# File I
# Implementation

## 1 l3backend-basics Implementation

```
⟨*package⟩
```

Whilst there is a reasonable amount of code overlap between backends, it is much clearer to have the blocks more-or-less separated than run in together and DocStripped out in parts. As such, most of the following is set up on a per-backend basis, though there is some common code (again given in blocks not interspersed with other material).

All the file identifiers are up-front so that they come out in the right place in the files.

```
\ProvidesExplFile
⟨*dvipdfmx⟩
  {l3backend-dvipdfmx.def}{2023-01-16}{}
  {L3 backend support: dvipdfmx}
⟨/dvipdfmx⟩
⟨*dvips⟩
  {l3backend-dvips.def}{2023-01-16}{}
  {L3 backend support: dvips}
⟨/dvips⟩
⟨*dvisvgm⟩
  {l3backend-dvisvgm.def}{2023-01-16}{}
  {L3 backend support: dvisvgm}
⟨/dvisvgm⟩
⟨*luatex⟩
  {l3backend-luatex.def}{2023-01-16}{}
  {L3 backend support: PDF output (LuaTeX)}
⟨/luatex⟩
⟨*pdftex⟩
  {l3backend-pdftex.def}{2023-01-16}{}
  {L3 backend support: PDF output (pdfTeX)}
⟨/pdftex⟩
⟨*xetex⟩
  {l3backend-xetex.def}{2023-01-16}{}
  {L3 backend support: XeTeX}
⟨/xetex⟩
```

Check if the loaded kernel is at least enough to load this file. The kernel date has to be at least equal to `\ExplBackendFileDate` or later. If `\__kernel_dependency_version_check:Nn` doesn't exist we're loading in an older kernel, so it's an error anyway. With time, this test should vanish and only the dependency check should remain.

```
\cs_if_exist:NTF \__kernel_dependency_version_check:nn
  {
    \__kernel_dependency_version_check:nn {2021-02-18}
⟨dvipdfmx⟩      {l3backend-dvipdfmx.def}
⟨dvips⟩      {l3backend-dvips.def}
⟨dvisvgm⟩      {l3backend-dvisvgm.def}
⟨luatex⟩      {l3backend-luatex.def}
⟨pdftex⟩      {l3backend-pdftex.def}
⟨xetex⟩      {l3backend-xetex.def}
```

```
  }
  {
    \cs_if_exist_use:cF { @latex@error } { \errmessage }
      {
        Mismatched~LaTeX~support~files~detected. \MessageBreak
        Loading~aborted!
      }
      { \use:c { @ehd } }
    \tex_endinput:D
  }
```

The order of the backend code here is such that we get somewhat logical outcomes in terms of code sharing whilst keeping things readable. (Trying to mix all of the code by concept is almost unmanageable.) The key parts which are shared are

- Color support is either `dvips`-like or LuaTEX/pdfTeX-like.
- LuaTEX/pdfTeX and `dvipdfmx`/XƎTEX share drawing routines.
- XƎTEX is the same as `dvipdfmx` other than image size extraction so takes most of the same code.

`\__kernel_backend_literal:e`
`\__kernel_backend_literal:n`
`\__kernel_backend_literal:x`

The one shared function for all backends is access to the basic `\special` primitive: it has slightly odd expansion behaviour so a wrapper is provided.

```
\cs_new_eq:NN \__kernel_backend_literal:e \tex_special:D
\cs_new_protected:Npn \__kernel_backend_literal:n #1
  { \__kernel_backend_literal:e { \exp_not:n {#1} } }
\cs_generate_variant:Nn \__kernel_backend_literal:n { x }
```

(*End definition for* `\__kernel_backend_literal:e`.)

`\__kernel_backend_first_shipout:n`

We need to write at first shipout in a few places. As we want to use the most up-to-date method,

```
\cs_if_exist:NTF \@ifl@t@r
  {
    \@ifl@t@r \fmtversion { 2020-10-01 }
      {
        \cs_new_protected:Npn \__kernel_backend_first_shipout:n #1
          { \hook_gput_code:nnn { shipout / firstpage } { l3backend } {#1} }
      }
      { \cs_new_eq:NN \__kernel_backend_first_shipout:n \AtBeginDvi }
  }
  { \cs_new_eq:NN \__kernel_backend_first_shipout:n \use:n }
```

(*End definition for* `\__kernel_backend_first_shipout:n`.)

## 1.1 dvips backend

```
⟨*dvips⟩
```

`\__kernel_backend_literal_postscript:n`
`\__kernel_backend_literal_postscript:x`

Literal PostScript can be included using a few low-level formats. Here, we use the form with no positioning: this is overall more convenient as a wrapper. Note that this does require that where position is important, an appropriate wrapper is included.

```
\cs_new_protected:Npn \__kernel_backend_literal_postscript:n #1
  { \__kernel_backend_literal:n { ps:: #1 } }
\cs_generate_variant:Nn \__kernel_backend_literal_postscript:n { x }
```

(*End definition for* `\__kernel_backend_literal_postscript:n`.)

`\__kernel_backend_postscript:n`
`\__kernel_backend_postscript:x`

PostScript data that does have positioning, and also applying a shift to `SDict` (which is not done automatically by `ps:` or `ps::`, in contrast to `!` or `"`).

```
\cs_new_protected:Npn \__kernel_backend_postscript:n #1
  { \__kernel_backend_literal:n { ps: SDict ~ begin ~ #1 ~ end } }
\cs_generate_variant:Nn \__kernel_backend_postscript:n { x }
```

(*End definition for* `\__kernel_backend_postscript:n`.)

PostScript for the header: a small saving but makes the code clearer. This is held until the start of shipout such that a document with no actual output does not write anything.

```
\bool_if:NT \g__kernel_backend_header_bool
  {
    \__kernel_backend_first_shipout:n
      { \__kernel_backend_literal:n { header = l3backend-dvips.pro } }
  }
```

`\__kernel_backend_align_begin:`
`\__kernel_backend_align_end:`

In `dvips` there is no built-in saving of the current position, and so some additional PostScript is required to set up the transformation matrix and also to restore it afterwards. Notice the use of the stack to save the current position "up front" and to move back to it at the end of the process. Notice that the `[begin]`/`[end]` pair here mean that we can use a run of PostScript statements in separate lines: not *required* but does make the code and output more clear.

```
\cs_new_protected:Npn \__kernel_backend_align_begin:
  {
    \__kernel_backend_literal:n { ps::[begin] }
    \__kernel_backend_literal_postscript:n { currentpoint }
    \__kernel_backend_literal_postscript:n { currentpoint~translate }
  }
\cs_new_protected:Npn \__kernel_backend_align_end:
  {
    \__kernel_backend_literal_postscript:n { neg~exch~neg~exch~translate }
    \__kernel_backend_literal:n { ps::[end] }
  }
```

(*End definition for* `\__kernel_backend_align_begin:` *and* `\__kernel_backend_align_end:`.)

`\__kernel_backend_scope_begin:`
`\__kernel_backend_scope_end:`

Saving/restoring scope for general operations needs to be done with `dvips` positioning (try without to see this!). Thus we need the `ps:` version of the special here. As only the graphics state is ever altered within this pairing, we use the lower-cost `g`-versions.

```
\cs_new_protected:Npn \__kernel_backend_scope_begin:
  { \__kernel_backend_literal:n { ps:gsave } }
\cs_new_protected:Npn \__kernel_backend_scope_end:
  { \__kernel_backend_literal:n { ps:grestore } }
```

(*End definition for* `\__kernel_backend_scope_begin:` *and* `\__kernel_backend_scope_end:`.)

```
⟨/dvips⟩
```

## 1.2 LuaTeX and pdfTeX backends

```
⟨*luatex | pdftex⟩
```

Both LuaTeX and pdfTeX write PDFs directly rather than via an intermediate file. Although there are similarities, the move of LuaTeX to have more code in Lua means we create two independent files using shared DocStrip code.

__kernel_backend_literal_pdf:n
__kernel_backend_literal_pdf:x

This is equivalent to `\special{pdf:}` but the engine can track it. Without the `direct` keyword everything is kept in sync: the transformation matrix is set to the current point automatically. Note that this is still inside the text (`BT` ... `ET` block).

```
\cs_new_protected:Npn \__kernel_backend_literal_pdf:n #1
  {
⟨*luatex⟩
    \tex_pdfextension:D literal
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfliteral:D
⟨/pdftex⟩
      { \exp_not:n {#1} }
  }
\cs_generate_variant:Nn \__kernel_backend_literal_pdf:n { x }
```

(*End definition for* `\__kernel_backend_literal_pdf:n`.)

__kernel_backend_literal_page:n

Page literals are pretty simple. To avoid an expansion, we write out by hand.

```
\cs_new_protected:Npn \__kernel_backend_literal_page:n #1
  {
⟨*luatex⟩
    \tex_pdfextension:D literal ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfliteral:D
⟨/pdftex⟩
        page { \exp_not:n {#1} }
  }
```

(*End definition for* `\__kernel_backend_literal_page:n`.)

__kernel_backend_scope_begin:
__kernel_backend_scope_end:

Higher-level interfaces for saving and restoring the graphic state.

```
\cs_new_protected:Npn \__kernel_backend_scope_begin:
  {
⟨*luatex⟩
    \tex_pdfextension:D save \scan_stop:
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfsave:D
⟨/pdftex⟩
  }
\cs_new_protected:Npn \__kernel_backend_scope_end:
  {
⟨*luatex⟩
    \tex_pdfextension:D restore \scan_stop:
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfrestore:D
⟨/pdftex⟩
  }
```

(*End definition for* `\__kernel_backend_scope_begin:` *and* `\__kernel_backend_scope_end:`.)

`\__kernel_backend_matrix:n`
`\__kernel_backend_matrix:x`

Here the appropriate function is set up to insert an affine matrix into the PDF. With pdfTEX and LuaTEX in direct PDF output mode there is a primitive for this, which only needs the rotation/scaling/skew part.

```
\cs_new_protected:Npn \__kernel_backend_matrix:n #1
  {
⟨*luatex⟩
    \tex_pdfextension:D setmatrix
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfsetmatrix:D
⟨/pdftex⟩
        { \exp_not:n {#1} }
  }
\cs_generate_variant:Nn \__kernel_backend_matrix:n { x }
```

(*End definition for* `\__kernel_backend_matrix:n`.)

```
⟨/luatex | pdftex⟩
```

## 1.3 dvipdfmx backend

```
⟨*dvipdfmx | xetex⟩
```

The `dvipdfmx` shares code with the PDF mode one (using the common section to this file) but also with XƎTEX. The latter is close to identical to `dvipdfmx` and so all of the code here is extracted for both backends, with some `clean up` for XƎTEX as required.

`\__kernel_backend_literal_pdf:n`
`\__kernel_backend_literal_pdf:x`

Undocumented but equivalent to pdfTEX's `literal` keyword. It's similar to be not the same as the documented `contents` keyword as that adds a `q`/`Q` pair.

```
\cs_new_protected:Npn \__kernel_backend_literal_pdf:n #1
  { \__kernel_backend_literal:n { pdf:literal~ #1 } }
\cs_generate_variant:Nn \__kernel_backend_literal_pdf:n { x }
```

(*End definition for* `\__kernel_backend_literal_pdf:n`.)

`\__kernel_backend_literal_page:n`

Whilst the manual says this is like `literal direct` in pdfTEX, it closes the `BT` block!

```
\cs_new_protected:Npn \__kernel_backend_literal_page:n #1
  { \__kernel_backend_literal:n { pdf:literal~direct~ #1 } }
```

(*End definition for* `\__kernel_backend_literal_page:n`.)

`\__kernel_backend_scope_begin:`
`\__kernel_backend_scope_end:`

Scoping is done using the backend-specific specials. We use the versions originally from `xdvidfpmx` (`x:`) as these are well-tested "in the wild".

```
\cs_new_protected:Npn \__kernel_backend_scope_begin:
  { \__kernel_backend_literal:n { x:gsave } }
\cs_new_protected:Npn \__kernel_backend_scope_end:
  { \__kernel_backend_literal:n { x:grestore } }
```

(*End definition for* `\__kernel_backend_scope_begin:` *and* `\__kernel_backend_scope_end:`.)

```
⟨/dvipdfmx | xetex⟩
```

## 1.4 `dvisvgm` backend

```
⟨*dvisvgm⟩
```

__kernel_backend_literal_svg:n
__kernel_backend_literal_svg:x

Unlike the other backends, the requirements for making SVG files mean that we can't conveniently transform all operations to the current point. That makes life a bit more tricky later as that needs to be accounted for. A new line is added after each call to help to keep the output readable for debugging.

```
\cs_new_protected:Npn \__kernel_backend_literal_svg:n #1
  { \__kernel_backend_literal:n { dvisvgm:raw~ #1 { ?nl } } }
\cs_generate_variant:Nn \__kernel_backend_literal_svg:n { x }
```

(*End definition for* `\__kernel_backend_literal_svg:n`.)

\g__kernel_backend_scope_int
\l__kernel_backend_scope_int

In SVG, we need to track scope nesting as properties attach to scopes; that requires a pair of `int` registers.

```
\int_new:N \g__kernel_backend_scope_int
\int_new:N \l__kernel_backend_scope_int
```

(*End definition for* `\g__kernel_backend_scope_int` *and* `\l__kernel_backend_scope_int`.)

__kernel_backend_scope_begin:
__kernel_backend_scope_end:
__kernel_backend_scope_begin:n
__kernel_backend_scope_begin:x
__kernel_backend_scope:n
__kernel_backend_scope:x

In SVG, the need to attach concepts to a scope means we need to be sure we will close all of the open scopes. That is easiest done if we only need an outer "wrapper" `begin`/`end` pair, and within that we apply operations as a simple scoped statements. To keep down the non-productive groups, we also have a `begin` version that does take an argument.

```
\cs_new_protected:Npn \__kernel_backend_scope_begin:
  {
    \__kernel_backend_literal_svg:n { <g> }
    \int_set_eq:NN
      \l__kernel_backend_scope_int
      \g__kernel_backend_scope_int
    \group_begin:
      \int_gset:Nn \g__kernel_backend_scope_int { 1 }
  }
\cs_new_protected:Npn \__kernel_backend_scope_end:
  {
      \prg_replicate:nn
        { \g__kernel_backend_scope_int }
        { \__kernel_backend_literal_svg:n { </g> } }
    \group_end:
    \int_gset_eq:NN
      \g__kernel_backend_scope_int
      \l__kernel_backend_scope_int
  }
\cs_new_protected:Npn \__kernel_backend_scope_begin:n #1
  {
    \__kernel_backend_literal_svg:n { <g ~ #1 > }
    \int_set_eq:NN
      \l__kernel_backend_scope_int
      \g__kernel_backend_scope_int
    \group_begin:
      \int_gset:Nn \g__kernel_backend_scope_int { 1 }
  }
\cs_generate_variant:Nn \__kernel_backend_scope_begin:n { x }
```

```
\cs_new_protected:Npn \__kernel_backend_scope:n #1
  {
    \__kernel_backend_literal_svg:n { <g ~ #1 > }
    \int_gincr:N \g__kernel_backend_scope_int
  }
\cs_generate_variant:Nn \__kernel_backend_scope:n { x }
```

(*End definition for* `\__kernel_backend_scope_begin:` *and others.*)

```
⟨/dvisvgm⟩
```

```
⟨/package⟩
```

## 2 l3backend-box Implementation

```
⟨*package⟩
⟨@@=box⟩
```

### 2.1 dvips backend

```
⟨*dvips⟩
```

`\__box_backend_clip:N`

The `dvips` backend scales all absolute dimensions based on the output resolution selected and any TeX magnification. Thus for any operation involving absolute lengths there is a correction to make. See `normalscale` from `special.pro` for the variables, noting that here everything is saved on the stack rather than as a separate variable. Once all of that is done, the actual clipping is trivial.

```
\cs_new_protected:Npn \__box_backend_clip:N #1
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_align_begin:
    \__kernel_backend_literal_postscript:n { matrix~currentmatrix }
    \__kernel_backend_literal_postscript:n
      { Resolution~72~div~VResolution~72~div~scale }
    \__kernel_backend_literal_postscript:n { DVImag~dup~scale }
    \__kernel_backend_literal_postscript:x
      {
        0 ~
        \dim_to_decimal_in_bp:n { \box_dp:N #1 } ~
        \dim_to_decimal_in_bp:n { \box_wd:N #1 } ~
        \dim_to_decimal_in_bp:n { -\box_ht:N #1 - \box_dp:N #1 } ~
        rectclip
      }
    \__kernel_backend_literal_postscript:n { setmatrix }
    \__kernel_backend_align_end:
    \hbox_overlap_right:n { \box_use:N #1 }
    \__kernel_backend_scope_end:
    \skip_horizontal:n { \box_wd:N #1 }
  }
```

(*End definition for* `\__box_backend_clip:N`*.*)

`\__box_backend_rotate:Nn`
`\__box_backend_rotate_aux:Nn`

Rotating using `dvips` does not require that the box dimensions are altered and has a very convenient built-in operation. Zero rotation must be written as `0` not `-0` so there is a quick test.

```
\cs_new_protected:Npn \__box_backend_rotate:Nn #1#2
  { \exp_args:NNf \__box_backend_rotate_aux:Nn #1 { \fp_eval:n {#2} } }
\cs_new_protected:Npn \__box_backend_rotate_aux:Nn #1#2
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_align_begin:
    \__kernel_backend_literal_postscript:x
      {
        \fp_compare:nNnTF {#2} = \c_zero_fp
          { 0 }
          { \fp_eval:n { round ( -(#2) , 5 ) } } ~
        rotate
      }
   \__kernel_backend_align_end:
   \box_use:N #1
   \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__box_backend_rotate:Nn` *and* `\__box_backend_rotate_aux:Nn`*.*)

`\__box_backend_scale:Nnn`

The `dvips` backend once again has a dedicated operation we can use here.

```
\cs_new_protected:Npn \__box_backend_scale:Nnn #1#2#3
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_align_begin:
    \__kernel_backend_literal_postscript:x
      {
        \fp_eval:n { round ( #2 , 5 ) } ~
        \fp_eval:n { round ( #3 , 5 ) } ~
        scale
      }
    \__kernel_backend_align_end:
    \hbox_overlap_right:n { \box_use:N #1 }
    \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__box_backend_scale:Nnn`*.*)

⟨/dvips⟩

## 2.2 LuaTeX and pdfTeX backends

⟨*luatex | pdftex⟩

`\__box_backend_clip:N`

The general method is to save the current location, define a clipping path equivalent to the bounding box, then insert the content at the current position and in a zero width box. The "real" width is then made up using a horizontal skip before tidying up. There are other approaches that can be taken (for example using XForm objects), but the logic here shares as much code as possible and uses the same conversions (and so same rounding errors) in all cases.

```
\cs_new_protected:Npn \__box_backend_clip:N #1
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_literal_pdf:x
      {
        0~
        \dim_to_decimal_in_bp:n { -\box_dp:N #1 } ~
        \dim_to_decimal_in_bp:n { \box_wd:N #1 } ~
        \dim_to_decimal_in_bp:n { \box_ht:N #1 + \box_dp:N #1 } ~
        re~W~n
      }
    \hbox_overlap_right:n { \box_use:N #1 }
    \__kernel_backend_scope_end:
    \skip_horizontal:n { \box_wd:N #1 }
  }
```

(*End definition for* `\__box_backend_clip:N`.)

`\__box_backend_rotate:Nn`
`\__box_backend_rotate_aux:Nn`
`\l__box_backend_cos_fp`
`\l__box_backend_sin_fp`

Rotations are set using an affine transformation matrix which therefore requires sine/cosine values not the angle itself. We store the rounded values to avoid rounding twice. There are also a couple of comparisons to ensure that `-0` is not written to the output, as this avoids any issues with problematic display programs. Note that numbers are compared to 0 after rounding.

```
\cs_new_protected:Npn \__box_backend_rotate:Nn #1#2
  { \exp_args:NNf \__box_backend_rotate_aux:Nn #1 { \fp_eval:n {#2} } }
\cs_new_protected:Npn \__box_backend_rotate_aux:Nn #1#2
  {
    \__kernel_backend_scope_begin:
    \box_set_wd:Nn #1 { 0pt }
    \fp_set:Nn \l__box_backend_cos_fp { round ( cosd ( #2 ) , 5 ) }
    \fp_compare:nNnT \l__box_backend_cos_fp = \c_zero_fp
      { \fp_zero:N \l__box_backend_cos_fp }
    \fp_set:Nn \l__box_backend_sin_fp { round ( sind ( #2 ) , 5 ) }
    \__kernel_backend_matrix:x
      {
        \fp_use:N \l__box_backend_cos_fp \c_space_tl
        \fp_compare:nNnTF \l__box_backend_sin_fp = \c_zero_fp
          { 0~0 }
          {
            \fp_use:N \l__box_backend_sin_fp
            \c_space_tl
            \fp_eval:n { -\l__box_backend_sin_fp }
          }
        \c_space_tl
        \fp_use:N \l__box_backend_cos_fp
      }
   \box_use:N #1
   \__kernel_backend_scope_end:
  }
\fp_new:N \l__box_backend_cos_fp
\fp_new:N \l__box_backend_sin_fp
```

(*End definition for* `\__box_backend_rotate:Nn` *and others.*)

`\__box_backend_scale:Nnn`

The same idea as for rotation but without the complexity of signs and cosines.

```
\cs_new_protected:Npn \__box_backend_scale:Nnn #1#2#3
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_matrix:x
```

```
      {
        \fp_eval:n { round ( #2 , 5 ) } ~
        0~0~
        \fp_eval:n { round ( #3 , 5 ) }
      }
    \hbox_overlap_right:n { \box_use:N #1 }
    \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__box_backend_scale:Nnn`.)

