102.2052pt

Poles of coffin \ExampleCoffin:
>  b => {51.1026pt}{0.0pt}{707.10678pt}{707.10678pt}
>  t => {0.0pt}{51.1026pt}{707.10678pt}{707.10678pt}
>  vc => {25.5513pt}{25.5513pt}{707.10678pt}{707.10678pt}
>  r => {102.2052pt}{51.1026pt}{-707.10678pt}{707.10678pt}
>  hc => {76.6539pt}{25.5513pt}{-707.10678pt}{707.10678pt}
>  T => {51.1026pt}{0.0pt}{707.10678pt}{707.10678pt}
>  H => {51.1026pt}{0.0pt}{707.10678pt}{707.10678pt}
>  B => {51.1026pt}{0.0pt}{707.10678pt}{707.10678pt}
>  l => {51.1026pt}{0.0pt}{-707.10678pt}{707.10678pt}.
\coffin_show_structure:N ...N \l_coffin_show_toks
```

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.

9 Implementation

```
⟨*package⟩
\ProvidesExplPackage
  {\filename}{\filedate}{\fileversion}{\filedescription}
\RequirePackage{l3coffins,xparse}
```

\l_coffin_A_hpole_tl
\l_coffin_A_vpole_tl
\l_coffin_B_hpole_tl
\l_coffin_B_vpole_tl
\l_coffin_bound_box_grow_bool
\l_coffin_hoffset_dim
\l_coffin_voffset_dim

Key–value definitions for the alignment system. With the exception of `grow-bounding-box`, all of these have to be given with a value.

```
\keys_define:nn { coffin } {
  coffin1-hpole     .tl_set:N        = \l_coffin_A_hpole_tl          ,
  coffin1-hpole     .value_required:                                 ,
  coffin1-vpole     .tl_set:N        = \l_coffin_A_vpole_tl          ,
  coffin1-vpole     .value_required:                                 ,
  coffin2-hpole     .tl_set:N        = \l_coffin_B_hpole_tl          ,
  coffin2-hpole     .value_required:                                 ,
  coffin2-vpole     .tl_set:N        = \l_coffin_B_vpole_tl          ,
  coffin2-vpole     .value_required:                                 ,
  grow-bounding-box .bool_set:N      = \l_coffin_bound_box_grow_bool ,
  grow-bounding-box .default:n       = true                          ,
  hoffset           .dim_set:N       = \l_coffin_hoffset_dim         ,
  hoffset           .value_required:                                 ,
  voffset           .dim_set:N       = \l_coffin_voffset_dim         ,
  voffset           .value_required:                                 ,
```

```
}
\keys_set:nn { coffin } {
  coffin1-hpole     = H    ,
  coffin1-vpole     = l    ,
  coffin2-hpole     = H    ,
  coffin2-vpole     = l    ,
  grow-bounding-box = true ,
  hoffset           = 0 pt ,
  voffset           = 0 pt ,
}
```

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

A lot of this is more-or-less just passing data straight through.

`\NewCoffin` This is a very easy conversion.

```
\NewDocumentCommand \NewCoffin { m } {
  \coffin_new:N #1
}
```

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

`\SetHorizontalCoffin` `\SetVerticalCoffin` These are again straight-forward translations.

```
\NewDocumentCommand \SetHorizontalCoffin { m +m } {
  \hcoffin_set:Nn #1 {#2}
}
\NewDocumentCommand \SetVerticalCoffin { m m +m } {
  \vcoffin_set:Nnn #1 {#2} {#3}
}
```

(*End definition for* `\SetHorizontalCoffin`. *This function is documented on page 3.*)

`\SetHorizontalPole` `\SetVerticalPole` Again straight-forward

```
\NewDocumentCommand \SetHorizontalPole { m m m } {
  \coffin_set_horizontal_pole:Nnn #1 {#2} {#3}
}
\NewDocumentCommand \SetVerticalPole { m m m } {
  \coffin_set_vertical_pole:Nnn #1 {#2} {#3}
}
```

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

`\JoinCoffins` The `\JoinCoffins` function needs to do a bit of work on the input syntax, as there are a number of optional arguments to worry about. The idea here is that `\JoinCoffins` can be used to either expand the bounding box of ⟨*coffin1*⟩ or add ⟨*coffin2*⟩ without any expansion of the bounding box. There are also the two handle positions and the offset to sort out.