```
⟨/luatex | pdftex⟩
```

## 2.3 `dvipdfmx`/XƎTEX backend

```
⟨*dvipdfmx | xetex⟩
```

`\__box_backend_clip:N`

The code here is identical to that for LuaTEX/pdfTEX: unlike rotation and scaling, there is no higher-level support in the backend for clipping.

```
\cs_new_protected:Npn \__box_backend_clip:N #1
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_literal_pdf:x
      {
        0~
        \dim_to_decimal_in_bp:n { -\box_dp:N #1 } ~
        \dim_to_decimal_in_bp:n { \box_wd:N #1 } ~
        \dim_to_decimal_in_bp:n { \box_ht:N #1 + \box_dp:N #1 } ~
        re~W~n
      }
    \hbox_overlap_right:n { \box_use:N #1 }
    \__kernel_backend_scope_end:
    \skip_horizontal:n { \box_wd:N #1 }
  }
```

(*End definition for* `\__box_backend_clip:N`.)

`\__box_backend_rotate:Nn`
`\__box_backend_rotate_aux:Nn`

Rotating in `dvipdmfx`/XƎTEX can be implemented using either PDF or backend-specific code. The former approach however is not “aware” of the content of boxes: this means that any embedded links would not be adjusted by the rotation. As such, the backend-native approach is preferred: the code therefore is similar (though not identical) to the `dvips` version (notice the rotation angle here is positive). As for `dvips`, zero rotation is written as `0` not `-0`.

```
\cs_new_protected:Npn \__box_backend_rotate:Nn #1#2
  { \exp_args:NNf \__box_backend_rotate_aux:Nn #1 { \fp_eval:n {#2} } }
\cs_new_protected:Npn \__box_backend_rotate_aux:Nn #1#2
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_literal:x
      {
        x:rotate~
        \fp_compare:nNnTF {#2} = \c_zero_fp
          { 0 }
          { \fp_eval:n { round ( #2 , 5 ) } }
      }
    \box_use:N #1
    \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__box_backend_rotate:Nn` *and* `\__box_backend_rotate_aux:Nn`.)

`\__box_backend_scale:Nnn`

Much the same idea for scaling: use the higher-level backend operation to allow for box content.

```
\cs_new_protected:Npn \__box_backend_scale:Nnn #1#2#3
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_literal:x
      {
        x:scale~
        \fp_eval:n { round ( #2 , 5 ) } ~
        \fp_eval:n { round ( #3 , 5 ) }
      }
    \hbox_overlap_right:n { \box_use:N #1 }
    \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__box_backend_scale:Nnn`.)

```
⟨/dvipdfmx | xetex⟩
```

## 2.4 `dvisvgm` backend

```
⟨*dvisvgm⟩
```

`\__box_backend_clip:N`
`\g__kernel_clip_path_int`

Clipping in SVG is more involved than with other backends. The first issue is that the clipping path must be defined separately from where it is used, so we need to track how many paths have applied. The naming here uses `l3cp` as the namespace with a number following. Rather than use a rectangular operation, we define the path manually as this allows it to have a depth: easier than the alternative approach of shifting content up and down using scopes to allow for the depth of the TeX box and keep the reference point the same!

```
\cs_new_protected:Npn \__box_backend_clip:N #1
  {
    \int_gincr:N \g__kernel_clip_path_int
    \__kernel_backend_literal_svg:x
      { < clipPath~id = " l3cp \int_use:N \g__kernel_clip_path_int " > }
    \__kernel_backend_literal_svg:x
      {
        <
          path ~ d =
            "
              M ~ 0 ~
                  \dim_to_decimal:n { -\box_dp:N #1 } ~
              L ~ \dim_to_decimal:n { \box_wd:N #1 } ~
                  \dim_to_decimal:n { -\box_dp:N #1 } ~
              L ~ \dim_to_decimal:n { \box_wd:N #1 }  ~
                  \dim_to_decimal:n { \box_ht:N #1 + \box_dp:N #1 } ~
              L ~ 0 ~
                  \dim_to_decimal:n { \box_ht:N #1 + \box_dp:N #1 } ~
              Z
            "
        />
      }
    \__kernel_backend_literal_svg:n
      { < /clipPath > }
```

In general the SVG set up does not try to transform coordinates to the current point. For clipping we need to do that, so have a transformation here to get us to the right place, and a matching one just before the TeX box is inserted to get things back on track. The clip path needs to come between those two such that if lines up with the current point, as does the TeX box.

```
    \__kernel_backend_scope_begin:n
      {
        transform =
          "
            translate ( { ?x } , { ?y } ) ~
            scale ( 1 , -1 )
          "
      }
    \__kernel_backend_scope:x
      {
        clip-path =
          "url ( \c_hash_str l3cp \int_use:N \g__kernel_clip_path_int ) "
      }
    \__kernel_backend_scope:n
      {
        transform =
          "
            scale ( -1 , 1 ) ~
            translate ( { ?x } , { ?y } ) ~
            scale ( -1 , -1 )
          "
      }
    \box_use:N #1
    \__kernel_backend_scope_end:
  }
\int_new:N \g__kernel_clip_path_int
```

(*End definition for* `\__box_backend_clip:N` *and* `\g__kernel_clip_path_int`.)

`\__box_backend_rotate:Nn` Rotation has a dedicated operation which includes a centre-of-rotation optional pair. That can be picked up from the backend syntax, so there is no need to worry about the transformation matrix.

```
\cs_new_protected:Npn \__box_backend_rotate:Nn #1#2
  {
    \__kernel_backend_scope_begin:x
      {
        transform =
          "
            rotate
            ( \fp_eval:n { round ( -(#2) , 5 ) } , ~ { ?x } , ~ { ?y } )
          "
      }
    \box_use:N #1
    \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__box_backend_rotate:Nn`.)

`\__box_backend_scale:Nnn`

In contrast to rotation, we have to account for the current position in this case. That is done using a couple of translations in addition to the scaling (which is therefore done backward with a flip).

```
\cs_new_protected:Npn \__box_backend_scale:Nnn #1#2#3
  {
    \__kernel_backend_scope_begin:x
      {
        transform =
          "
            translate ( { ?x } , { ?y } ) ~
            scale
              (
                \fp_eval:n { round ( -#2 , 5 ) } ,
                \fp_eval:n { round ( -#3 , 5 ) }
              ) ~
            translate ( { ?x } , { ?y } ) ~
            scale ( -1 )
          "
      }
    \hbox_overlap_right:n { \box_use:N #1 }
    \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__box_backend_scale:Nnn`.)

```
⟨/dvisvgm⟩
⟨/package⟩
```

# 3 **l3backend-color** Implementation

```
⟨*package⟩
⟨@@=color⟩
```

Color support is split into parts: collecting data from LaTeX 2ε, the color stack, general color, separations, and color for drawings. We have different approaches in each backend, and have some choices to make about `dvipdfmx`/XeTeX in particular. Whilst it is in some ways convenient to use the same approach in multiple backends, the fact that `dvipdfmx`/XeTeX is PDF-based means it (largely) sticks closer to direct PDF output.

## 3.1 Collecting information from LaTeX 2ε

### 3.1.1 `dvips`-style

```
⟨*dvisvgm | dvipdfmx | dvips | xetex⟩
```

`\__color_backend_pickup:N`
`\__color_backend_pickup:w`

Allow for LaTeX 2ε color. Here, the possible input values are limited: `dvips`-style colors can be taken as-is. The `x`-type expansion is there to cover the case where xcolor is in use.

```
\cs_new_protected:Npn \__color_backend_pickup:N #1
  {
    \exp_args:NV \tl_if_head_is_space:nTF \current@color
```

```
      {
        \tl_set:Nn #1 { { gray } { 0 } }
        \msg_warning:nnx { color } { unhandled }
          { \current@color }
      }
      {
        \exp_last_unbraced:Nx \__color_backend_pickup:w
          { \current@color } \s__color_stop #1
      }
  }
\cs_new_protected:Npn \__color_backend_pickup:w #1 ~ #2 \s__color_stop #3
  { \tl_set:Nn #3 { {#1} {#2} } }
```

(*End definition for* `\__color_backend_pickup:N` *and* `\__color_backend_pickup:w`.)

```
⟨/dvisvgm | dvipdfmx | dvips | xetex⟩
```

### 3.1.2 LuaTeX and pdfTeX

```
⟨*luatex | pdftex⟩
```

`\__color_backend_pickup:N`
`\__color_backend_pickup:w`

Same ideas, but with a different backend-dependent format.

```
\cs_new_protected:Npn \__color_backend_pickup:N #1
  {
    \exp_last_unbraced:Nx \__color_backend_pickup:w
      { \current@color } ~ 0 ~ 0 ~ 0 \s__color_stop #1
  }
\cs_new_protected:Npn \__color_backend_pickup:w
  #1 ~ #2 ~ #3 ~ #4 ~ #5 ~ #6 \s__color_stop #7
  {
    \str_if_eq:nnTF {#2} { g }
      { \tl_set:Nn #7 { { gray } {#1} } }
      {
        \str_if_eq:nnTF {#4} { rg }
          { \tl_set:Nn #7 { { rgb } { #1 ~ #2 ~ #3 } } }
          {
            \str_if_eq:nnTF {#5} { k }
              { \tl_set:Nn #7 { { cmyk } { #1 ~ #2 ~ #3 ~ #4 } } }
              {
                \tl_set:Nn #1 { { gray } { 0 } }
                \msg_warning:nnx { color } { unhandled }
                  { \current@color }
              }
          }
      }
  }
```

(*End definition for* `\__color_backend_pickup:N` *and* `\__color_backend_pickup:w`.)

```
⟨/luatex | pdftex⟩
```

## 3.2 The color stack

For PDF-based engines, we have a color stack available inside the specials. This is used for concepts beyond color itself: it is needed to manage the graphics state generally. Although `dvipdfmx`/XeTeX have multiple color stacks in recent releases, the way these interact with the original single stack and with other graphic state operations means that currently it is not feasible to use the multiple stacks.

### 3.2.1 Common code

```
⟨*luatex | pdftex⟩
```

\l__color_backend_stack_int

For tracking which stack is in use where multiple stacks are used: currently just pdfTEX/LuaTEX but at some future stage may also cover `dvipdfmx`/XƎTEX.

```
\int_new:N \l__color_backend_stack_int
```

(*End definition for* `\l__color_backend_stack_int`.)

```
⟨/luatex | pdftex⟩
```

### 3.2.2 LuaTEXand pdfTEX

```
⟨*luatex | pdftex⟩
```

__kernel_color_backend_stack_init:Nnn

```
\cs_new_protected:Npn \__kernel_color_backend_stack_init:Nnn #1#2#3
  {
    \int_const:Nn #1
      {
⟨*luatex⟩
        \tex_pdffeedback:D colorstackinit ~
⟨/luatex⟩
⟨*pdftex⟩
        \tex_pdfcolorstackinit:D
⟨/pdftex⟩
        \tl_if_blank:nF {#2} { #2 ~ }
        {#3}
      }
  }
```

(*End definition for* `\__kernel_color_backend_stack_init:Nnn`.)

__kernel_color_backend_stack_push:nn
__kernel_color_backend_stack_pop:n

```
\cs_new_protected:Npn \__kernel_color_backend_stack_push:nn #1#2
  {
⟨*luatex⟩
    \tex_pdfextension:D colorstack ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfcolorstack:D
⟨/pdftex⟩
      \int_eval:n {#1} ~ push ~ {#2}
  }
\cs_new_protected:Npn \__kernel_color_backend_stack_pop:n #1
  {
⟨*luatex⟩
    \tex_pdfextension:D colorstack ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfcolorstack:D
⟨/pdftex⟩
```

```
      \int_eval:n {#1} ~ pop \scan_stop:
  }
```

(*End definition for* `\__kernel_color_backend_stack_push:nn` *and* `\__kernel_color_backend_stack_pop:n`.)

```
⟨/luatex | pdftex⟩
```

## 3.3 General color

### 3.3.1 dvips-style

```
⟨*dvips | dvisvgm⟩
```

__color_backend_select_cmyk:n
__color_backend_select_gray:n
__color_backend_select_named:n
__color_backend_select_rgb:n
__color_backend_select:n
__color_backend_reset:
color.sc

Push the data to the stack. In the case of `dvips` also saves the drawing color in raw PostScript. The `spot` model is for handling data in classical format.

```
\cs_new_protected:Npn \__color_backend_select_cmyk:n #1
  { \__color_backend_select:n { cmyk ~ #1 } }
\cs_new_protected:Npn \__color_backend_select_gray:n #1
  { \__color_backend_select:n { gray ~ #1 } }
\cs_new_protected:Npn \__color_backend_select_named:n #1
  { \__color_backend_select:n { ~ #1 } }
\cs_new_protected:Npn \__color_backend_select_rgb:n #1
  { \__color_backend_select:n { rgb ~ #1 } }
\cs_new_protected:Npn \__color_backend_select:n #1
  {
    \__kernel_backend_literal:n { color~push~ #1 }
⟨*dvips⟩
    \__kernel_backend_postscript:n { /color.sc ~ { } ~ def }
⟨/dvips⟩
  }
\cs_new_protected:Npn \__color_backend_reset:
  { \__kernel_backend_literal:n { color~pop } }
```

(*End definition for* `\__color_backend_select_cmyk:n` *and others. This function is documented on page* **??**.)

```
⟨/dvips | dvisvgm⟩
```

### 3.3.2 LuaTEX and pdfTEX

```
⟨*luatex | pdftex⟩
```

\l__color_backend_fill_tl
\l__color_backend_stroke_tl

```
\tl_new:N \l__color_backend_fill_tl
\tl_new:N \l__color_backend_stroke_tl
```

(*End definition for* `\l__color_backend_fill_tl` *and* `\l__color_backend_stroke_tl`.)

__color_backend_select_cmyk:n
__color_backend_select_gray:n
__color_backend_select_rgb:n
__color_backend_select:nn
__color_backend_reset:

Store the values then pass to the stack.

```
\cs_new_protected:Npn \__color_backend_select_cmyk:n #1
  { \__color_backend_select:nn { #1 ~ k } { #1 ~ K } }
\cs_new_protected:Npn \__color_backend_select_gray:n #1
  { \__color_backend_select:nn { #1 ~ g } { #1 ~ G } }
\cs_new_protected:Npn \__color_backend_select_rgb:n #1
  { \__color_backend_select:nn { #1 ~ rg } { #1 ~ RG } }
\cs_new_protected:Npn \__color_backend_select:nn #1#2
```

```
  {
    \tl_set:Nn \l__color_backend_fill_tl {#1}
    \tl_set:Nn \l__color_backend_stroke_tl {#2}
    \__kernel_color_backend_stack_push:nn \l__color_backend_stack_int { #1 ~ #2 }
  }
\cs_new_protected:Npn \__color_backend_reset:
  { \__kernel_color_backend_stack_pop:n \l__color_backend_stack_int }
```

(*End definition for* `\__color_backend_select_cmyk:n` *and others.*)

```
⟨/luatex | pdftex⟩
```

### 3.3.3 `dvipmdfx`/XƎTEX

These backends have the most possible approaches: it recognises both `dvips`-based color specials and its own format, plus one can include PDF statements directly. Recent releases also have a color stack approach similar to pdfTEX. Of the stack methods, the dedicated the most versatile is the latter as it can cover all of the use cases we have. However, at present this interacts problematically with any color on the original stack. We therefore stick to a single-stack approach here.

```
⟨*dvipdfmx | xetex⟩
```

`\__color_backend_select:n`
`\__color_backend_select_cmyk:n`
`\__color_backend_select_gray:n`
`\__color_backend_select_rgb:n`
`\__color_backend_reset:`

Using the single stack is relatively easy as there is only one route.

```
\cs_new_protected:Npn \__color_backend_select:n #1
  { \__kernel_backend_literal:n { pdf : bc ~ [ #1 ] } }
\cs_new_eq:NN \__color_backend_select_cmyk:n \__color_backend_select:n
\cs_new_eq:NN \__color_backend_select_gray:n \__color_backend_select:n
\cs_new_eq:NN \__color_backend_select_rgb:n  \__color_backend_select:n
\cs_new_protected:Npn \__color_backend_reset:
  { \__kernel_backend_literal:n { pdf : ec } }
```

(*End definition for* `\__color_backend_select:n` *and others.*)

`\__color_backend_select_named:n`

For classical named colors, the only value we should get is `Black`.

```
\cs_new_protected:Npn \__color_backend_select_named:n #1
  {
    \str_if_eq:nnTF {#1} { Black }
      { \__color_backend_select_gray:n { 0 } }
      { \msg_error:nnn { color } { unknown-named-color } {#1} }
  }
\msg_new:nnn { color } { unknown-named-color }
  { Named~color~'#1'~is~not~known. }
```

(*End definition for* `\__color_backend_select_named:n`.)

```
⟨/dvipdfmx | xetex⟩
```

## 3.4 Separations

Here, life gets interesting and we need essentially one approach per backend.

```
⟨*dvipdfmx | luatex | pdftex | xetex | dvips⟩
```

But we start with some functionality needed for both PostScript and PDF based backends.

\g__color_backend_colorant_prop

```
\prop_new:N \g__color_backend_colorant_prop
```

(*End definition for* `\g__color_backend_colorant_prop`*.*)

\__color_backend_devicen_colorants:n
\__color_backend_devicen_colorants:w

```
\cs_new:Npx \__color_backend_devicen_colorants:n #1
  {
    \exp_not:N \tl_if_blank:nF {#1}
      {
        \c_space_tl
        << ~
          /Colorants ~
            << ~
              \exp_not:N \__color_backend_devicen_colorants:w #1 ~
                \exp_not:N \q_recursion_tail \c_space_tl
                \exp_not:N \q_recursion_stop
            >> ~
        >>
      }
  }
\cs_new:Npn \__color_backend_devicen_colorants:w #1 ~
  {
    \quark_if_recursion_tail_stop:n {#1}
    \prop_if_in:NnT \g__color_backend_colorant_prop {#1}
      {
        #1 ~
        \prop_item:Nn \g__color_backend_colorant_prop {#1} ~
      }
    \__color_backend_devicen_colorants:w
  }
```

(*End definition for* `\__color_backend_devicen_colorants:n` *and* `\__color_backend_devicen_colorants:w`*.*)

```
⟨/dvipdfmx | luatex | pdftex | xetex | dvips⟩
⟨*dvips⟩
```

\__color_backend_select_separation:nn
\__color_backend_select_devicen:nn

```
\cs_new_protected:Npn \__color_backend_select_separation:nn #1#2
  { \__color_backend_select:n { separation ~ #1 ~ #2 } }
\cs_new_eq:NN \__color_backend_select_devicen:nn \__color_backend_select_separation:nn
```

(*End definition for* `\__color_backend_select_separation:nn` *and* `\__color_backend_select_devicen:nn`*.*)

\__color_backend_select_iccbased:nn

No support.

```
\cs_new_protected:Npn \__color_backend_select_iccbased:nn #1#2 { }
```

(*End definition for* `\__color_backend_select_iccbased:nn`*.*)

\__color_backend_separation_init:nnnnn
\__color_backend_separation_init:nxxnn
\__color_backend_separation_init_aux:nnnnnn
lor_backend_separation_init_/DeviceCMYK:nnn
lor_backend_separation_init_/DeviceGray:nnn
olor_backend_separation_init_/DeviceRGB:nnn
\__color_backend_separation_init_Device:Nn
\__color_backend_separation_init:nnn
\__color_backend_separation_init_count:n
\__color_backend_separation_init_count:w
\__color_backend_separation_init:nnnn
\__color_backend_separation_init:w
\__color_backend_separation_init:n
\__color_backend_separation_init:nw
\__color_backend_separation_init_CIELAB:nnn

Initialising here means creating a small header set up plus massaging some data. This comes about as we have to deal with PDF-focussed data, which makes most sense "higher-up". The approach is based on ideas from https://tex.stackexchange.com/q/560093 plus using the PostScript manual for other aspects.

```
\cs_new_protected:Npx \__color_backend_separation_init:nnnnn #1#2#3#4#5
  {
    \bool_if:NT \g__kernel_backend_header_bool
      {
        \exp_args:Nx \__kernel_backend_first_shipout:n
          {
            \exp_not:N \__color_backend_separation_init_aux:nnnnnn
              { \exp_not:N \int_use:N \g__color_model_int }
              {#1} {#2} {#3} {#4} {#5}
          }
        \prop_gput:Nxx \exp_not:N \g__color_backend_colorant_prop
          { / \exp_not:N \str_convert_pdfname:n {#1} }
          {
            << ~
              /setcolorspace ~ {} ~
            >> ~ begin ~
              color \exp_not:N \int_use:N \g__color_model_int \c_space_tl
            end
          }
      }
  }
\cs_generate_variant:Nn \__color_backend_separation_init:nnnnn { nxx }
\cs_new_protected:Npn \__color_backend_separation_init_aux:nnnnnn #1#2#3#4#5#6
  {
    \__kernel_backend_literal:e
      {
        !
        TeXDict ~ begin ~
        /color #1
          {
            [ ~
              /Separation ~ ( \str_convert_pdfname:n {#2} ) ~
              [ ~ #3 ~ ] ~
                {
                  \cs_if_exist_use:cF { __color_backend_separation_init_ #3 :nnn }
                    { \__color_backend_separation_init:nnn }
                      {#4} {#5} {#6}
                }
            ] ~ setcolorspace
          } ~ def ~
        end
      }
  }
\cs_new:cpn { __color_backend_separation_init_ /DeviceCMYK :nnn } #1#2#3
  { \__color_backend_separation_init_Device:Nn 4 {#3} }
\cs_new:cpn { __color_backend_separation_init_ /DeviceGray :nnn } #1#2#3
  { \__color_backend_separation_init_Device:Nn 1 {#3} }
\cs_new:cpn { __color_backend_separation_init_ /DeviceRGB :nnn } #1#2#3
  { \__color_backend_separation_init_Device:Nn 2 {#3} }
\cs_new:Npn \__color_backend_separation_init_Device:Nn #1#2
  {
    #2 ~
    \prg_replicate:nn {#1}
      { #1 ~ index ~ mul ~ #1 ~ 1 ~ roll ~ }
    \int_eval:n { #1 + 1 } ~ -1 ~ roll ~ pop
  }
```

For the generic case, we cannot use `/FunctionType 2` unfortunately, so we have to code that idea up in PostScript. Here, we will therefore assume that a range is *always* given. First, we count values in each argument: at the backend level, we can assume there are always well-behaved with spaces present.

```
\cs_new:Npn \__color_backend_separation_init:nnn #1#2#3
  {
   \exp_args:Ne \__color_backend_separation_init:nnnn
     { \__color_backend_separation_init_count:n {#2} }
     {#1} {#2} {#3}
  }
\cs_new:Npn \__color_backend_separation_init_count:n #1
  { \int_eval:n { 0 \__color_backend_separation_init_count:w #1 ~ \s__color_stop } }
\cs_new:Npn \__color_backend_separation_init_count:w #1 ~ #2 \s__color_stop
  {
    +1
    \tl_if_blank:nF {#2}
      { \__color_backend_separation_init_count:w #2 \s__color_stop }
  }
```

Now we implement the algorithm. In the terms in the PostScript manual, we have $\mathbf{N} = 1$ and $\mathbf{Domain} = [0\ 1]$, with **Range** as `#2`, **C0** as `#3` and **C1** as `#4`, with the number of output components in `#1`. So all we have to do is implement $y_i = \mathbf{C0}_i + x(\mathbf{C1}_i - \mathbf{C0}_i)$ with lots of stack manipulation, then check the ranges. That's done by adding everything to the stack first, then using the fact we know all of the offsets. As manipulating the stack is tricky, we start by re-formatting the **C0** and **C1** arrays to be interleaved, and add a `0` to each pair: this is used to keep the stack of constant length while we are doing the first pass of mathematics. We then working through that list, calculating from the last to the first value before tidying up by removing all of the input values. We do that by first copying all of the final $y$ values to the end of the stack, then rolling everything so we can pop the now-unneeded material.

```
\cs_new:Npn \__color_backend_separation_init:nnnn #1#2#3#4
  {
    \__color_backend_separation_init:w #3 ~ \s__color_stop #4 ~ \s__color_stop
    \prg_replicate:nn {#1}
      {
        pop ~ 1 ~ index ~ neg ~ 1 ~ index ~ add ~
        \int_eval:n { 3 * #1 } ~ index ~ mul ~
        2 ~ index ~ add ~
        \int_eval:n { 3 * #1 } ~ #1 ~ roll ~
      }
    \int_step_function:nnnN {#1} { -1 } { 1 }
      \__color_backend_separation_init:n
    \int_eval:n { 4 * #1 + 1 } ~ #1 ~ roll ~
    \prg_replicate:nn { 3 * #1 + 1 } { pop ~ }
    \tl_if_blank:nF {#2}
      { \__color_backend_separation_init:nw {#1} #2 ~ \s__color_stop }
  }
\cs_new:Npn \__color_backend_separation_init:w
  #1 ~ #2 \s__color_stop #3 ~ #4 \s__color_stop
  {
    #1 ~ #3 ~ 0 ~
    \tl_if_blank:nF {#2}
      { \__color_backend_separation_init:w #2 \s__color_stop #4 \s__color_stop }
  }
\cs_new:Npn \__color_backend_separation_init:n #1
  { \int_eval:n { #1 * 2 } ~ index ~ }
```

Finally, we deal with the range limit if required. This is handled by splitting the range into pairs. It's then just a question of doing the comparisons, this time dropping everything except the desired result.

```
\cs_new:Npn \__color_backend_separation_init:nw #1#2 ~ #3 ~ #4 \s__color_stop
  {
     #2 ~ #3 ~
     2 ~ index ~ 2 ~ index ~ lt ~
       { ~ pop ~ exch ~ pop ~ } ~
       { ~
         2 ~ index ~ 1 ~ index ~ gt ~
           { ~ exch ~ pop ~ exch ~ pop ~ } ~
           { ~ pop ~ pop ~ } ~
         ifelse ~
       }
    ifelse ~
    #1 ~ 1 ~ roll ~
    \tl_if_blank:nF {#4}
      { \__color_backend_separation_init:nw {#1} #4 \s__color_stop }
  }
```

CIELAB support uses the detail from the PostScript reference, page 227; other than that block of PostScript, this is the same as for PDF-based routes.

```
\cs_new_protected:Npn \__color_backend_separation_init_CIELAB:nnn #1#2#3
  {
    \__color_backend_separation_init:nxxnn
      {#2}
      {
        /CIEBasedABC ~
            << ~
              /RangeABC ~ [ ~ \c__color_model_range_CIELAB_tl \c_space_tl ] ~
              /DecodeABC ~
                [ ~
                  { ~ 16 ~ add ~ 116 ~ div ~ } ~ bind ~
                  { ~ 500 ~ div ~ } ~ bind ~
                  { ~ 200 ~ div ~ } ~ bind ~
                ] ~
              /MatrixABC ~ [ ~ 1 ~ 1 ~ 1 ~ 1 ~ 0 ~ 0 ~ 0 ~ 0 ~ -1 ~ ] ~
              /DecodeLMN ~
                [ ~
                  { ~
                    dup ~ 6 ~ 29 ~ div ~ ge ~
                      { ~ dup ~ dup ~ mul ~ mul ~ ~ } ~
                      { ~ 4 ~ 29 ~ div ~ sub ~ 108 ~ 841 ~ div ~ mul ~ } ~
                    ifelse ~
                    0.9505 ~ mul ~
                  } ~ bind ~
                  { ~
                    dup ~ 6 ~ 29 ~ div ~ ge ~
                      { ~ dup ~ dup ~ mul ~ mul ~ } ~
                      { ~ 4 ~ 29 ~ div ~ sub ~ 108 ~ 841 ~ div ~ mul ~ } ~
                    ifelse ~
                  } ~ bind ~
                  { ~
                    dup ~ 6 ~ 29 ~ div ~ ge ~
                      { ~ dup ~ dup ~ mul ~ mul ~ } ~
                      { ~ 4 ~ 29 ~ div ~ sub ~ 108 ~ 841 ~ div ~ mul ~ } ~
                    ifelse ~
                    1.0890 ~ mul ~
                  } ~ bind
                ] ~
              /WhitePoint ~
                [ ~ \tl_use:c { c__color_model_whitepoint_CIELAB_ #1 _tl } ~ ] ~
            >>
      }
      { \c__color_model_range_CIELAB_tl }
      { 100 ~ 0 ~ 0 }
      {#3}
  }
```

(*End definition for* `\__color_backend_separation_init:nnnnn` *and others.*)

`\__color_backend_devicen_init:nnn`

Trivial as almost all of the work occurs in the shared code.

```
\cs_new_protected:Npn \__color_backend_devicen_init:nnn #1#2#3
  {
    \__kernel_backend_literal:e
      {
        !
        TeXDict ~ begin ~
        /color \int_use:N \g__color_model_int
          {
            [ ~
              /DeviceN ~
              [ ~ #1 ~ ] ~
              #2 ~
              { ~ #3 ~ } ~
              \__color_backend_devicen_colorants:n {#1}
            ] ~ setcolorspace
          } ~ def ~
        end
      }
  }
```

(*End definition for* `\__color_backend_devicen_init:nnn`*.*)

`\__color_backend_iccbased_init:nnn`

No support at present.

```
\cs_new_protected:Npn \__color_backend_iccbased_init:nnn #1#2#3 { }
```

(*End definition for* `\__color_backend_iccbased_init:nnn`*.*)

```
⟨/dvips⟩
⟨*dvisvgm⟩
```

`\__color_backend_select_separation:nn`
`\__color_backend_select_devicen:nn`

No support at present.

```
\cs_new_protected:Npn \__color_backend_select_separation:nn #1#2 { }
\cs_new_eq:NN \__color_backend_select_devicen:nn \__color_backend_select_separation:nn
```

(*End definition for* `\__color_backend_select_separation:nn` *and* `\__color_backend_select_devicen:nn`*.*)

`\__color_backend_separation_init:nnnnn`
`\__color_backend_separation_init_CIELAB:nnn`

No support at present.

```
\cs_new_protected:Npn \__color_backend_separation_init:nnnnn #1#2#3#4#5 { }
\cs_new_protected:Npn \__color_backend_separation_init_CIELAB:nnnnnn #1#2#3 { }
```

(*End definition for* `\__color_backend_separation_init:nnnnn` *and* `\__color_backend_separation_init_CIELAB:nnn`*.*)

`\__color_backend_select_iccbased:nn`

As detailed in https://www.w3.org/TR/css-color-4/#at-profile, we can apply a color profile using CSS. As we have a local file, we use a relative URL.

```
\cs_new_protected:Npn \__color_backend_select_iccbased:nn #1#2
  {
    \__kernel_backend_literal_svg:x
      {
        <style>
          @color-profile ~
            \str_if_eq:nnTF {#2} { cmyk }
              { device-cmyk }
              { --color \int_use:N \g__color_model_int }
                \c_space_tl
            {
              src:("#1")
            }
        </style>
      }
  }
```

(*End definition for* `\__color_backend_select_iccbased:nn`*.*)

```
⟨/dvisvgm⟩
```

```
⟨*dvipdfmx | luatex | pdftex | xetex⟩
```

`\__color_backend_select_separation:nn`
`\__color_backend_select_devicen:nn`
`\__color_backend_select_iccbased:nn`

```
⟨*dvipdfmx | xetex⟩
\cs_new_protected:Npn \__color_backend_select_separation:nn #1#2
  { \__kernel_backend_literal:x { pdf : bc ~ \pdf_object_ref:n {#1} ~ [ #2 ] } }
⟨/dvipdfmx | xetex⟩
⟨*luatex | pdftex⟩
\cs_new_protected:Npn \__color_backend_select_separation:nn #1#2
  { \__color_backend_select:nn { /#1 ~ cs ~ #2 ~ scn  } { /#1 ~ CS ~ #2 ~ SCN } }
⟨/luatex | pdftex⟩
\cs_new_eq:NN \__color_backend_select_devicen:nn \__color_backend_select_separation:nn
\cs_new_eq:NN \__color_backend_select_iccbased:nn \__color_backend_select_separation:nn
```

(*End definition for* `\__color_backend_select_separation:nn`*,* `\__color_backend_select_devicen:nn`*, and* `\__color_backend_select_iccbased:nn`*.*)

`\__color_backend_init_resource:n`

Resource initiation comes up a few times. For `dvipdfmx`/XeTeX, we skip this as at present it's handled by the backend.

```
\cs_new_protected:Npn \__color_backend_init_resource:n #1
  {
⟨*luatex | pdftex⟩
    \bool_lazy_and:nnT
      { \cs_if_exist_p:N \pdfmanagement_if_active_p: }
      { \pdfmanagement_if_active_p: }
      {
        \use:x
          {
            \pdfmanagement_add:nnn
              { Page / Resources / ColorSpace }
              { #1 }
              { \pdf_object_ref_last: }
          }
      }
⟨/luatex | pdftex⟩
  }
```

(*End definition for* `\__color_backend_init_resource:n`.)

`\__color_backend_separation_init:nnnnn`
`\__color_backend_separation_init:nn`
`\__color_backend_separation_init_CIELAB:nnn`

Initialising the PDF structures needs two parts: creating an object containing the "real" name of the Separation, then adding a reference to that to each page. We use a separate object for the tint transformation following the model in the PDF reference. The object here for the color needs to be named as that way it's accessible to `dvipdfmx`/XeTeX.

```
\cs_new_protected:Npn \__color_backend_separation_init:nnnnn #1#2#3#4#5
  {
    \pdf_object_unnamed_write:nx { dict }
      {
        /FunctionType ~ 2
        /Domain ~ [0 ~ 1]
        \tl_if_blank:nF {#3} { /Range ~ [#3] }
        /C0 ~ [#4] ~
        /C1 ~ [#5] /N ~ 1
      }
    \exp_args:Nx \__color_backend_separation_init:nn
      { \str_convert_pdfname:n {#1} } {#2}
    \__color_backend_init_resource:n { color \int_use:N \g__color_model_int }
  }
\cs_new_protected:Npn \__color_backend_separation_init:nn #1#2
  {
    \use:x
      {
        \pdf_object_new:n { color \int_use:N \g__color_model_int }
        \pdf_object_write:nnn { color \int_use:N \g__color_model_int } { array }
          { /Separation /#1 ~ #2 ~ \pdf_object_ref_last: }
      }
    \prop_gput:Nnx \g__color_backend_colorant_prop { /#1 }
      { \pdf_object_ref_last: }
  }
```

For CIELAB colors, we need one object per document for the illuminant, plus initialisation of the color space referencing that object.

```
\cs_new_protected:Npn \__color_backend_separation_init_CIELAB:nnn #1#2#3
  {
    \pdf_object_if_exist:nF { __color_illuminant_CIELAB_ #1 }
      {
        \pdf_object_new:n { __color_illuminant_CIELAB_ #1 }
        \pdf_object_write:nnx { __color_illuminant_CIELAB_ #1 } { array }
          {
            /Lab ~
            <<
             /WhitePoint ~
               [ \tl_use:c { c__color_model_whitepoint_CIELAB_ #1 _tl } ]
             /Range ~ [ \c__color_model_range_CIELAB_tl ]
            >>
          }
      }
    \__color_backend_separation_init:nnnnn
      {#2}
      { \pdf_object_ref:n { __color_illuminant_CIELAB_ #1 } }
      { \c__color_model_range_CIELAB_tl }
      { 100 ~ 0 ~ 0 }
      {#3}
  }
```

(*End definition for* `\__color_backend_separation_init:nnnnn`, `\__color_backend_separation_init:nn`, *and* `\__color_backend_separation_init_CIELAB:nnn`.)

`\__color_backend_devicen_init:nnn`
`\__color_backend_devicen_init:w`

Similar to the Separations case, but with an arbitrary function for the alternative space work.

```
\cs_new_protected:Npn \__color_backend_devicen_init:nnn #1#2#3
  {
    \pdf_object_unnamed_write:nx { stream }
      {
        {
          /FunctionType ~ 4 ~
          /Domain ~
            [ ~
              \prg_replicate:nn
                { 0 \__color_backend_devicen_init:w #1 ~ \s__color_stop }
                { 0 ~ 1 ~ }
            ] ~
          /Range ~
            [ ~
              \str_case:nn {#2}
                {
                  { /DeviceCMYK } { 0 ~ 1 ~ 0 ~ 1 ~ 0 ~ 1 ~ 0 ~ 1 }
                  { /DeviceGray } { 0 ~ 1 }
                  { /DeviceRGB }  { 0 ~ 1 ~ 0 ~ 1 ~ 0 ~ 1 }
                } ~
            ]
        }
        { {#3} }
     }
    \use:x
      {
        \pdf_object_new:n { color \int_use:N \g__color_model_int }
        \pdf_object_write:nnn { color \int_use:N \g__color_model_int } { array }
          {
            /DeviceN ~
            [ ~ #1 ~ ] ~
            #2 ~
            \pdf_object_ref_last:
```

```
            \__color_backend_devicen_colorants:n {#1}
          }
      }
    \__color_backend_init_resource:n { color \int_use:N \g__color_model_int }
  }
\cs_new:Npn \__color_backend_devicen_init:w #1 ~ #2 \s__color_stop
  {
    + 1
    \tl_if_blank:nF {#2}
      { \__color_backend_devicen_init:w #2 \s__color_stop }
  }
```

(*End definition for* `\__color_backend_devicen_init:nnn` *and* `\__color_backend_devicen_init:w`.)

`\__color_backend_iccbased_init:nnn`

Lots of data to save here: we only want to do that once per file, so track it by name.

```
\cs_new_protected:Npn \__color_backend_iccbased_init:nnn #1#2#3
  {
    \pdf_object_if_exist:nF { __color_icc_ #1 }
      {
        \pdf_object_new:n { __color_icc_ #1 }
        \pdf_object_write:nnx { __color_icc_ #1 } { fstream }
          {
            {
              /N ~ \exp_not:n { #2 } ~
              \tl_if_empty:nF { #3 } { /Range~[ #3 ] }
            }
            {#1}
          }
      }
    \pdf_object_unnamed_write:nx { array }
      { /ICCBased ~ \pdf_object_ref:n { __color_icc_ #1 } }
    \__color_backend_init_resource:n { color \int_use:N \g__color_model_int }
  }
```

(*End definition for* `\__color_backend_iccbased_init:nnn`.)

`\__color_backend_iccbased_device:nnn`

This is very similar to setting up a color space: the only part we add to the page resources differently.

```
\cs_new_protected:Npn \__color_backend_iccbased_device:nnn #1#2#3
  {
    \pdf_object_if_exist:nF { __color_icc_ #1 }
      {
        \pdf_object_new:n { __color_icc_ #1 }
        \pdf_object_write:nnn { __color_icc_ #1 } { fstream }
          {
            { /N ~ #3 }
            {#1}
          }
      }
    \pdf_object_unnamed_write:nx { array }
      { /ICCBased ~ \pdf_object_ref:n { __color_icc_ #1 } }
    \__color_backend_init_resource:n { Default #2 }
  }
```

(*End definition for* `\__color_backend_iccbased_device:nnn`.)

```
⟨/dvipdfmx | luatex | pdftex | xetex⟩
```

### 3.5 Fill and stroke color

Here, `dvipdfmx`/XeTeX we write direct PDF specials for the fill, and only use the stack for the stroke color (see above for comments on why we cannot use multiple stacks with these backends). LuaTeX and pdfTeX have mutiple stacks that can deal with fill and stroke. For `dvips` we have to manage fill and stroke color ourselves. We also handle `dvisvgm` independently, as there we can create SVG directly.