```
\NewDocumentCommand \JoinCoffins
  {
    o
    s
    m
    > { \SplitArgument { 1 } { , } } O { H , 1 }
    m
    > { \SplitArgument { 1 } { , } } O { H , 1 }
    > { \SplitArgument { 1 } { , } } D ( ) { 0 pt , 0 pt }
  }
  {
    \IfNoValueTF {#1}
      {
        \IfBooleanTF #2
          { \coffin_attach:NnnNnnnn #3 #4 #5 #6 #7 }
          { \coffin_join:NnnNnnnn #3 #4 #5 #6 #7 }
      }
      {
        \group_begin:
          \keys_set:nn { coffin } {#1}
          \tl_set:Nx \l_coffin_tmp_tl
            {
              \group_end:
              \bool_if:NTF \l_coffin_bound_box_grow_bool
                { \coffin_join:NnnNnnnn }
                { \coffin_attach:NnnNnnnn }
              \exp_not:N #3
              { \exp_not:o { \l_coffin_A_hpole_tl } }
              { \exp_not:o { \l_coffin_A_vpole_tl } }
              \exp_not:N #5
              { \exp_not:o { \l_coffin_B_hpole_tl } }
              { \exp_not:o { \l_coffin_B_vpole_tl } }
              { \dim_use:N \l_coffin_hoffset_dim }
              { \dim_use:N \l_coffin_voffset_dim }
            }
        \l_coffin_tmp_tl
      }
  }
```

(*End definition for* `\JoinCoffins`*. This function is documented on page* 6.)

`\TypesetCoffin` For typesetting coffins there are two optional arguments, both of which need to be split. This is a simpler case of the code needed for `\JoinCoffins`.

```
\NewDocumentCommand \TypesetCoffin
  {
     m
     > { \SplitArgument { 1 } { , } } O { H , 1 }
     > { \SplitArgument { 1 } { , } } D ( ) { 0 pt , 0 pt }
  }
```

```
  { \coffin_typeset:Nnnnn #1 #2 #3 }
```

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

`\RotateCoffin`
`\ResizeCoffin`
`\ScaleCoffin`

Mores straight-forward copies.

```
\NewDocumentCommand \RotateCoffin  { m m } {
  \coffin_rotate:Nn #1 {#2}
}
\NewDocumentCommand \ResizeCoffin  { m m m } {
  \coffin_resize:Nnn #1 {#2} {#3}
}
\NewDocumentCommand \ScaleCoffin  { m m m } {
  \coffin_scale:Nnn #1 {#2} {#3}
}
```

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

`\DisplayCoffinHandles`

Displaying all of the handles is a bit easier, as there is no need to worry about the handle.

```
\NewDocumentCommand \DisplayCoffinHandles { m m } {
  \cs_if_exist:cTF { l_coffin_poles_ \tex_number:D #1 _prop }
    { \coffin_display_handles:Nn #1 {#2} }
    { \msg_kernel_error:nx { unknown-coffin } { \token_to_str:N #1 } }
}
```

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

`\MarkCoffinHandle`

Marking a handle requires a bit of work with the input, so that the design-level interface is 'nice'.

```
\NewDocumentCommand \MarkCoffinHandle
  { m > { \SplitArgument { 1 } { , } } O { H , l } m }
  {
    \cs_if_exist:cTF { l_coffin_poles_ \tex_number:D #1 _prop }
      { \coffin_mark_handle:Nnnn #1 #2 {#3} }
      { \msg_kernel_error:nx { unknown-coffin } { \token_to_str:N #1 }
  }
}
```

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

`\ShowCoffinStructure`

Back again to easy-to-implement functions.

```
\NewDocumentCommand \ShowCoffinStructure { m } {
  \coffin_show_structure:N #1
}
```

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

⟨/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.