```
⟨*dvipdfmx | xetex⟩
```

__color_backend_fill:n
__color_backend_fill_cmyk:n
__color_backend_fill_gray:n
__color_backend_fill_rgb:n
__color_backend_stroke:n
__color_backend_stroke_cmyk:n
__color_backend_stroke_gray:n
__color_backend_stroke_rgb:n

```
\cs_new_protected:Npn \__color_backend_fill:n #1
  { \__kernel_backend_literal:n { pdf : bc ~ fill ~ [ #1 ] } }
\cs_new_eq:NN \__color_backend_fill_cmyk:n \__color_backend_fill:n
\cs_new_eq:NN \__color_backend_fill_gray:n \__color_backend_fill:n
\cs_new_eq:NN \__color_backend_fill_rgb:n  \__color_backend_fill:n
\cs_new_protected:Npn \__color_backend_stroke:n #1
  { \__kernel_backend_literal:n { pdf : bc ~ stroke ~ [ #1 ] } }
\cs_new_eq:NN \__color_backend_stroke_cmyk:n \__color_backend_stroke:n
\cs_new_eq:NN \__color_backend_stroke_gray:n \__color_backend_stroke:n
\cs_new_eq:NN \__color_backend_stroke_rgb:n  \__color_backend_stroke:n
```

(*End definition for* `\__color_backend_fill:n` *and others.*)

__color_backend_fill_separation:nn
__color_backend_stroke_separation:nn
__color_backend_fill_devicen:nn
__color_backend_stroke_devicen:nn

```
\cs_new_protected:Npn \__color_backend_fill_separation:nn #1#2
  {
    \__kernel_backend_literal:x
      { pdf : bc ~ fill ~ \pdf_object_ref:n {#1} ~ [ #2 ] }
  }
\cs_new_protected:Npn \__color_backend_stroke_separation:nn #1#2
  {
    \__kernel_backend_literal:x
      { pdf : bc ~ stroke ~ \pdf_object_ref:n {#1} ~ [ #2 ] }
  }
\cs_new_eq:NN \__color_backend_fill_devicen:nn \__color_backend_fill_separation:nn
\cs_new_eq:NN \__color_backend_stroke_devicen:nn \__color_backend_stroke_separation:nn
```

(*End definition for* `\__color_backend_fill_separation:nn` *and others.*)

__color_backend_fill_reset:
__color_backend_stroke_reset:

```
\cs_new_eq:NN \__color_backend_fill_reset: \__color_backend_reset:
\cs_new_eq:NN \__color_backend_stroke_reset: \__color_backend_reset:
```

(*End definition for* `\__color_backend_fill_reset:` *and* `\__color_backend_stroke_reset:`*.*)

```
⟨/dvipdfmx | xetex⟩
⟨*luatex | pdftex⟩
```

__color_backend_fill_cmyk:n
__color_backend_fill_gray:n
__color_backend_fill_rgb:n
__color_backend_fill:n
__color_backend_stroke_cmyk:n
__color_backend_stroke_gray:n
__color_backend_stroke_rgb:n
__color_backend_stroke:n

Drawing (fill/stroke) color is handled in `dvipdfmx`/XeTeX in the same way as LuaTeX/pdfTeX. We use the same approach as earlier, except the color stack is not involved so the generic direct PDF operation is used. There is no worry about the nature of strokes: everything is handled automatically.

```
\cs_new_protected:Npn \__color_backend_fill_cmyk:n #1
  { \__color_backend_fill:n { #1 ~ k } }
```

```
\cs_new_protected:Npn \__color_backend_fill_gray:n #1
  { \__color_backend_fill:n { #1 ~ g } }
\cs_new_protected:Npn \__color_backend_fill_rgb:n #1
  { \__color_backend_fill:n { #1 ~ rg } }
\cs_new_protected:Npn \__color_backend_fill:n #1
  {
    \tl_set:Nn \l__color_backend_fill_tl {#1}
    \__kernel_color_backend_stack_push:nn \l__color_backend_stack_int
      { #1 ~ \l__color_backend_stroke_tl }
  }
\cs_new_protected:Npn \__color_backend_stroke_cmyk:n #1
  { \__color_backend_stroke:n { #1 ~ K } }
\cs_new_protected:Npn \__color_backend_stroke_gray:n #1
  { \__color_backend_stroke:n { #1 ~ G } }
\cs_new_protected:Npn \__color_backend_stroke_rgb:n #1
  { \__color_backend_stroke:n { #1 ~ RG } }
\cs_new_protected:Npn \__color_backend_stroke:n #1
  {
    \tl_set:Nn \l__color_backend_stroke_tl {#1}
    \__kernel_color_backend_stack_push:nn \l__color_backend_stack_int
      { \l__color_backend_fill_tl \c_space_tl #1 }
  }
```

(*End definition for* `\__color_backend_fill_cmyk:n` *and others.*)

`\__color_backend_fill_separation:nn`
`\__color_backend_stroke_separation:nn`
`\__color_backend_fill_devicen:nn`
`\__color_backend_stroke_devicen:nn`

```
\cs_new_protected:Npn \__color_backend_fill_separation:nn #1#2
  { \__color_backend_fill:n { /#1 ~ cs ~ #2 ~ scn } }
\cs_new_protected:Npn \__color_backend_stroke_separation:nn #1#2
  { \__color_backend_stroke:n { /#1 ~ CS ~ #2 ~ SCN } }
\cs_new_eq:NN \__color_backend_fill_devicen:nn \__color_backend_fill_separation:nn
\cs_new_eq:NN \__color_backend_stroke_devicen:nn \__color_backend_stroke_separation:nn
```

(*End definition for* `\__color_backend_fill_separation:nn` *and others.*)

**`\__color_backend_fill_reset:`**
`\__color_backend_stroke_reset:`

```
\cs_new_eq:NN \__color_backend_fill_reset: \__color_backend_reset:
\cs_new_eq:NN \__color_backend_stroke_reset: \__color_backend_reset:
```

(*End definition for* `\__color_backend_fill_reset:` *and* `\__color_backend_stroke_reset:`*.*)

⟨/luatex | pdftex⟩

⟨*dvips⟩

**`\__color_backend_fill_cmyk:n`**
**`\__color_backend_fill_gray:n`**
**`\__color_backend_fill_rgb:n`**
**`\__color_backend_fill:n`**
`\__color_backend_stroke_cmyk:n`
`\__color_backend_stroke_gray:n`
`\__color_backend_stroke_rgb:n`

Fill color here is the same as general color *except* we skip the stroke part.

```
\cs_new_protected:Npn \__color_backend_fill_cmyk:n #1
  { \__color_backend_fill:n { cmyk ~ #1 } }
\cs_new_protected:Npn \__color_backend_fill_gray:n #1
  { \__color_backend_fill:n { gray ~ #1 } }
\cs_new_protected:Npn \__color_backend_fill_rgb:n #1
  { \__color_backend_fill:n { rgb ~ #1 } }
\cs_new_protected:Npn \__color_backend_fill:n #1
  {
    \__kernel_backend_literal:n { color~push~ #1 }
  }
```

```
\cs_new_protected:Npn \__color_backend_stroke_cmyk:n #1
  { \__kernel_backend_postscript:n { /color.sc { #1 ~ setcmykcolor } def } }
\cs_new_protected:Npn \__color_backend_stroke_gray:n #1
  { \__kernel_backend_postscript:n { /color.sc { #1 ~ setgray } def } }
\cs_new_protected:Npn \__color_backend_stroke_rgb:n #1
  { \__kernel_backend_postscript:n { /color.sc { #1 ~ setrgbcolor } def } }
```

(*End definition for* `\__color_backend_fill_cmyk:n` *and others.*)

__color_backend_fill_separation:nn
__color_backend_stroke_separation:nn
__color_backend_fill_devicen:nn
__color_backend_stroke_devicen:nn

```
\cs_new_protected:Npn \__color_backend_fill_separation:nn #1#2
  { \__color_backend_fill:n { separation ~ #1 ~ #2 } }
\cs_new_protected:Npn \__color_backend_stroke_separation:nn #1#2
  { \__kernel_backend_postscript:n { /color.sc { separation ~ #1 ~ #2 } def } }
\cs_new_eq:NN \__color_backend_fill_devicen:nn \__color_backend_fill_separation:nn
\cs_new_eq:NN \__color_backend_stroke_devicen:nn \__color_backend_stroke_separation:nn
```

(*End definition for* `\__color_backend_fill_separation:nn` *and others.*)

**__color_backend_fill_reset:**
__color_backend_stroke_reset:

```
\cs_new_eq:NN \__color_backend_fill_reset: \__color_backend_reset:
\cs_new_protected:Npn \__color_backend_stroke_reset: { }
```

(*End definition for* `\__color_backend_fill_reset:` *and* `\__color_backend_stroke_reset:`*.*)

```
⟨/dvips⟩
⟨*dvisvgm⟩
```

**__color_backend_fill_cmyk:n**
**__color_backend_fill_gray:n**
**__color_backend_fill_rgb:n**
**__color_backend_fill:n**

Fill color here is the same as general color *except* we skip the stroke part.

```
\cs_new_protected:Npn \__color_backend_fill_cmyk:n #1
  { \__color_backend_fill:n { cmyk ~ #1 } }
\cs_new_protected:Npn \__color_backend_fill_gray:n #1
  { \__color_backend_fill:n { gray ~ #1 } }
\cs_new_protected:Npn \__color_backend_fill_rgb:n #1
  { \__color_backend_fill:n { rgb ~ #1 } }
\cs_new_protected:Npn \__color_backend_fill:n #1
  {
    \__kernel_backend_literal:n { color~push~ #1 }
  }
```

(*End definition for* `\__color_backend_fill_cmyk:n` *and others.*)

__color_backend_stroke_cmyk:n
__color_backend_stroke_cmyk:w
__color_backend_stroke_gray:n
__color_backend_stroke_gray_aux:n
__color_backend_stroke_rgb:n
__color_backend_stroke_rgb:w
**__color_backend:nnn**

For drawings in SVG, we use scopes for all stroke colors. That requires using `RGB` values, which luckily are easy to convert here (`cmyk` to `RGB` is a fixed function).

```
\cs_new_protected:Npn \__color_backend_stroke_cmyk:n #1
  { \__color_backend_cmyk:w #1 \s__color_stop }
\cs_new_protected:Npn \__color_backend_stroke_cmyk:w
  #1 ~ #2 ~ #3 ~ #4 \s__color_stop
  {
    \use:x
      {
        \__color_backend:nnn
          { \fp_eval:n { -100 * ( 1 - min ( 1 , #1 + #4 ) ) } }
          { \fp_eval:n { -100 * ( 1 - min ( 1 , #2 + #4 ) ) } }
          { \fp_eval:n { -100 * ( 1 - min ( 1 , #3 + #4 ) ) } }
      }
  }
\cs_new_protected:Npn \__color_backend_stroke_gray:n #1
  {
    \use:x
      {
        \__color_backend_stroke_gray_aux:n
          { \fp_eval:n { 100 * (#1) } }
      }
  }
\cs_new_protected:Npn \__color_backend_stroke_gray_aux:n #1
  { \__color_backend:nnn {#1} {#1} {#1} }
\cs_new_protected:Npn \__color_backend_stroke_rgb:n #1
  { \__color_backend_rgb:w #1 \s__color_stop }
\cs_new_protected:Npn \__color_backend_stroke_rgb:w
  #1 ~ #2 ~ #3 \s__color_stop
  {
    \use:x
      {
        \__color_backend:nnn
          { \fp_eval:n { 100 * (#1) } }
          { \fp_eval:n { 100 * (#2) } }
          { \fp_eval:n { 100 * (#3) } }
      }
  }
\cs_new_protected:Npx \__color_backend:nnn #1#2#3
  {
    \__kernel_backend_scope:n
      {
        stroke =
         "
           rgb
             (
               #1 \c_percent_str ,
               #2 \c_percent_str ,
               #3 \c_percent_str
             )
         "
      }
  }
```

(*End definition for* `\__color_backend_stroke_cmyk:n` *and others.*)

__color_backend_fill_separation:nn
__color_backend_stroke_separation:nn
__color_backend_fill_devicen:nn
__color_backend_stroke_devicen:nn

At present, these are no-ops.

```
\cs_new_protected:Npn \__color_backend_fill_separation:nn #1#2 { }
\cs_new_protected:Npn \__color_backend_stroke_separation:nn #1#2 { }
\cs_new_eq:NN \__color_backend_fill_devicen:nn \__color_backend_fill_separation:nn
\cs_new_eq:NN \__color_backend_stroke_devicen:nn \__color_backend_stroke_separation:nn
```

(*End definition for* `\__color_backend_fill_separation:nn` *and others.*)

__color_backend_fill_reset:
__color_backend_stroke_reset:

```
\cs_new_eq:NN \__color_backend_fill_reset: \__color_backend_reset:
\cs_new_protected:Npn \__color_backend_stroke_reset: { }
```

(*End definition for* `\__color_backend_fill_reset:` *and* `\__color_backend_stroke_reset:`*.*)

`\__color_backend_devicen_init:nnn`
`\__color_backend_iccbased_init:nnn`

No support at present.

```
\cs_new_protected:Npn \__color_backend_devicen_init:nnn #1#2#3 { }
\cs_new_protected:Npn \__color_backend_iccbased_init:nnn #1#2#3 { }
```

(*End definition for* `\__color_backend_devicen_init:nnn` *and* `\__color_backend_iccbased_init:nnn`*.*)

```
⟨/dvisvgm⟩
```

```
⟨/package⟩
```

# 4 l3backend-draw Implementation

```
⟨*package⟩
⟨@@=draw⟩
```

## 4.1 dvips backend

```
⟨*dvips⟩
```

`\__draw_backend_literal:n`
`\__draw_backend_literal:x`

The same as literal PostScript: same arguments about positioning apply her.

```
\cs_new_eq:NN \__draw_backend_literal:n \__kernel_backend_literal_postscript:n
\cs_generate_variant:Nn \__draw_backend_literal:n { x }
```

(*End definition for* `\__draw_backend_literal:n`*.*)

`\__draw_backend_begin:`
`\__draw_backend_end:`

The `ps::[begin]` special here deals with positioning but allows us to continue on to a matching `ps::[end]`: contrast with `ps:`, which positions but where we can't split material between separate calls. The `@beginspecial`/`@endspecial` pair are from `special.pro` and correct the scale and $y$-axis direction. In contrast to `pgf`, we don't save the current point: discussion with Tom Rokici suggested a better way to handle the necessary translations (see `\__draw_backend_box_use:Nnnnn`). (Note that `@beginspecial`/`@endspecial` forms a backend scope.) The `[begin]`/`[end]` lines are handled differently from the rest as they are conceptually different: not really drawing literals but instructions to `dvips` itself.

```
\cs_new_protected:Npn \__draw_backend_begin:
  {
    \__kernel_backend_literal:n { ps::[begin] }
    \__draw_backend_literal:n { @beginspecial }
  }
\cs_new_protected:Npn \__draw_backend_end:
  {
    \__draw_backend_literal:n { @endspecial }
    \__kernel_backend_literal:n { ps::[end] }
  }
```

(*End definition for* `\__draw_backend_begin:` *and* `\__draw_backend_end:`*.*)

`\__draw_backend_scope_begin:`
`\__draw_backend_scope_end:`

Scope here may need to contain saved definitions, so the entire memory rather than just the graphic state has to be sent to the stack.

```
\cs_new_protected:Npn \__draw_backend_scope_begin:
  { \__draw_backend_literal:n { save } }
\cs_new_protected:Npn \__draw_backend_scope_end:
  { \__draw_backend_literal:n { restore } }
```

(*End definition for* `\__draw_backend_scope_begin:` *and* `\__draw_backend_scope_end:`.)

`\__draw_backend_moveto:nn`
`\__draw_backend_lineto:nn`
`\__draw_backend_rectangle:nnnn`
`\__draw_backend_curveto:nnnnnn`

Path creation operations mainly resolve directly to PostScript primitive steps, with only the need to convert to `bp`. Notice that `x`-type expansion is included here to ensure that any variable values are forced to literals before any possible caching. There is no native rectangular path command (without also clipping, filling or stroking), so that task is done using a small amount of PostScript.

```
\cs_new_protected:Npn \__draw_backend_moveto:nn #1#2
  {
    \__draw_backend_literal:x
      {
        \dim_to_decimal_in_bp:n {#1} ~
        \dim_to_decimal_in_bp:n {#2} ~ moveto
      }
  }
\cs_new_protected:Npn \__draw_backend_lineto:nn #1#2
  {
    \__draw_backend_literal:x
      {
        \dim_to_decimal_in_bp:n {#1} ~
        \dim_to_decimal_in_bp:n {#2} ~ lineto
      }
  }
\cs_new_protected:Npn \__draw_backend_rectangle:nnnn #1#2#3#4
  {
     \__draw_backend_literal:x
       {
         \dim_to_decimal_in_bp:n {#4} ~ \dim_to_decimal_in_bp:n {#3} ~
         \dim_to_decimal_in_bp:n {#1} ~ \dim_to_decimal_in_bp:n {#2} ~
         moveto~dup~0~rlineto~exch~0~exch~rlineto~neg~0~rlineto~closepath
      }
  }
\cs_new_protected:Npn \__draw_backend_curveto:nnnnnn #1#2#3#4#5#6
  {
    \__draw_backend_literal:x
      {
        \dim_to_decimal_in_bp:n {#1} ~ \dim_to_decimal_in_bp:n {#2} ~
        \dim_to_decimal_in_bp:n {#3} ~ \dim_to_decimal_in_bp:n {#4} ~
        \dim_to_decimal_in_bp:n {#5} ~ \dim_to_decimal_in_bp:n {#6} ~
        curveto
      }
 }
```

(*End definition for* `\__draw_backend_moveto:nn` *and others.*)

`\__draw_backend_evenodd_rule:`
`\__draw_backend_nonzero_rule:`
`\g__draw_draw_eor_bool`

The even-odd rule here can be implemented as a simply switch.

```
\cs_new_protected:Npn \__draw_backend_evenodd_rule:
  { \bool_gset_true:N \g__draw_draw_eor_bool }
\cs_new_protected:Npn \__draw_backend_nonzero_rule:
  { \bool_gset_false:N \g__draw_draw_eor_bool }
\bool_new:N \g__draw_draw_eor_bool
```

(*End definition for* `\__draw_backend_evenodd_rule:`, `\__draw_backend_nonzero_rule:`, *and* `\g__draw_draw_eor_bool`.)

__draw_backend_closepath:
__draw_backend_stroke:
__draw_backend_closestroke:
__draw_backend_fill:
__draw_backend_fillstroke:
__draw_backend_clip:
__draw_backend_discardpath:
\g__draw_draw_clip_bool

Unlike PDF, PostScript doesn't track separate colors for strokes and other elements. It is also desirable to have the `clip` keyword after a stroke or fill. To achieve those outcomes, there is some work to do. For color, the stoke color is simple but the fill one has to be inserted by hand. For clipping, the required ordering is achieved using a TeX switch. All of the operations end with a new path instruction as they do not terminate (again in contrast to PDF).

```
\cs_new_protected:Npn \__draw_backend_closepath:
  { \__draw_backend_literal:n { closepath } }
\cs_new_protected:Npn \__draw_backend_stroke:
  {
    \__draw_backend_literal:n { gsave }
    \__draw_backend_literal:n { color.sc }
    \__draw_backend_literal:n { stroke }
    \__draw_backend_literal:n { grestore }
    \bool_if:NT \g__draw_draw_clip_bool
      {
        \__draw_backend_literal:x
          {
            \bool_if:NT \g__draw_draw_eor_bool { eo }
            clip
          }
      }
    \__draw_backend_literal:n { newpath }
    \bool_gset_false:N \g__draw_draw_clip_bool
  }
\cs_new_protected:Npn \__draw_backend_closestroke:
  {
    \__draw_backend_closepath:
    \__draw_backend_stroke:
  }
\cs_new_protected:Npn \__draw_backend_fill:
  {
    \__draw_backend_literal:x
      {
        \bool_if:NT \g__draw_draw_eor_bool { eo }
        fill
      }
    \bool_if:NT \g__draw_draw_clip_bool
      {
        \__draw_backend_literal:x
          {
            \bool_if:NT \g__draw_draw_eor_bool { eo }
            clip
          }
      }
    \__draw_backend_literal:n { newpath }
    \bool_gset_false:N \g__draw_draw_clip_bool
  }
\cs_new_protected:Npn \__draw_backend_fillstroke:
  {
    \__draw_backend_literal:x
      {
        \bool_if:NT \g__draw_draw_eor_bool { eo }
        fill
      }
    \__draw_backend_literal:n { gsave }
    \__draw_backend_literal:n { color.sc }
    \__draw_backend_literal:n { stroke }
    \__draw_backend_literal:n { grestore }
    \bool_if:NT \g__draw_draw_clip_bool
      {
        \__draw_backend_literal:x
          {
            \bool_if:NT \g__draw_draw_eor_bool { eo }
            clip
          }
      }
    \__draw_backend_literal:n { newpath }
    \bool_gset_false:N \g__draw_draw_clip_bool
  }
\cs_new_protected:Npn \__draw_backend_clip:
  { \bool_gset_true:N \g__draw_draw_clip_bool }
\bool_new:N \g__draw_draw_clip_bool
\cs_new_protected:Npn \__draw_backend_discardpath:
  {
    \bool_if:NT \g__draw_draw_clip_bool
      {
        \__draw_backend_literal:x
          {
            \bool_if:NT \g__draw_draw_eor_bool { eo }
            clip
          }
      }
    \__draw_backend_literal:n { newpath }
    \bool_gset_false:N \g__draw_draw_clip_bool
  }
```

(*End definition for* `\__draw_backend_closepath:` *and others.*)

`\__draw_backend_dash_pattern:nn`
`\__draw_backend_dash:n`
`\__draw_backend_linewidth:n`
`\__draw_backend_miterlimit:n`
`\__draw_backend_cap_butt:`
`\__draw_backend_cap_round:`
`\__draw_backend_cap_rectangle:`
`\__draw_backend_join_miter:`
`\__draw_backend_join_round:`
`\__draw_backend_join_bevel:`

Converting paths to output is again a case of mapping directly to PostScript operations.

```
\cs_new_protected:Npn \__draw_backend_dash_pattern:nn #1#2
  {
    \__draw_backend_literal:x
      {
        [
          \exp_args:Nf \use:n
            { \clist_map_function:nN {#1} \__draw_backend_dash:n }
        ] ~
        \dim_to_decimal_in_bp:n {#2} ~ setdash
      }
  }
\cs_new:Npn \__draw_backend_dash:n #1
  { ~ \dim_to_decimal_in_bp:n {#1} }
\cs_new_protected:Npn \__draw_backend_linewidth:n #1
  {
    \__draw_backend_literal:x
      { \dim_to_decimal_in_bp:n {#1} ~ setlinewidth }
  }
\cs_new_protected:Npn \__draw_backend_miterlimit:n #1
  { \__draw_backend_literal:n { #1 ~ setmiterlimit } }
\cs_new_protected:Npn \__draw_backend_cap_butt:
  { \__draw_backend_literal:n { 0 ~ setlinecap } }
\cs_new_protected:Npn \__draw_backend_cap_round:
  { \__draw_backend_literal:n { 1 ~ setlinecap } }
\cs_new_protected:Npn \__draw_backend_cap_rectangle:
  { \__draw_backend_literal:n { 2 ~ setlinecap } }
\cs_new_protected:Npn \__draw_backend_join_miter:
  { \__draw_backend_literal:n { 0 ~ setlinejoin } }
\cs_new_protected:Npn \__draw_backend_join_round:
  { \__draw_backend_literal:n { 1 ~ setlinejoin } }
\cs_new_protected:Npn \__draw_backend_join_bevel:
  { \__draw_backend_literal:n { 2 ~ setlinejoin } }
```

(*End definition for* `\__draw_backend_dash_pattern:nn` *and others.*)

`\__draw_backend_cm:nnnn`

In `dvips`, keeping the transformations in line with the engine is unfortunately not possible for scaling and rotations: even if we decompose the matrix into those operations, there is still no backend tracking (*cf.* `dvipdfmx`/XƎTEX). Thus we take the shortest path available and simply dump the matrix as given.

```
\cs_new_protected:Npn \__draw_backend_cm:nnnn #1#2#3#4
  {
    \__draw_backend_literal:n
      { [ #1 ~ #2 ~ #3 ~ #4 ~ 0 ~ 0 ] ~ concat }
  }
```

(*End definition for* `\__draw_backend_cm:nnnn`*.*)

`\__draw_backend_box_use:Nnnnn`

Inside a picture `@beginspecial`/`@endspecial` are active, which is normally a good thing but means that the position and scaling would be off if the box was inserted directly. To deal with that, there are a number of possible approaches. The implementation here was suggested by Tom Rokici (author of `dvips`). We end the current special placement, then set the current point with a literal `[begin]`. As for general literals, we then use the stack to store the current point and move to it. To insert the required transformation, we have to flip the $y$-axis, once before and once after it. Then we get back to the TEX reference point to insert our content. The clean up has to happen in the right places, hence the `[begin]`/`[end]` pair around `restore`. Finally, we can return to "normal" drawing mode. Notice that the set up here is very similar to that in `\__draw_align_currentpoint_...`, but the ordering of saving and restoring is different (intermixed).

```
\cs_new_protected:Npn \__draw_backend_box_use:Nnnnn #1#2#3#4#5
  {
    \__draw_backend_literal:n { @endspecial }
    \__draw_backend_literal:n { [end] }
    \__draw_backend_literal:n { [begin] }
    \__draw_backend_literal:n { save }
    \__draw_backend_literal:n { currentpoint }
    \__draw_backend_literal:n { currentpoint~translate }
    \__draw_backend_cm:nnnn { 1 } { 0 } { 0 } { -1 }
    \__draw_backend_cm:nnnn {#2} {#3} {#4} {#5}
    \__draw_backend_cm:nnnn { 1 } { 0 } { 0 } { -1 }
    \__draw_backend_literal:n { neg~exch~neg~exch~translate }
    \__draw_backend_literal:n { [end] }
    \hbox_overlap_right:n { \box_use:N #1 }
    \__draw_backend_literal:n { [begin] }
    \__draw_backend_literal:n { restore }
    \__draw_backend_literal:n { [end] }
    \__draw_backend_literal:n { [begin] }
    \__draw_backend_literal:n { @beginspecial }
  }
```

(*End definition for* `\__draw_backend_box_use:Nnnnn`.)

```
⟨/dvips⟩
```

## 4.2 LuaTeX, pdfTeX, dvipdfmx and XeTeX

LuaTeX, pdfTeX, `dvipdfmx` and XeTeX directly produce PDF output and understand a shared set of specials for drawing commands.

```
⟨*dvipdfmx | luatex | pdftex | xetex⟩
```

### 4.2.1 Drawing

`\__draw_backend_literal:n`
`\__draw_backend_literal:x`

Pass data through using a dedicated interface.

```
\cs_new_eq:NN \__draw_backend_literal:n \__kernel_backend_literal_pdf:n
\cs_generate_variant:Nn \__draw_backend_literal:n { x }
```

(*End definition for* `\__draw_backend_literal:n`.)

`\__draw_backend_begin:`
`\__draw_backend_end:`

No special requirements here, so simply set up a drawing scope.

```
\cs_new_protected:Npn \__draw_backend_begin:
  { \__draw_backend_scope_begin: }
\cs_new_protected:Npn \__draw_backend_end:
  { \__draw_backend_scope_end: }
```

(*End definition for* `\__draw_backend_begin:` *and* `\__draw_backend_end:`.)

`\__draw_backend_scope_begin:`
`\__draw_backend_scope_end:`

Use the backend-level scope mechanisms.

```
\cs_new_eq:NN \__draw_backend_scope_begin: \__kernel_backend_scope_begin:
\cs_new_eq:NN \__draw_backend_scope_end: \__kernel_backend_scope_end:
```

(*End definition for* `\__draw_backend_scope_begin:` *and* `\__draw_backend_scope_end:`.)

`\__draw_backend_moveto:nn`
`\__draw_backend_lineto:nn`
`\__draw_backend_curveto:nnnnnn`
`\__draw_backend_rectangle:nnnn`

Path creation operations all resolve directly to PDF primitive steps, with only the need to convert to `bp`.

```
\cs_new_protected:Npn \__draw_backend_moveto:nn #1#2
  {
    \__draw_backend_literal:x
      { \dim_to_decimal_in_bp:n {#1} ~ \dim_to_decimal_in_bp:n {#2} ~ m }
  }
\cs_new_protected:Npn \__draw_backend_lineto:nn #1#2
  {
    \__draw_backend_literal:x
      { \dim_to_decimal_in_bp:n {#1} ~ \dim_to_decimal_in_bp:n {#2} ~ l }
  }
\cs_new_protected:Npn \__draw_backend_curveto:nnnnnn #1#2#3#4#5#6
  {
    \__draw_backend_literal:x
      {
        \dim_to_decimal_in_bp:n {#1} ~ \dim_to_decimal_in_bp:n {#2} ~
        \dim_to_decimal_in_bp:n {#3} ~ \dim_to_decimal_in_bp:n {#4} ~
        \dim_to_decimal_in_bp:n {#5} ~ \dim_to_decimal_in_bp:n {#6} ~
        c
      }
 }
\cs_new_protected:Npn \__draw_backend_rectangle:nnnn #1#2#3#4
  {
     \__draw_backend_literal:x
      {
        \dim_to_decimal_in_bp:n {#1} ~ \dim_to_decimal_in_bp:n {#2} ~
        \dim_to_decimal_in_bp:n {#3} ~ \dim_to_decimal_in_bp:n {#4} ~
        re
      }
  }
```

(*End definition for* `\__draw_backend_moveto:nn` *and others.*)

`\__draw_backend_evenodd_rule:`
`\__draw_backend_nonzero_rule:`
`\g__draw_draw_eor_bool`

The even-odd rule here can be implemented as a simply switch.

```
\cs_new_protected:Npn \__draw_backend_evenodd_rule:
  { \bool_gset_true:N \g__draw_draw_eor_bool }
\cs_new_protected:Npn \__draw_backend_nonzero_rule:
  { \bool_gset_false:N \g__draw_draw_eor_bool }
\bool_new:N \g__draw_draw_eor_bool
```

(*End definition for* `\__draw_backend_evenodd_rule:`*,* `\__draw_backend_nonzero_rule:`*, and* `\g__draw_draw_eor_bool`*.*)

`\__draw_backend_closepath:`
`\__draw_backend_stroke:`
`\__draw_backend_closestroke:`
`\__draw_backend_fill:`
`\__draw_backend_fillstroke:`
`\__draw_backend_clip:`
`\__draw_backend_discardpath:`

Converting paths to output is again a case of mapping directly to PDF operations.

```
\cs_new_protected:Npn \__draw_backend_closepath:
  { \__draw_backend_literal:n { h } }
\cs_new_protected:Npn \__draw_backend_stroke:
  { \__draw_backend_literal:n { S } }
\cs_new_protected:Npn \__draw_backend_closestroke:
  { \__draw_backend_literal:n { s } }
\cs_new_protected:Npn \__draw_backend_fill:
  {
    \__draw_backend_literal:x
      { f \bool_if:NT \g__draw_draw_eor_bool * }
  }
\cs_new_protected:Npn \__draw_backend_fillstroke:
  {
    \__draw_backend_literal:x
      { B \bool_if:NT \g__draw_draw_eor_bool * }
  }
\cs_new_protected:Npn \__draw_backend_clip:
  {
    \__draw_backend_literal:x
      { W \bool_if:NT \g__draw_draw_eor_bool * }
  }
\cs_new_protected:Npn \__draw_backend_discardpath:
  { \__draw_backend_literal:n { n } }
```

(*End definition for* `\__draw_backend_closepath:` *and others.*)

`\__draw_backend_dash_pattern:nn`
`\__draw_backend_dash:n`
`\__draw_backend_linewidth:n`
`\__draw_backend_miterlimit:n`
`\__draw_backend_cap_butt:`
`\__draw_backend_cap_round:`
`\__draw_backend_cap_rectangle:`
`\__draw_backend_join_miter:`
`\__draw_backend_join_round:`
`\__draw_backend_join_bevel:`

Converting paths to output is again a case of mapping directly to PDF operations.

```
\cs_new_protected:Npn \__draw_backend_dash_pattern:nn #1#2
  {
    \__draw_backend_literal:x
      {
        [
          \exp_args:Nf \use:n
            { \clist_map_function:nN {#1} \__draw_backend_dash:n }
        ] ~
        \dim_to_decimal_in_bp:n {#2} ~ d
      }
  }
\cs_new:Npn \__draw_backend_dash:n #1
  { ~ \dim_to_decimal_in_bp:n {#1} }
\cs_new_protected:Npn \__draw_backend_linewidth:n #1
  {
    \__draw_backend_literal:x
      { \dim_to_decimal_in_bp:n {#1} ~ w }
  }
\cs_new_protected:Npn \__draw_backend_miterlimit:n #1
  { \__draw_backend_literal:x { #1 ~ M } }
\cs_new_protected:Npn \__draw_backend_cap_butt:
  { \__draw_backend_literal:n { 0 ~ J } }
\cs_new_protected:Npn \__draw_backend_cap_round:
  { \__draw_backend_literal:n { 1 ~ J } }
\cs_new_protected:Npn \__draw_backend_cap_rectangle:
  { \__draw_backend_literal:n { 2 ~ J } }
\cs_new_protected:Npn \__draw_backend_join_miter:
  { \__draw_backend_literal:n { 0 ~ j } }
\cs_new_protected:Npn \__draw_backend_join_round:
  { \__draw_backend_literal:n { 1 ~ j } }
\cs_new_protected:Npn \__draw_backend_join_bevel:
  { \__draw_backend_literal:n { 2 ~ j } }
```

(*End definition for* `\__draw_backend_dash_pattern:nn` *and others.*)

`\__draw_backend_cm:nnnn`
`\__draw_backend_cm_aux:nnnn`

Another split here between LuaTEX/pdfTeX and `dvipdfmx`/XƎTEX. In the former, we have a direct method to maintain alignment: the backend can use a matrix itself. For `dvipdfmx`/XƎTEX, we can to decompose the matrix into rotations and a scaling, then use those operations as they are handled by the backend. (There is backend support for matrix operations in `dvipdfmx`/XƎTEX, but as a matched pair so not suitable for the "stand alone" transformation set up here.) The specials used here are from `xdvipdfmx` originally: they are well-tested, but probably equivalent to the `pdf:` versions!

```
\cs_new_protected:Npn \__draw_backend_cm:nnnn #1#2#3#4
  {
⟨*luatex | pdftex⟩
    \__kernel_backend_matrix:n { #1 ~ #2 ~ #3 ~ #4 }
⟨/luatex | pdftex⟩
⟨*dvipdfmx | xetex⟩
    \__draw_backend_cm_decompose:nnnnN {#1} {#2} {#3} {#4}
      \__draw_backend_cm_aux:nnnn
⟨/dvipdfmx | xetex⟩
```

```
  }
⟨*dvipdfmx | xetex⟩
\cs_new_protected:Npn \__draw_backend_cm_aux:nnnn #1#2#3#4
  {
    \__kernel_backend_literal:x
      {
        x:rotate~
        \fp_compare:nNnTF {#1} = \c_zero_fp
          { 0 }
          { \fp_eval:n { round ( -#1 , 5 ) } }
      }
    \__kernel_backend_literal:x
      {
        x:scale~
        \fp_eval:n { round ( #2 , 5 ) } ~
        \fp_eval:n { round ( #3 , 5 ) }
      }
    \__kernel_backend_literal:x
      {
        x:rotate~
        \fp_compare:nNnTF {#4} = \c_zero_fp
          { 0 }
          { \fp_eval:n { round ( -#4 , 5 ) } }
      }
  }
⟨/dvipdfmx | xetex⟩
```

(*End definition for* `\__draw_backend_cm:nnnn` *and* `\__draw_backend_cm_aux:nnnn`.)

`\__draw_backend_cm_decompose:nnnnN`
`\__draw_backend_cm_decompose_auxi:nnnnN`
`\__draw_backend_cm_decompose_auxii:nnnnN`
`\__draw_backend_cm_decompose_auxiii:nnnnN`

Internally, transformations for drawing are tracked as a matrix. Not all engines provide a way of dealing with this: if we use a raw matrix, the engine looses track of positions (for example for hyperlinks), and this is not desirable. They do, however, allow us to track rotations and scalings. Luckily, we can decompose any (two-dimensional) matrix into two rotations and a single scaling:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} w_1 & 0 \\ 0 & w_2 \end{bmatrix} \begin{bmatrix} \cos\gamma & \sin\gamma \\ -\sin\gamma & \cos\gamma \end{bmatrix}$$

The parent matrix can be converted to

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} E & H \\ -H & E \end{bmatrix} + \begin{bmatrix} F & G \\ G & -F \end{bmatrix}$$

From these, we can find that

$$\begin{aligned} \frac{w_1 + w_2}{2} &= \sqrt{E^2 + H^2} \\ \frac{w_1 - w_2}{2} &= \sqrt{F^2 + G^2} \\ \gamma - \beta &= \tan^{-1}(G/F) \\ \gamma + \beta &= \tan^{-1}(H/E) \end{aligned}$$

at which point we just have to do various pieces of re-arrangement to get all of the values. (See J. Blinn, *IEEE Comput. Graph. Appl.*, 1996, **16**, 82–88.) There is one wrinkle: the

PostScript (and PDF) way of specifying a transformation matrix exchanges where one would normally expect $B$ and $C$ to be.

```
⟨*dvipdfmx | xetex⟩
\cs_new_protected:Npn \__draw_backend_cm_decompose:nnnnN #1#2#3#4#5
  {
    \use:x
      {
        \__draw_backend_cm_decompose_auxi:nnnnN
          { \fp_eval:n { (#1 + #4) / 2 } }
          { \fp_eval:n { (#1 - #4) / 2 } }
          { \fp_eval:n { (#3 + #2) / 2 } }
          { \fp_eval:n { (#3 - #2) / 2 } }
      }
        #5
  }
\cs_new_protected:Npn \__draw_backend_cm_decompose_auxi:nnnnN #1#2#3#4#5
  {
    \use:x
      {
        \__draw_backend_cm_decompose_auxii:nnnnN
          { \fp_eval:n { 2 * sqrt ( #1 * #1 + #4 * #4 ) } }
          { \fp_eval:n { 2 * sqrt ( #2 * #2 + #3 * #3 ) } }
          { \fp_eval:n { atand ( #3 , #2 ) } }
          { \fp_eval:n { atand ( #4 , #1 ) } }
      }
         #5
  }
\cs_new_protected:Npn \__draw_backend_cm_decompose_auxii:nnnnN #1#2#3#4#5
  {
    \use:x
      {
        \__draw_backend_cm_decompose_auxiii:nnnnN
          { \fp_eval:n { ( #4 - #3 ) / 2 } }
          { \fp_eval:n { ( #1 + #2 ) / 2 } }
          { \fp_eval:n { ( #1 - #2 ) / 2 } }
          { \fp_eval:n { ( #4 + #3 ) / 2 } }
      }
        #5
  }
\cs_new_protected:Npn \__draw_backend_cm_decompose_auxiii:nnnnN #1#2#3#4#5
  {
    \fp_compare:nNnTF { abs( #2 ) } > { abs ( #3 ) }
      { #5 {#1} {#2} {#3} {#4} }
      { #5 {#1} {#3} {#2} {#4} }
  }
⟨/dvipdfmx | xetex⟩
```

(*End definition for* `\__draw_backend_cm_decompose:nnnnN` *and others.*)

`\__draw_backend_box_use:Nnnnn`

Inserting a TeX box transformed to the requested position and using the current matrix is done using a mixture of TeX and low-level manipulation. The offset can be handled by TeX, so only any rotation/skew/scaling component needs to be done using the matrix operation. As this operation can never be cached, the scope is set directly not using the `draw` version.

```
\cs_new_protected:Npn \__draw_backend_box_use:Nnnnn #1#2#3#4#5
  {
    \__kernel_backend_scope_begin:
⟨*luatex | pdftex⟩
    \__draw_backend_cm:nnnn {#2} {#3} {#4} {#5}
⟨/luatex | pdftex⟩
⟨*dvipdfmx | xetex⟩
    \__kernel_backend_literal:n
      { pdf:btrans~matrix~ #2 ~ #3 ~ #4 ~ #5 ~ 0 ~ 0 }
⟨/dvipdfmx | xetex⟩
    \hbox_overlap_right:n { \box_use:N #1 }
⟨*dvipdfmx | xetex⟩
    \__kernel_backend_literal:n { pdf:etrans }
⟨/dvipdfmx | xetex⟩
    \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__draw_backend_box_use:Nnnnn`.)

```
⟨/dvipdfmx | luatex | pdftex | xetex⟩
```

## 4.3 `dvisvgm` backend

```
⟨*dvisvgm⟩
```

`\__draw_backend_literal:n`
`\__draw_backend_literal:x`

The same as the more general literal call.

```
\cs_new_eq:NN \__draw_backend_literal:n \__kernel_backend_literal_svg:n
\cs_generate_variant:Nn \__draw_backend_literal:n { x }
```

(*End definition for* `\__draw_backend_literal:n`.)

`\__draw_backend_scope_begin:`
`\__draw_backend_scope_end:`

Use the backend-level scope mechanisms.

```
\cs_new_eq:NN \__draw_backend_scope_begin: \__kernel_backend_scope_begin:
\cs_new_eq:NN \__draw_backend_scope_end: \__kernel_backend_scope_end:
```

(*End definition for* `\__draw_backend_scope_begin:` *and* `\__draw_backend_scope_end:`.)

`\__draw_backend_begin:`
`\__draw_backend_end:`

A drawing needs to be set up such that the co-ordinate system is translated. That is done inside a scope, which as described below

```
\cs_new_protected:Npn \__draw_backend_begin:
  {
    \__kernel_backend_scope_begin:
    \__kernel_backend_scope:n { transform="translate({?x},{?y})~scale(1,-1)" }
  }
\cs_new_eq:NN \__draw_backend_end: \__kernel_backend_scope_end:
```

(*End definition for* `\__draw_backend_begin:` *and* `\__draw_backend_end:`.)

`\__draw_backend_moveto:nn`
`\__draw_backend_lineto:nn`
`\__draw_backend_rectangle:nnnn`
`\__draw_backend_curveto:nnnnnn`
`\__draw_backend_add_to_path:n`
`\g__draw_backend_path_tl`

Once again, some work is needed to get path constructs correct. Rather then write the values as they are given, the entire path needs to be collected up before being output in one go. For that we use a dedicated storage routine, which adds spaces as required. Since paths should be fully expanded there is no need to worry about the internal `x`-type expansion.

```
\cs_new_protected:Npn \__draw_backend_moveto:nn #1#2
  {
    \__draw_backend_add_to_path:n
      { M ~ \dim_to_decimal:n {#1} ~ \dim_to_decimal:n {#2} }
  }
\cs_new_protected:Npn \__draw_backend_lineto:nn #1#2
  {
    \__draw_backend_add_to_path:n
      { L ~ \dim_to_decimal:n {#1} ~ \dim_to_decimal:n {#2} }
  }
\cs_new_protected:Npn \__draw_backend_rectangle:nnnn #1#2#3#4
  {
    \__draw_backend_add_to_path:n
      {
        M ~ \dim_to_decimal:n {#1} ~ \dim_to_decimal:n {#2}
        h ~ \dim_to_decimal:n {#3} ~
        v ~ \dim_to_decimal:n {#4} ~
        h ~ \dim_to_decimal:n { -#3 } ~
        Z
      }
  }
\cs_new_protected:Npn \__draw_backend_curveto:nnnnnn #1#2#3#4#5#6
  {
    \__draw_backend_add_to_path:n
      {
        C ~
        \dim_to_decimal:n {#1} ~ \dim_to_decimal:n {#2} ~
        \dim_to_decimal:n {#3} ~ \dim_to_decimal:n {#4} ~
        \dim_to_decimal:n {#5} ~ \dim_to_decimal:n {#6}
      }
  }
\cs_new_protected:Npn \__draw_backend_add_to_path:n #1
  {
    \tl_gset:Nx \g__draw_backend_path_tl
      {
        \g__draw_backend_path_tl
        \tl_if_empty:NF \g__draw_backend_path_tl { \c_space_tl }
        #1
      }
  }
\tl_new:N \g__draw_backend_path_tl
```

(*End definition for* `\__draw_backend_moveto:nn` *and others.*)

`\__draw_backend_evenodd_rule:`
`\__draw_backend_nonzero_rule:`

The fill rules here have to be handled as scopes.

```
\cs_new_protected:Npn \__draw_backend_evenodd_rule:
  { \__kernel_backend_scope:n { fill-rule="evenodd" } }
\cs_new_protected:Npn \__draw_backend_nonzero_rule:
  { \__kernel_backend_scope:n { fill-rule="nonzero" } }
```

(*End definition for* `\__draw_backend_evenodd_rule:` *and* `\__draw_backend_nonzero_rule:`*.*)

`\__draw_backend_path:n`
`\__draw_backend_closepath:`
`\__draw_backend_stroke:`
`\__draw_backend_closestroke:`
`\__draw_backend_fill:`
`\__draw_backend_fillstroke:`
`\__draw_backend_clip:`
`\__draw_backend_discardpath:`
`\g__draw_draw_clip_bool`
`\g__draw_draw_path_int`

Setting fill and stroke effects and doing clipping all has to be done using scopes. This means setting up the various requirements in a shared auxiliary which deals with the bits and pieces. Clipping paths are reused for path drawing: not essential but avoids constructing them twice. Discarding a path needs a separate function as it's not quite the same.

```
\cs_new_protected:Npn \__draw_backend_closepath:
  { \__draw_backend_add_to_path:n { Z } }
\cs_new_protected:Npn \__draw_backend_path:n #1
  {
    \bool_if:NTF \g__draw_draw_clip_bool
      {
        \int_gincr:N \g__kernel_clip_path_int
        \__draw_backend_literal:x
          {
            < clipPath~id = " l3cp \int_use:N \g__kernel_clip_path_int " >
              { ?nl }
            <path~d=" \g__draw_backend_path_tl "/> { ?nl }
            < /clipPath > { ? nl }
            <
              use~xlink:href =
                "\c_hash_str l3path \int_use:N \g__draw_backend_path_int " ~
                #1
            />
          }
        \__kernel_backend_scope:x
          {
            clip-path =
              "url( \c_hash_str l3cp \int_use:N \g__kernel_clip_path_int)"
          }
      }
      {
        \__draw_backend_literal:x
          { <path ~ d=" \g__draw_backend_path_tl " ~ #1 /> }
      }
    \tl_gclear:N \g__draw_backend_path_tl
    \bool_gset_false:N \g__draw_draw_clip_bool
  }
\int_new:N \g__draw_backend_path_int
\cs_new_protected:Npn \__draw_backend_stroke:
  { \__draw_backend_path:n { style="fill:none" } }
\cs_new_protected:Npn \__draw_backend_closestroke:
  {
    \__draw_backend_closepath:
    \__draw_backend_stroke:
  }
\cs_new_protected:Npn \__draw_backend_fill:
  { \__draw_backend_path:n { style="stroke:none" } }
\cs_new_protected:Npn \__draw_backend_fillstroke:
  { \__draw_backend_path:n { } }
\cs_new_protected:Npn \__draw_backend_clip:
  { \bool_gset_true:N \g__draw_draw_clip_bool }
\bool_new:N \g__draw_draw_clip_bool
\cs_new_protected:Npn \__draw_backend_discardpath:
  {
    \bool_if:NT \g__draw_draw_clip_bool
      {
        \int_gincr:N \g__kernel_clip_path_int
        \__draw_backend_literal:x
          {
            < clipPath~id = " l3cp \int_use:N \g__kernel_clip_path_int " >
              { ?nl }
            <path~d=" \g__draw_backend_path_tl "/> { ?nl }
            < /clipPath >
          }
        \__kernel_backend_scope:x
          {
            clip-path =
              "url( \c_hash_str l3cp \int_use:N \g__kernel_clip_path_int)"
          }
      }
    \tl_gclear:N \g__draw_path_tl
    \bool_gset_false:N \g__draw_draw_clip_bool
  }
```

(*End definition for* `\__draw_backend_path:n` *and others.*)

`\__draw_backend_dash_pattern:nn`
`\__draw_backend_dash:n`
`\__draw_backend_dash_aux:nn`
`\__draw_backend_linewidth:n`
`\__draw_backend_miterlimit:n`
`\__draw_backend_cap_butt:`
`\__draw_backend_cap_round:`
`\__draw_backend_cap_rectangle:`
`\__draw_backend_join_miter:`
`\__draw_backend_join_round:`
`\__draw_backend_join_bevel:`

All of these ideas are properties of scopes in SVG. The only slight complexity is converting the dash array properly (doing any required maths).

```
\cs_new_protected:Npn \__draw_backend_dash_pattern:nn #1#2
  {
    \use:x
      {
        \__draw_backend_dash_aux:nn
          { \clist_map_function:nN {#1} \__draw_backend_dash:n }
          { \dim_to_decimal:n {#2} }
      }
  }
\cs_new:Npn \__draw_backend_dash:n #1
  { , \dim_to_decimal_in_bp:n {#1} }
\cs_new_protected:Npn \__draw_backend_dash_aux:nn #1#2
  {
    \__kernel_backend_scope:x
      {
        stroke-dasharray =
          "
            \tl_if_empty:nTF {#1}
              { none }
              { \use_none:n #1 }
          " ~
          stroke-offset=" #2 "
      }
  }
\cs_new_protected:Npn \__draw_backend_linewidth:n #1
  { \__kernel_backend_scope:x { stroke-width=" \dim_to_decimal:n {#1} " } }
\cs_new_protected:Npn \__draw_backend_miterlimit:n #1
  { \__kernel_backend_scope:x { stroke-miterlimit=" #1 " } }
\cs_new_protected:Npn \__draw_backend_cap_butt:
  { \__kernel_backend_scope:n { stroke-linecap="butt" } }
\cs_new_protected:Npn \__draw_backend_cap_round:
  { \__kernel_backend_scope:n { stroke-linecap="round" } }
\cs_new_protected:Npn \__draw_backend_cap_rectangle:
  { \__kernel_backend_scope:n { stroke-linecap="square" } }
\cs_new_protected:Npn \__draw_backend_join_miter:
  { \__kernel_backend_scope:n { stroke-linejoin="miter" } }
\cs_new_protected:Npn \__draw_backend_join_round:
  { \__kernel_backend_scope:n { stroke-linejoin="round" } }
\cs_new_protected:Npn \__draw_backend_join_bevel:
  { \__kernel_backend_scope:n { stroke-linejoin="bevel" } }
```

(*End definition for* `\__draw_backend_dash_pattern:nn` *and others.*)

`\__draw_backend_cm:nnnn` The four arguments here are floats (the affine matrix), the last two are a displacement vector.

```
\cs_new_protected:Npn \__draw_backend_cm:nnnn #1#2#3#4
  {
    \__kernel_backend_scope:n
      {
       transform =
         " matrix ( #1 , #2 , #3 , #4 , 0pt , 0pt ) "
      }
  }
```

(*End definition for* `\__draw_backend_cm:nnnn`*.*)

`\__draw_backend_box_use:Nnnnn` No special savings can be made here: simply displace the box inside a scope. As there is nothing to re-box, just make the box passed of zero size.

```
\cs_new_protected:Npn \__draw_backend_box_use:Nnnnn #1#2#3#4#5
  {
    \__kernel_backend_scope_begin:
    \__draw_backend_cm:nnnn {#2} {#3} {#4} {#5}
    \__kernel_backend_literal_svg:n
      {
        < g~
            stroke="none"~
            transform="scale(-1,1)~translate({?x},{?y})~scale(-1,-1)"
        >
      }
    \box_set_wd:Nn #1 { 0pt }
    \box_set_ht:Nn #1 { 0pt }
    \box_set_dp:Nn #1 { 0pt }
    \box_use:N #1
    \__kernel_backend_literal_svg:n { </g> }
    \__kernel_backend_scope_end:
  }
```

(*End definition for* `\__draw_backend_box_use:Nnnnn`*.*)

```
⟨/dvisvgm⟩
```

```
⟨/package⟩
```

# 5 l3backend-graphics Implementation

```
⟨*package⟩
⟨@@=graphics⟩
```

`\__graphics_backend_loaded:n` To deal with file load ordering. Plain users are on their own.

```
\cs_new_protected:Npn \__graphics_backend_loaded:n #1
```

```
  {
    \cs_if_exist:NTF \hook_gput_code:nnn
      {
        \hook_gput_code:nnn
          { file / l3graphics.sty / after }
          { backend }
          {#1}
      }
      {#1}
  }
```

(*End definition for* `\__graphics_backend_loaded:n`*.*)

## 5.1 dvips backend

```
⟨*dvips⟩
```

`\l_graphics_search_ext_seq`

```
\__graphics_backend_loaded:n
  { \seq_set_from_clist:Nn \l_graphics_search_ext_seq { .eps , .ps } }
```

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

`\__graphics_backend_getbb_eps:n`
`\__graphics_backend_getbb_ps:n`

Simply use the generic function.

```
\__graphics_backend_loaded:n
  {
    \cs_new_eq:NN \__graphics_backend_getbb_eps:n \__graphics_read_bb:n
    \cs_new_eq:NN \__graphics_backend_getbb_ps:n \__graphics_read_bb:n
  }
```

(*End definition for* `\__graphics_backend_getbb_eps:n` *and* `\__graphics_backend_getbb_ps:n`*.*)

`\__graphics_backend_include_eps:n`
`\__graphics_backend_include_ps:n`

The special syntax is relatively clear here: remember we need PostScript sizes here.

```
\cs_new_protected:Npn \__graphics_backend_include_eps:n #1
  {
    \__kernel_backend_literal:x
      {
        PSfile = #1 \c_space_tl
        llx = \dim_to_decimal_in_bp:n \l__graphics_llx_dim \c_space_tl
        lly = \dim_to_decimal_in_bp:n \l__graphics_lly_dim \c_space_tl
        urx = \dim_to_decimal_in_bp:n \l__graphics_urx_dim \c_space_tl
        ury = \dim_to_decimal_in_bp:n \l__graphics_ury_dim
      }
  }
\cs_new_eq:NN \__graphics_backend_include_ps:n \__graphics_backend_include_eps:n
```

(*End definition for* `\__graphics_backend_include_eps:n` *and* `\__graphics_backend_include_ps:n`*.*)

`\__graphics_backend_get_pagecount:n`

```
\__graphics_backend_loaded:n
  { \cs_new_eq:NN \__graphics_backend_get_pagecount:n \__graphics_get_pagecount:n }
```

(*End definition for* `\__graphics_backend_get_pagecount:n`*.*)

```
⟨/dvips⟩
```

## 5.2 LuaTeX and pdfTeX backends

```
⟨*luatex | pdftex⟩
```

**\l_graphics_search_ext_seq**

```
\__graphics_backend_loaded:n
  {
    \seq_set_from_clist:Nn
      \l_graphics_search_ext_seq
      { .pdf , .eps , .ps , .png , .jpg , .jpeg  }
  }
```

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

\l__graphics_graphics_attr_tl

In PDF mode, additional attributes of an graphic (such as page number) are needed both to obtain the bounding box and when inserting the graphic: this occurs as the graphic dictionary approach means they are read as part of the bounding box operation. As such, it is easier to track additional attributes using a dedicated `tl` rather than build up the same data twice.

```
\tl_new:N \l__graphics_graphics_attr_tl
```

(*End definition for* `\l__graphics_graphics_attr_tl`.)

__graphics_backend_getbb_jpg:n
__graphics_backend_getbb_jpeg:n
__graphics_backend_getbb_pdf:n
__graphics_backend_getbb_png:n
__graphics_backend_getbb_auxi:n
__graphics_backend_getbb_auxii:n
__graphics_backend_getbb_auxiii:n
__graphics_backend_dequote:w

Getting the bounding box here requires us to box up the graphic and measure it. To deal with the difference in feature support in bitmap and vector graphics but keeping the common parts, there is a little work to do in terms of auxiliaries. The key here is to notice that we need two forms of the attributes: a "short" set to allow us to track for caching, and the full form to pass to the primitive.

```
\cs_new_protected:Npn \__graphics_backend_getbb_jpg:n #1
  {
    \int_zero:N \l__graphics_page_int
    \tl_clear:N \l__graphics_pagebox_tl
    \tl_set:Nx \l__graphics_graphics_attr_tl
      {
        \tl_if_empty:NF \l__graphics_decodearray_str
          { :D \l__graphics_decodearray_str }
        \bool_if:NT \l__graphics_interpolate_bool
          { :I }
      }
    \tl_clear:N \l__graphics_graphics_attr_tl
    \__graphics_backend_getbb_auxi:n {#1}
  }
\cs_new_eq:NN \__graphics_backend_getbb_jpeg:n \__graphics_backend_getbb_jpg:n
\cs_new_eq:NN \__graphics_backend_getbb_png:n \__graphics_backend_getbb_jpg:n
\cs_new_protected:Npn \__graphics_backend_getbb_pdf:n #1
  {
    \tl_clear:N \l__graphics_decodearray_str
    \bool_set_false:N \l__graphics_interpolate_bool
    \tl_set:Nx \l__graphics_graphics_attr_tl
      {
        : \l__graphics_pagebox_tl
        \int_compare:nNnT \l__graphics_page_int > 1
          { :P \int_use:N \l__graphics_page_int }
      }
    \__graphics_backend_getbb_auxi:n {#1}
  }
\cs_new_protected:Npn \__graphics_backend_getbb_auxi:n #1
  {
    \__graphics_bb_restore:xF { #1 \l__graphics_graphics_attr_tl }
      { \__graphics_backend_getbb_auxii:n {#1} }
  }
```

Measuring the graphic is done by boxing up: for PDF graphics we could use `\tex_pdfximagebbox:D`, but if doesn't work for other types. As the box always starts at $(0, 0)$ there is no need to worry about the lower-left position. Quotes need to be *removed* as LuaTEX does not like them here.

```
\cs_new_protected:Npn \__graphics_backend_getbb_auxii:n #1
  {
    \exp_args:Ne \__graphics_backend_getbb_auxiii:n
      { \__graphics_backend_dequote:w #1 " #1 " \s__graphics_stop }
    \int_const:cn { c__graphics_ #1 \l__graphics_graphics_attr_tl _int }
      { \tex_the:D \tex_pdflastximage:D }
    \__graphics_bb_save:x { #1 \l__graphics_graphics_attr_tl }
  }
\cs_new_protected:Npn \__graphics_backend_getbb_auxiii:n #1
  {
    \tex_immediate:D \tex_pdfximage:D
      \bool_lazy_or:nnT
        { \l__graphics_interpolate_bool }
        { ! \tl_if_empty_p:N \l__graphics_decodearray_str }
        {
          attr ~
            {
              \tl_if_empty:NF \l__graphics_decodearray_str
                { /Decode~[ \l__graphics_decodearray_str ] }
              \bool_if:NT \l__graphics_interpolate_bool
                { /Interpolate~true }
            }
        }
      \int_compare:nNnT \l__graphics_page_int > 0
        { page ~ \int_use:N \l__graphics_page_int }
      \tl_if_empty:NF \l__graphics_pagebox_tl
        { \l__graphics_pagebox_tl }
      {#1}
    \hbox_set:Nn \l__graphics_internal_box
      { \tex_pdfrefximage:D \tex_pdflastximage:D }
    \dim_set:Nn \l__graphics_urx_dim { \box_wd:N \l__graphics_internal_box }
    \dim_set:Nn \l__graphics_ury_dim { \box_ht:N \l__graphics_internal_box }
  }
\cs_new:Npn \__graphics_backend_dequote:w #1 " #2 " #3 \s__graphics_stop {#2}
```

(*End definition for* `\__graphics_backend_getbb_jpg:n` *and others.*)

`\__graphics_backend_include_jpg:n`
`\__graphics_backend_include_jpeg:n`
`\__graphics_backend_include_pdf:n`
`\__graphics_backend_include_png:n`

Images are already loaded for the measurement part of the code, so inclusion is straightforward, with only any attributes to worry about. The latter carry through from determination of the bounding box.

```
\cs_new_protected:Npn \__graphics_backend_include_jpg:n #1
  {
    \tex_pdfrefximage:D
      \int_use:c { c__graphics_ #1 \l__graphics_graphics_attr_tl _int }
  }
\cs_new_eq:NN \__graphics_backend_include_jpeg:n \__graphics_backend_include_jpg:n
\cs_new_eq:NN \__graphics_backend_include_pdf:n \__graphics_backend_include_jpg:n
\cs_new_eq:NN \__graphics_backend_include_png:n \__graphics_backend_include_jpg:n
```

(*End definition for* `\__graphics_backend_include_jpg:n` *and others.*)

`\__graphics_backend_getbb_eps:n`
`\__graphics_backend_getbb_ps:n`
`\__graphics_backend_getbb_eps:nm`
`\__graphics_backend_include_eps:n`
`\__graphics_backend_include_ps:n`
`\l__graphics_backend_dir_str`
`\l__graphics_backend_name_str`
`\l__graphics_backend_ext_str`

EPS graphics may be included in LuaTEX/pdfTeX by conversion to PDF: this requires restricted shell escape. Modelled on the epstopdf LATEX 2$_\varepsilon$ package, but simplified, conversion takes place here if we have shell access.

```
 \sys_if_shell:T
  {
    \str_new:N \l__graphics_backend_dir_str
    \str_new:N \l__graphics_backend_name_str
    \str_new:N \l__graphics_backend_ext_str
    \cs_new_protected:Npn \__graphics_backend_getbb_eps:n #1
      {
        \file_parse_full_name:nNNN {#1}
          \l__graphics_backend_dir_str
          \l__graphics_backend_name_str
          \l__graphics_backend_ext_str
        \exp_args:Nx \__graphics_backend_getbb_eps:nn
          {
            \exp_args:Ne \__kernel_file_name_quote:n
              {
                \l__graphics_backend_name_str
                - \str_tail:N \l__graphics_backend_ext_str
                -converted-to.pdf
              }
          }
          {#1}
     }
    \cs_new_eq:NN \__graphics_backend_getbb_ps:n \__graphics_backend_getbb_eps:n
    \cs_new_protected:Npn \__graphics_backend_getbb_eps:nn #1#2
      {
        \file_compare_timestamp:nNnT {#2} > {#1}
          {
            \sys_shell_now:n
              { repstopdf ~ #2 ~ #1 }
          }
        \tl_set:Nn \l__graphics_final_name_str {#1}
        \__graphics_backend_getbb_pdf:n {#1}
      }
    \cs_new_protected:Npn \__graphics_backend_include_eps:n #1
      {
        \file_parse_full_name:nNNN {#1}
          \l__graphics_backend_dir_str \l__graphics_backend_name_str \l__graphics_backend_ex
        \exp_args:Nx \__graphics_backend_include_pdf:n
          {
            \exp_args:Ne \__kernel_file_name_quote:n
              {
                \l__graphics_backend_name_str
                - \str_tail:N \l__graphics_backend_ext_str
                -converted-to.pdf
              }
          }
      }
    \cs_new_eq:NN \__graphics_backend_include_ps:n \__graphics_backend_include_eps:n
  }
```

(*End definition for* `\__graphics_backend_getbb_eps:n` *and others.*)

`\__graphics_backend_get_pagecount:n`

Simply load and store.

```
\cs_new_protected:Npn \__graphics_backend_get_pagecount:n #1
  {
    \tex_pdfximage:D {#1}
    \int_const:cn { c__graphics_ #1 _pages_int }
      { \int_use:N \tex_pdflastximagepages:D }
  }
```

(*End definition for* `\__graphics_backend_get_pagecount:n`*.*)

```
⟨/luatex | pdftex⟩
```

## 5.3 `dvipdfmx` backend

```
⟨*dvipdfmx | xetex⟩
```

**`\l_graphics_search_ext_seq`**

```
\__graphics_backend_loaded:n
  {
    \seq_set_from_clist:Nn \l_graphics_search_ext_seq
      { .pdf , .eps , .ps , .png , .jpg , .jpeg , .bmp }
   }
```

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

`\__graphics_backend_getbb_eps:n`
`\__graphics_backend_getbb_ps:n`
`\__graphics_backend_getbb_jpg:n`
`\__graphics_backend_getbb_jpeg:n`
`\__graphics_backend_getbb_pdf:n`
`\__graphics_backend_getbb_png:n`
`\__graphics_backend_getbb_bmp:n`

Simply use the generic functions: only for `dvipdfmx` in the extraction cases.

```
\__graphics_backend_loaded:n
  {
    \cs_new_eq:NN \__graphics_backend_getbb_eps:n \__graphics_read_bb:n
    \cs_new_eq:NN \__graphics_backend_getbb_ps:n \__graphics_read_bb:n
  }
⟨*dvipdfmx⟩
\cs_new_protected:Npn \__graphics_backend_getbb_jpg:n #1
  {
    \int_zero:N \l__graphics_page_int
    \tl_clear:N \l__graphics_pagebox_tl
    \__graphics_extract_bb:n {#1}
  }
\cs_new_eq:NN \__graphics_backend_getbb_jpeg:n \__graphics_backend_getbb_jpg:n
\cs_new_eq:NN \__graphics_backend_getbb_png:n \__graphics_backend_getbb_jpg:n
\cs_new_eq:NN \__graphics_backend_getbb_bmp:n \__graphics_backend_getbb_jpg:n
\cs_new_protected:Npn \__graphics_backend_getbb_pdf:n #1
  {
    \tl_clear:N \l__graphics_decodearray_str
    \bool_set_false:N \l__graphics_interpolate_bool
    \__graphics_extract_bb:n {#1}
  }
⟨/dvipdfmx⟩
```

(*End definition for* `\__graphics_backend_getbb_eps:n` *and others.*)

`\g__graphics_track_int`

Used to track the object number associated with each graphic.

```
\int_new:N \g__graphics_track_int
```

(*End definition for* `\g__graphics_track_int`.)

`\__graphics_backend_include_eps:n`
`\__graphics_backend_include_ps:n`
`\__graphics_backend_include_jpg:n`
`\__graphics_backend_include_jpseg:n`
`\__graphics_backend_include_pdf:n`
`\__graphics_backend_include_png:n`
`\__graphics_backend_include_bmp:n`
`\__graphics_backend_include_auxi:nn`
`\__graphics_backend_include_auxii:nnn`
`\__graphics_backend_include_auxii:xnn`
`\__graphics_backend_include_auxiii:nnn`

The special syntax depends on the file type. There is a difference in how PDF graphics are best handled between `dvipdfmx` and XƎTEX: for the latter it is better to use the primitive route. The relevant code for that is included later in this file.

```
\cs_new_protected:Npn \__graphics_backend_include_eps:n #1
  {
    \__kernel_backend_literal:x
      {
        PSfile = #1 \c_space_tl
        llx = \dim_to_decimal_in_bp:n \l__graphics_llx_dim \c_space_tl
        lly = \dim_to_decimal_in_bp:n \l__graphics_lly_dim \c_space_tl
        urx = \dim_to_decimal_in_bp:n \l__graphics_urx_dim \c_space_tl
        ury = \dim_to_decimal_in_bp:n \l__graphics_ury_dim
      }
  }
\cs_new_eq:NN \__graphics_backend_include_ps:n \__graphics_backend_include_eps:n
\cs_new_protected:Npn \__graphics_backend_include_jpg:n #1
  { \__graphics_backend_include_auxi:nn {#1} { image } }
\cs_new_eq:NN \__graphics_backend_include_jpeg:n \__graphics_backend_include_jpg:n
\cs_new_eq:NN \__graphics_backend_include_png:n \__graphics_backend_include_jpg:n
\cs_new_eq:NN \__graphics_backend_include_bmp:n \__graphics_backend_include_jpg:n
⟨*dvipdfmx⟩
\cs_new_protected:Npn \__graphics_backend_include_pdf:n #1
  { \__graphics_backend_include_auxi:nn {#1} { epdf } }
⟨/dvipdfmx⟩
```

Graphic inclusion is set up to use the fact that each image is stored in the PDF as an XObject. This means that we can include repeated images only once and refer to them. To allow that, track the nature of each image: much the same as for the direct PDF mode case.

```
\cs_new_protected:Npn \__graphics_backend_include_auxi:nn #1#2
  {
    \__graphics_backend_include_auxii:xnn
      {
        \tl_if_empty:NF \l__graphics_pagebox_tl
          { : \l__graphics_pagebox_tl }
        \int_compare:nNnT \l__graphics_page_int > 1
          { :P \int_use:N \l__graphics_page_int }
        \tl_if_empty:NF \l__graphics_decodearray_str
          { :D \l__graphics_decodearray_str }
        \bool_if:NT \l__graphics_interpolate_bool
           { :I }
      }
      {#1} {#2}
  }
\cs_new_protected:Npn \__graphics_backend_include_auxii:nnn #1#2#3
  {
    \int_if_exist:cTF { c__graphics_ #2#1 _int }
      {
        \__kernel_backend_literal:x
          { pdf:usexobj~@graphic \int_use:c { c__graphics_ #2#1 _int } }
      }
      { \__graphics_backend_include_auxiii:nnn {#2} {#1} {#3} }
  }
\cs_generate_variant:Nn \__graphics_backend_include_auxii:nnn { x }
```

Inclusion using the specials is relatively straight-forward, but there is one wrinkle. To get the `pagebox` correct for PDF graphics in all cases, it is necessary to provide both that information and the `bbox` argument: odd things happen otherwise!

```
\cs_new_protected:Npn \__graphics_backend_include_auxiii:nnn #1#2#3
  {
    \int_gincr:N \g__graphics_track_int
    \int_const:cn { c__graphics_ #1#2 _int } { \g__graphics_track_int }
    \__kernel_backend_literal:x
      {
        pdf:#3~
        @graphic \int_use:c { c__graphics_ #1#2 _int } ~
        \int_compare:nNnT \l__graphics_page_int > 1
          { page ~ \int_use:N \l__graphics_page_int \c_space_tl }
        \tl_if_empty:NF \l__graphics_pagebox_tl
          {
            pagebox ~ \l__graphics_pagebox_tl \c_space_tl
            bbox ~
              \dim_to_decimal_in_bp:n \l__graphics_llx_dim \c_space_tl
              \dim_to_decimal_in_bp:n \l__graphics_lly_dim \c_space_tl
              \dim_to_decimal_in_bp:n \l__graphics_urx_dim \c_space_tl
              \dim_to_decimal_in_bp:n \l__graphics_ury_dim \c_space_tl
          }
        (#1)
        \bool_lazy_or:nnT
          { \l__graphics_interpolate_bool }
          { ! \tl_if_empty_p:N \l__graphics_decodearray_str }
          {
            <<
              \tl_if_empty:NF \l__graphics_decodearray_str
                { /Decode~[ \l__graphics_decodearray_str ] }
              \bool_if:NT \l__graphics_interpolate_bool
                { /Interpolate~true> }
            >>
          }
      }
  }
```

(*End definition for* `\__graphics_backend_include_eps:n` *and others.*)

`\__graphics_backend_get_pagecount:n`

```
⟨*dvipdfmx⟩
\__graphics_backend_loaded:n
  { \cs_new_eq:NN \__graphics_backend_get_pagecount:n \__graphics_get_pagecount:n }
```

```
⟨/dvipdfmx⟩
```

(*End definition for* `\__graphics_backend_get_pagecount:n`.)

```
⟨/dvipdfmx | xetex⟩
```

## 5.4 XeTeX backend

```
⟨*xetex⟩
```

__graphics_backend_getbb_jpg:n
__graphics_backend_getbb_jpeg:n
__graphics_backend_getbb_pdf:n
__graphics_backend_getbb_png:n
__graphics_backend_getbb_bmp:n
__graphics_backend_getbb_auxi:nN
__graphics_backend_getbb_auxii:nnN
__graphics_backend_getbb_auxii:VnN
__graphics_backend_getbb_auxiii:nNnn
__graphics_backend_getbb_auxiv:nnNnn
__graphics_backend_getbb_auxiv:VnNnn
__graphics_backend_getbb_auxv:nNnn
__graphics_backend_getbb_auxv:nNnn
__graphics_backend_getbb_pagebox:w

For XeTeX, there are two primitives that allow us to obtain the bounding box without needing `extractbb`. The only complexity is passing the various minor variations to a common core process. The XeTeX primitive omits the text `box` from the page box specification, so there is also some “trimming” to do here.

```
\cs_new_protected:Npn \__graphics_backend_getbb_jpg:n #1
  {
    \int_zero:N \l__graphics_page_int
    \tl_clear:N \l__graphics_pagebox_tl
    \__graphics_backend_getbb_auxi:nN {#1} \tex_XeTeXpicfile:D
  }
\cs_new_eq:NN \__graphics_backend_getbb_jpeg:n \__graphics_backend_getbb_jpg:n
\cs_new_eq:NN \__graphics_backend_getbb_png:n \__graphics_backend_getbb_jpg:n
\cs_new_eq:NN \__graphics_backend_getbb_bmp:n \__graphics_backend_getbb_jpg:n
\cs_new_protected:Npn \__graphics_backend_getbb_pdf:n #1
  {
    \tl_clear:N \l__graphics_decodearray_str
    \bool_set_false:N \l__graphics_interpolate_bool
    \__graphics_backend_getbb_auxi:nN {#1} \tex_XeTeXpdffile:D
  }
\cs_new_protected:Npn \__graphics_backend_getbb_auxi:nN #1#2
  {
    \int_compare:nNnTF \l__graphics_page_int > 1
      { \__graphics_backend_getbb_auxii:VnN \l__graphics_page_int {#1} #2  }
      { \__graphics_backend_getbb_auxiii:nNnn {#1} #2 { :P 1 } { page 1 } }
  }
\cs_new_protected:Npn \__graphics_backend_getbb_auxii:nnN #1#2#3
  { \__graphics_backend_getbb_auxiii:nNnn {#2} #3 { :P #1 } { page #1 } }
\cs_generate_variant:Nn \__graphics_backend_getbb_auxii:nnN { V }
\cs_new_protected:Npn \__graphics_backend_getbb_auxiii:nNnn #1#2#3#4
  {
    \tl_if_empty:NTF \l__graphics_pagebox_tl
      { \__graphics_backend_getbb_auxiv:VnNnn \l__graphics_pagebox_tl }
      { \__graphics_backend_getbb_auxv:nNnn }
      {#1} #2 {#3} {#4}
  }
\cs_new_protected:Npn \__graphics_backend_getbb_auxiv:nnNnn #1#2#3#4#5
  {
    \use:x
      {
        \__graphics_backend_getbb_auxv:nNnn {#2} #3 { : #1 #4 }
          {
            #5
            \tl_if_blank:nF {#1}
              { \c_space_tl \__graphics_backend_getbb_pagebox:w #1 }
          }
      }
  }
\cs_generate_variant:Nn \__graphics_backend_getbb_auxiv:nnNnn { V }
\cs_new_protected:Npn \__graphics_backend_getbb_auxv:nNnn #1#2#3#4
  {
    \__graphics_bb_restore:nF {#1#3}
      { \__graphics_backend_getbb_auxvi:nNnn {#1} #2 {#3} {#4} }
  }
\cs_new_protected:Npn \__graphics_backend_getbb_auxvi:nNnn #1#2#3#4
  {
    \hbox_set:Nn \l__graphics_internal_box { #2 #1 ~ #4 }
    \dim_set:Nn \l__graphics_urx_dim { \box_wd:N \l__graphics_internal_box }
    \dim_set:Nn \l__graphics_ury_dim { \box_ht:N \l__graphics_internal_box }
    \__graphics_bb_save:n {#1#3}
  }
\cs_new:Npn \__graphics_backend_getbb_pagebox:w #1 box {#1}
```

(*End definition for* `\__graphics_backend_getbb_jpg:n` *and others.*)

`\__graphics_backend_include_pdf:n`

For PDF graphics, properly supporting the `pagebox` concept in XƎTEX is best done using the `\tex_XeTeXpdffile:D` primitive. The syntax here is the same as for the graphic measurement part, although we know at this stage that there must be some valid setting for `\l__graphics_pagebox_tl`.

```
\cs_new_protected:Npn \__graphics_backend_include_pdf:n #1
  {
    \tex_XeTeXpdffile:D #1 ~
      \int_compare:nNnT \l__graphics_page_int > 0
        { page ~ \int_use:N \l__graphics_page_int \c_space_tl }
        \exp_after:wN \__graphics_backend_getbb_pagebox:w \l__graphics_pagebox_tl
  }
```

(*End definition for* `\__graphics_backend_include_pdf:n`.)

`\__graphics_backend_get_pagecount:n`

Very little to do here other than cover the case of a non-PDF file.

```
\cs_new_protected:Npn \__graphics_backend_get_pagecount:n #1
  {
    \int_const:cn { c__graphics_ #1 _pages_int }
      {
        \int_max:nn
          { \int_use:N \tex_XeTeXpdfpagecount:D #1 ~ }
          { 1 }
      }
  }
```

(*End definition for* `\__graphics_backend_get_pagecount:n`.)

⟨/xetex⟩

### 5.5 dvisvgm backend

⟨*dvisvgm⟩

`\l_graphics_search_ext_seq`

```
\__graphics_backend_loaded:n
  {
    \seq_set_from_clist:Nn
      \l_graphics_search_ext_seq
      { .svg , .pdf , .eps , .ps , .png , .jpg , .jpeg }
  }
```

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

__graphics_backend_getbb_svg:n
__graphics_backend_getbb_svg_auxi:nNn
__graphics_backend_getbb_svg_auxii:w
__graphics_backend_getbb_svg_auxiii:Nw
__graphics_backend_getbb_svg_auxiv:Nw
__graphics_backend_getbb_svg_auxv:Nw
__graphics_backend_getbb_svg_auxvi:Nn
__graphics_backend_getbb_svg_auxvii:w

This is relatively similar to reading bounding boxes for `.eps` files. Life is though made more tricky as we cannot pick a single line for the data. So we have to loop until we collect up both height and width. To do that, we can use a marker value. We also have to allow for the default units of the lengths: they are big points and may be omitted.

```
\cs_new_protected:Npn \__graphics_backend_getbb_svg:n #1
  {
    \__graphics_bb_restore:nF {#1}
      {
        \ior_open:Nn \l__graphics_internal_ior {#1}
        \ior_if_eof:NTF \l__graphics_internal_ior
          { \msg_error:nnn { graphics } { graphic-not-found } {#1} }
          {
            \dim_zero:N \l__graphics_llx_dim
            \dim_zero:N \l__graphics_lly_dim
            \dim_set:Nn \l__graphics_urx_dim { -\c_max_dim }
            \dim_set:Nn \l__graphics_ury_dim { -\c_max_dim }
            \ior_str_map_inline:Nn \l__graphics_internal_ior
              {
                \dim_compare:nNnT \l__graphics_urx_dim = { -\c_max_dim }
                  {
                    \__graphics_backend_getbb_svg_auxi:nNn
                      { width } \l__graphics_urx_dim {##1}
                  }
                \dim_compare:nNnT \l__graphics_ury_dim = { -\c_max_dim }
                  {
                    \__graphics_backend_getbb_svg_auxi:nNn
                      { height } \l__graphics_ury_dim {##1}
                  }
                \bool_lazy_and:nnF
                  { \dim_compare_p:nNn \l__graphics_urx_dim = { -\c_max_dim } }
                  { \dim_compare_p:nNn \l__graphics_ury_dim = { -\c_max_dim } }
                  { \ior_map_break: }
              }
            \__graphics_bb_save:n {#1}
          }
        \ior_close:N \l__graphics_internal_ior
      }
  }
\cs_new_protected:Npn \__graphics_backend_getbb_svg_auxi:nNn #1#2#3
  {
    \use:x
      {
        \cs_set_protected:Npn \__graphics_backend_getbb_svg_auxii:w
          ####1 \tl_to_str:n {#1} = ####2 \tl_to_str:n {#1} = ####3
          \s__graphics_stop
      }
      {
        \tl_if_blank:nF {##2}
          {
            \peek_remove_spaces:n
              {
                \peek_meaning:NTF ' % '
                  { \__graphics_backend_getbb_svg_auxiii:Nw #2 }
                  {
                    \peek_meaning:NTF " % "
                      { \__graphics_backend_getbb_svg_auxiv:Nw #2 }
                      { \__graphics_backend_getbb_svg_auxv:Nw #2 }
                  }
               }
                ##2 \s__graphics_stop
          }
      }
    \use:x
      {
        \__graphics_backend_getbb_svg_auxii:w #3
          \tl_to_str:n {#1} = \tl_to_str:n {#1} =
          \s__graphics_stop
       }
  }
\cs_new_protected:Npn \__graphics_backend_getbb_svg_auxii:w { }
\cs_new_protected:Npn \__graphics_backend_getbb_svg_auxiii:Nw #1 ' #2 ' #3 \s__graphics_stop
  { \__graphics_backend_getbb_svg_auxvi:Nn #1 {#2} }
\cs_new_protected:Npn \__graphics_backend_getbb_svg_auxiv:Nw #1 " #2 " #3 \s__graphics_stop
  { \__graphics_backend_getbb_svg_auxvi:Nn #1 {#2} }
\cs_new_protected:Npn \__graphics_backend_getbb_svg_auxv:Nw #1  #2 ~ #3 \s__graphics_stop
  { \__graphics_backend_getbb_svg_auxvi:Nn #1 {#2} }
\cs_new_protected:Npn \__graphics_backend_getbb_svg_auxvi:Nn #1#2
  {
    \tex_afterassignment:D \__graphics_backend_getbb_svg_auxvii:w
      \l__graphics_internal_dim #2 bp \scan_stop:
    \dim_set_eq:NN #1 \l__graphics_internal_dim
  }
\cs_new_protected:Npn \__graphics_backend_getbb_svg_auxvii:w #1 \scan_stop: { }
```

(*End definition for* `\__graphics_backend_getbb_svg:n` *and others.*)

`\__graphics_backend_getbb_eps:n`
`\__graphics_backend_getbb_ps:n`

Simply use the generic function.

```
\__graphics_backend_loaded:n
  {
    \cs_new_eq:NN \__graphics_backend_getbb_eps:n \__graphics_read_bb:n
    \cs_new_eq:NN \__graphics_backend_getbb_ps:n \__graphics_read_bb:n
  }
```

(*End definition for* `\__graphics_backend_getbb_eps:n` *and* `\__graphics_backend_getbb_ps:n`.)

`\__graphics_backend_getbb_png:n`
`\__graphics_backend_getbb_jpg:n`
`\__graphics_backend_getbb_jpeg:n`

These can be included by extracting the bounding box data.

```
\cs_new_protected:Npn \__graphics_backend_getbb_jpg:n #1
  {
    \int_zero:N \l__graphics_page_int
    \tl_clear:N \l__graphics_pagebox_tl
    \__graphics_extract_bb:n {#1}
  }
```

```
\cs_new_eq:NN \__graphics_backend_getbb_jpeg:n \__graphics_backend_getbb_jpg:n
\cs_new_eq:NN \__graphics_backend_getbb_png:n \__graphics_backend_getbb_jpg:n
```

(*End definition for* `\__graphics_backend_getbb_png:n`*,* `\__graphics_backend_getbb_jpg:n`*, and* `\__graphics_backend_getbb_jpeg:n`*.*)

`\__graphics_backend_getbb_pdf:n`

Same as for `dvipdfmx`: use the generic function

```
\cs_new_protected:Npn \__graphics_backend_getbb_pdf:n #1
  {
    \tl_clear:N \l__graphics_decodearray_str
    \bool_set_false:N \l__graphics_interpolate_bool
    \__graphics_extract_bb:n {#1}
  }
```

(*End definition for* `\__graphics_backend_getbb_pdf:n`*.*)

`\__graphics_backend_include_eps:n`
`\__graphics_backend_include_ps:n`
`\__graphics_backend_include_pdf:n`
`\__graphics_backend_include:nn`

The special syntax is relatively clear here: remember we need PostScript sizes here. (This is the same as the `dvips` code.)

```
\cs_new_protected:Npn \__graphics_backend_include_eps:n #1
  { \__graphics_backend_include:nn { PSfile } {#1} }
\cs_new_eq:NN \__graphics_backend_include_ps:n \__graphics_backend_include_eps:n
\cs_new_protected:Npn \__graphics_backend_include_pdf:n #1
  { \__graphics_backend_include:nn { pdffile } {#1} }
\cs_new_protected:Npn \__graphics_backend_include:nn #1#2
  {
    \__kernel_backend_literal:x
      {
        #1 = #2 \c_space_tl
        llx = \dim_to_decimal_in_bp:n \l__graphics_llx_dim \c_space_tl
        lly = \dim_to_decimal_in_bp:n \l__graphics_lly_dim \c_space_tl
        urx = \dim_to_decimal_in_bp:n \l__graphics_urx_dim \c_space_tl
        ury = \dim_to_decimal_in_bp:n \l__graphics_ury_dim
      }
  }
```

(*End definition for* `\__graphics_backend_include_eps:n` *and others.*)

`\__graphics_backend_include_svg:n`
`\__graphics_backend_include_png:n`
`\__graphics_backend_include_jpg:n`
`\__graphics_backend_include_jpeg:n`
`\__graphics_backend_include_dequote:w`

The backend here has built-in support for basic graphic inclusion (see `dvisvgm.def` for a more complex approach, needed if clipping, *etc.*, is covered at the graphic backend level). We have to deal with the fact that the image reference point is at the *top*, so there is a need for a vertical shift to put it in the right place. The other issue is that `#1` must be quote-corrected. The `dvisvgm:img` operation quotes the file name, but if it is already quoted (contains spaces) then we have an issue: we simply strip off any quotes as a result.

```
\cs_new_protected:Npn \__graphics_backend_include_svg:n #1
  {
    \box_move_up:nn { \l__graphics_ury_dim }
      {
        \hbox:n
          {
            \__kernel_backend_literal:x
              {
                dvisvgm:img~
                \dim_to_decimal:n { \l__graphics_urx_dim } ~
                \dim_to_decimal:n { \l__graphics_ury_dim } ~
                \__graphics_backend_include_dequote:w #1 " #1 " \s__graphics_stop
              }
          }
      }
  }
\cs_new_eq:NN \__graphics_backend_include_png:n \__graphics_backend_include_svg:n
\cs_new_eq:NN \__graphics_backend_include_jpeg:n \__graphics_backend_include_svg:n
\cs_new_eq:NN \__graphics_backend_include_jpg:n \__graphics_backend_include_svg:n
\cs_new:Npn \__graphics_backend_include_dequote:w #1 " #2 " #3 \s__graphics_stop
  {#2}
```

(*End definition for* `\__graphics_backend_include_svg:n` *and others.*)

`\__graphics_backend_get_pagecount:n`

```
\__graphics_backend_loaded:n
  { \cs_new_eq:NN \__graphics_backend_get_pagecount:n \__graphics_get_pagecount:n }
```

(*End definition for* `\__graphics_backend_get_pagecount:n`.)

```
⟨/dvisvgm⟩
⟨/package⟩
```

## 6 l3backend-pdf Implementation

```
⟨*package⟩
⟨@@=pdf⟩
```

Setting up PDF resources is a complex area with only limited documentation in the engine manuals. The following code builds heavily on existing ideas from hyperref work by Sebastian Rahtz and Heiko Oberdiek, and significant contributions by Alexander Grahn, in addition to the specific code referenced a various points.

### 6.1 Shared code

A very small number of items that belong at the backend level but which are common to all backends.

`\l__pdf_internal_box`

```
\box_new:N \l__pdf_internal_box
```

(*End definition for* `\l__pdf_internal_box`.)

### 6.2 dvips backend

```
⟨*dvips⟩
```

`\__pdf_backend_pdfmark:n`
`\__pdf_backend_pdfmark:x`

Used often enough it should be a separate function.

```
\cs_new_protected:Npn \__pdf_backend_pdfmark:n #1
  { \__kernel_backend_postscript:n { mark #1 ~ pdfmark } }
\cs_generate_variant:Nn \__pdf_backend_pdfmark:n { x }
```

(*End definition for* `\__pdf_backend_pdfmark:n`.)

### 6.2.1 Catalogue entries

__pdf_backend_catalog_gput:nn
__pdf_backend_info_gput:nn

```
\cs_new_protected:Npn \__pdf_backend_catalog_gput:nn #1#2
  { \__pdf_backend_pdfmark:n { { Catalog } << /#1 ~ #2 >> /PUT } }
\cs_new_protected:Npn \__pdf_backend_info_gput:nn #1#2
  { \__pdf_backend_pdfmark:n { /#1 ~ #2 /DOCINFO } }
```

(*End definition for* `\__pdf_backend_catalog_gput:nn` *and* `\__pdf_backend_info_gput:nn`.)

### 6.2.2 Objects

\g__pdf_backend_object_int

For tracking objects.

```
\int_new:N \g__pdf_backend_object_int
```

(*End definition for* `\g__pdf_backend_object_int`.)

__pdf_backend_object_new:n
__pdf_backend_object_ref:n

```
\cs_new_protected:Npn \__pdf_backend_object_new:n #1
  {
    \int_gincr:N \g__pdf_backend_object_int
    \int_const:cn
      { c__pdf_object_ \tl_to_str:n {#1} _int }
      { \g__pdf_backend_object_int }
  }
\cs_new:Npn \__pdf_backend_object_ref:n #1
  { { pdf.obj \int_use:c { c__pdf_object_ \tl_to_str:n {#1} _int } } }
```

(*End definition for* `\__pdf_backend_object_new:n` *and* `\__pdf_backend_object_ref:n`.)

__pdf_backend_object_write:nnn
__pdf_backend_object_write:nnx
__pdf_backend_object_write_aux:nnn
__pdf_backend_object_write_array:nn
__pdf_backend_object_write_dict:nn
__pdf_backend_object_write_fstream:nn
__pdf_backend_object_write_stream:nn
__pdf_backend_object_write_stream:nnn

This is where we choose the actual type: some work to get things right. To allow code sharing with the anonymous version, we use an auxiliary.

```
\cs_new_protected:Npn \__pdf_backend_object_write:nnn #1#2#3
  {
    \__pdf_backend_object_write_aux:nnn
      { \__pdf_backend_object_ref:n {#1} }
      {#2} {#3}
  }
\cs_generate_variant:Nn \__pdf_backend_object_write:nnn { nnx }
\cs_new_protected:Npn \__pdf_backend_object_write_aux:nnn #1#2#3
  {
    \__pdf_backend_pdfmark:x
      {
        /_objdef ~ #1
        /type
        \str_case:nn {#2}
          {
            { array }   { /array }
            { dict }    { /dict }
            { fstream } { /stream }
            { stream }  { /stream }
          }
        /OBJ
      }
    \use:c { __pdf_backend_object_write_ #2 :nn } {#1} {#3}
  }
\cs_new_protected:Npn \__pdf_backend_object_write_array:nn #1#2
  {
    \__pdf_backend_pdfmark:x
      { #1 ~0~ [ ~ \exp_not:n {#2} ~ ] ~ /PUTINTERVAL }
  }
\cs_new_protected:Npn \__pdf_backend_object_write_dict:nn #1#2
  {
    \__pdf_backend_pdfmark:x
      { #1 << \exp_not:n {#2} >> /PUT }
  }
\cs_new_protected:Npn \__pdf_backend_object_write_fstream:nn #1#2
  {
    \exp_args:Nx
      \__pdf_backend_object_write_fstream:nnn {#1} #2
  }
\cs_new_protected:Npn \__pdf_backend_object_write_fstream:nnn #1#2#3
  {
    \__kernel_backend_postscript:n
      {
        SDict ~ begin ~
        mark ~ #1 ~ << #2 >> /PUT ~ pdfmark ~
        mark ~ #1 ~ ( #3 )~ ( r )~ file ~ /PUT ~ pdfmark ~
        end
      }
  }
\cs_new_protected:Npn \__pdf_backend_object_write_stream:nn #1#2
  {
    \exp_args:Nx
      \__pdf_backend_object_write_stream:nnn {#1} #2
  }
\cs_new_protected:Npn \__pdf_backend_object_write_stream:nnn #1#2#3
  {
    \__kernel_backend_postscript:n
      {
        mark ~ #1 ~ ( #3 ) /PUT ~ pdfmark ~
        mark ~ #1 ~ << #2 >> /PUT ~ pdfmark
      }
  }
```

(*End definition for* `\__pdf_backend_object_write:nnn` *and others.*)

`\__pdf_backend_object_now:nn`
`\__pdf_backend_object_now:nx`

No anonymous objects, so things are done manually.

```
\cs_new_protected:Npn \__pdf_backend_object_now:nn #1#2
  {
    \int_gincr:N \g__pdf_backend_object_int
    \__pdf_backend_object_write_aux:nnn
      { { pdf.obj \int_use:N \g__pdf_backend_object_int } }
      {#1} {#2}
  }
\cs_generate_variant:Nn \__pdf_backend_object_now:nn { nx }
```

(*End definition for* `\__pdf_backend_object_now:nn`.)

`\__pdf_backend_object_last:` Much like the annotation version.

```
\cs_new:Npn \__pdf_backend_object_last:
  { { pdf.obj \int_use:N \g__pdf_backend_object_int } }
```

(*End definition for* `\__pdf_backend_object_last:`.)

`\__pdf_backend_pageobject_ref:n` Page references are easy in `dvips`.

```
\cs_new:Npn \__pdf_backend_pageobject_ref:n #1
  { { Page #1 } }
```

(*End definition for* `\__pdf_backend_pageobject_ref:n`.)

### 6.2.3 Annotations

In `dvips`, annotations have to be constructed manually. As such, we need the object code above for some definitions.

`\l__pdf_backend_content_box` The content of an annotation.

```
\box_new:N \l__pdf_backend_content_box
```

(*End definition for* `\l__pdf_backend_content_box`.)

`\l__pdf_backend_model_box` For creating model sizing for links.

```
\box_new:N \l__pdf_backend_model_box
```

(*End definition for* `\l__pdf_backend_model_box`.)

`\g__pdf_backend_annotation_int` Needed as objects which are not annotations could be created.

```
\int_new:N \g__pdf_backend_annotation_int
```

(*End definition for* `\g__pdf_backend_annotation_int`.)

`\__pdf_backend_annotation:nnnn` Annotations are objects, but we track them separately. Notably, they are not in the object data lists. Here, to get the co-ordinates of the annotation, we need to have the data collected at the PostScript level. That requires a bit of box trickery (effectively a LaTeX 2$_\varepsilon$ `picture` of zero size). Once the data is collected, use it to set up the annotation border.

```
\cs_new_protected:Npn \__pdf_backend_annotation:nnnn #1#2#3#4
  {
    \exp_args:Nf \__pdf_backend_annotation_aux:nnnn
      { \dim_eval:n {#1} } {#2} {#3} {#4}
  }
\cs_new_protected:Npn \__pdf_backend_annotation_aux:nnnn #1#2#3#4
  {
    \box_move_down:nn {#3}
      { \hbox:n { \__kernel_backend_postscript:n { pdf.save.ll } } }
    \box_move_up:nn {#2}
      {
        \hbox:n
          {
            \__kernel_kern:n {#1}
            \__kernel_backend_postscript:n { pdf.save.ur }
            \__kernel_kern:n { -#1 }
          }
      }
    \int_gincr:N \g__pdf_backend_object_int
    \int_gset_eq:NN \g__pdf_backend_annotation_int \g__pdf_backend_object_int
    \__pdf_backend_pdfmark:x
      {
        /_objdef { pdf.obj \int_use:N \g__pdf_backend_object_int }
        pdf.rect
        #4 ~
        /ANN
      }
  }
```

(*End definition for* `\__pdf_backend_annotation:nnnn`.)

`\__pdf_backend_annotation_last:`

Provide the last annotation we created: could get tricky of course if other packages are loaded.

```
\cs_new:Npn \__pdf_backend_annotation_last:
  { { pdf.obj \int_use:N \g__pdf_backend_annotation_int } }
```

(*End definition for* `\__pdf_backend_annotation_last:`.)

`\g__pdf_backend_link_int`

To track annotations which are links.

```
\int_new:N \g__pdf_backend_link_int
```

(*End definition for* `\g__pdf_backend_link_int`.)

`\g__pdf_backend_link_dict_tl`

To pass information to the end-of-link function.

```
\tl_new:N \g__pdf_backend_link_dict_tl
```

(*End definition for* `\g__pdf_backend_link_dict_tl`.)

`\g__pdf_backend_link_sf_int`

Needed to save/restore space factor, which is needed to deal with the face we need a box.

```
\int_new:N \g__pdf_backend_link_sf_int
```

(*End definition for* `\g__pdf_backend_link_sf_int`.)

`\g__pdf_backend_link_math_bool`

Needed to save/restore math mode.

```
\bool_new:N \g__pdf_backend_link_math_bool
```

(*End definition for* `\g__pdf_backend_link_math_bool`.)

`\g__pdf_backend_link_bool`

Track link formation: we cannot nest at all.

```
\bool_new:N \g__pdf_backend_link_bool
```

(*End definition for* `\g__pdf_backend_link_bool`.)

`\l__pdf_breaklink_pdfmark_tl`

Swappable content for link breaking.

```
\tl_new:N \l__pdf_breaklink_pdfmark_tl
\tl_set:Nn \l__pdf_breaklink_pdfmark_tl { pdfmark }
```

(*End definition for* `\l__pdf_breaklink_pdfmark_tl`.)

`\__pdf_breaklink_postscript:n`

To allow dropping material unless link breaking is active.

```
\cs_new_protected:Npn \__pdf_breaklink_postscript:n #1 { }
```

(*End definition for* `\__pdf_breaklink_postscript:n`.)

`\__pdf_breaklink_usebox:N`

Swappable box unpacking or use.

```
\cs_new_eq:NN \__pdf_breaklink_usebox:N \box_use:N
```

(*End definition for* `\__pdf_breaklink_usebox:N`.)

`\__pdf_backend_link_begin_goto:nnw`
`\__pdf_backend_link_begin_user:nnw`
`\__pdf_backend_link:nw`
`\__pdf_backend_link_aux:nw`
`\__pdf_backend_link_end:`
`\__pdf_backend_link_end_aux:`
`\__pdf_backend_link_minima:`
`\__pdf_backend_link_outerbox:n`
`\__pdf_backend_link_sf_save:`
`\__pdf_backend_link_sf_restore:`
`pdf.linkdp.pad`
`pdf.linkht.pad`
`pdf.llx`
`pdf.lly`
`pdf.ury`
`pdf.link.dict`
`pdf.outerbox`
`pdf.baselineskip`

Links are crated like annotations but with dedicated code to allow for adjusting the size of the rectangle. In contrast to hyperref, we grab the link content as a box which can then unbox: this allows the same interface as for pdfTEX.

Notice that the link setup here uses `/Action` not `/A`. That is because Distiller *requires* this trigger word, rather than a "raw" PDF dictionary key (Ghostscript can handle either form).

Taking the idea of `evenboxes` from hypdvips, we implement a minimum box height and depth for link placement. This means that "underlining" with a hyperlink will generally give an even appearance. However, to ensure that the full content is always above the link border, we do not allow this to be negative (contrast hypdvips approach). The result should be similar to pdfTEX in the vast majority of foreseeable cases.

The object number for a link is saved separately from the rest of the dictionary as this allows us to insert it just once, at either an unbroken link or only in the first line of a broken one. That makes the code clearer but also avoids a low-level PostScript error with the code as taken from hypdvips.

Getting the outer dimensions of the text area may be better using a two-pass approach and `\tex_savepos:D`. That plus generic mode are still to re-examine.

```
\cs_new_protected:Npn \__pdf_backend_link_begin_goto:nnw #1#2
  {
    \__pdf_backend_link_begin:nw
      { #1 /Subtype /Link /Action << /S /GoTo /D ( #2 ) >> }
  }
\cs_new_protected:Npn \__pdf_backend_link_begin_user:nnw #1#2
  { \__pdf_backend_link_begin:nw {#1#2} }
\cs_new_protected:Npn \__pdf_backend_link_begin:nw #1
  {
    \bool_if:NF \g__pdf_backend_link_bool
      { \__pdf_backend_link_begin_aux:nw {#1} }
  }
```

The definition of `pdf.link.dict` here is needed as there is code in the PostScript headers for breaking links, and that can only work with this available.

```
\cs_new_protected:Npn \__pdf_backend_link_begin_aux:nw #1
  {
    \bool_gset_true:N \g__pdf_backend_link_bool
    \__kernel_backend_postscript:n
      { /pdf.link.dict ( #1 ) def }
    \tl_gset:Nn \g__pdf_backend_link_dict_tl {#1}
    \__pdf_backend_link_sf_save:
    \mode_if_math:TF
      { \bool_gset_true:N \g__pdf_backend_link_math_bool }
      { \bool_gset_false:N \g__pdf_backend_link_math_bool }
    \hbox_set:Nw \l__pdf_backend_content_box
      \__pdf_backend_link_sf_restore:
      \bool_if:NT \g__pdf_backend_link_math_bool
        { \c_math_toggle_token }
  }
\cs_new_protected:Npn \__pdf_backend_link_end:
  {
    \bool_if:NT \g__pdf_backend_link_bool
      { \__pdf_backend_link_end_aux: }
  }
\cs_new_protected:Npn \__pdf_backend_link_end_aux:
  {
      \bool_if:NT \g__pdf_backend_link_math_bool
        { \c_math_toggle_token }
      \__pdf_backend_link_sf_save:
    \hbox_set_end:
    \__pdf_backend_link_minima:
    \hbox_set:Nn \l__pdf_backend_model_box { Gg }
    \exp_args:Nx \__pdf_backend_link_outerbox:n
      {
         \int_if_odd:nTF { \value { page } }
           { \oddsidemargin }
           { \evensidemargin }
      }
    \box_move_down:nn { \box_dp:N \l__pdf_backend_content_box }
      { \hbox:n { \__kernel_backend_postscript:n { pdf.save.linkll } } }
    \__pdf_breaklink_postscript:n { pdf.bordertracking.begin }
    \__pdf_breaklink_usebox:N \l__pdf_backend_content_box
    \__pdf_breaklink_postscript:n { pdf.bordertracking.end }
    \box_move_up:nn { \box_ht:N \l__pdf_backend_content_box }
      {
        \hbox:n
          { \__kernel_backend_postscript:n { pdf.save.linkur } }
      }
    \int_gincr:N \g__pdf_backend_object_int
    \int_gset_eq:NN \g__pdf_backend_link_int \g__pdf_backend_object_int
    \__kernel_backend_postscript:x
      {
        mark
        /_objdef { pdf.obj \int_use:N \g__pdf_backend_link_int }
        \g__pdf_backend_link_dict_tl \c_space_tl
        pdf.rect
        /ANN ~ \l__pdf_breaklink_pdfmark_tl
      }
    \__pdf_backend_link_sf_restore:
    \bool_gset_false:N \g__pdf_backend_link_bool
  }
\cs_new_protected:Npn \__pdf_backend_link_minima:
  {
    \hbox_set:Nn \l__pdf_backend_model_box { Gg }
    \__kernel_backend_postscript:x
      {
        /pdf.linkdp.pad ~
          \dim_to_decimal:n
            {
              \dim_max:nn
                {
                    \box_dp:N \l__pdf_backend_model_box
                  - \box_dp:N \l__pdf_backend_content_box
                }
                { 0pt }
            } ~
              pdf.pt.dvi ~ def
        /pdf.linkht.pad ~
          \dim_to_decimal:n
            {
              \dim_max:nn
                {
                    \box_ht:N \l__pdf_backend_model_box
                  - \box_ht:N \l__pdf_backend_content_box
                }
                { 0pt }
            } ~
              pdf.pt.dvi ~ def
      }
  }
\cs_new_protected:Npn \__pdf_backend_link_outerbox:n #1
  {
    \__kernel_backend_postscript:x
      {
        /pdf.outerbox
          [
            \dim_to_decimal:n {#1} ~
            \dim_to_decimal:n { -\box_dp:N \l__pdf_backend_model_box } ~
            \dim_to_decimal:n { #1 + \textwidth } ~
            \dim_to_decimal:n { \box_ht:N \l__pdf_backend_model_box }
          ]
          [ exch { pdf.pt.dvi } forall ] def
        /pdf.baselineskip ~
          \dim_to_decimal:n { \tex_baselineskip:D } ~ dup ~ 0 ~ gt
            { pdf.pt.dvi ~ def }
            { pop ~ pop }
          ifelse
      }
  }
\cs_new_protected:Npn \__pdf_backend_link_sf_save:
  {
    \int_gset:Nn \g__pdf_backend_link_sf_int
      {
        \mode_if_horizontal:TF
          { \tex_spacefactor:D }
          { 0 }
      }
  }
\cs_new_protected:Npn \__pdf_backend_link_sf_restore:
  {
    \mode_if_horizontal:T
      {
        \int_compare:nNnT \g__pdf_backend_link_sf_int > { 0 }
          { \int_set_eq:NN \tex_spacefactor:D \g__pdf_backend_link_sf_int }
      }
  }
```

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

`\@makecol@hook`

Hooks to allow link breaking: something will be needed in format mode at some stage. At present this code is disabled as there is an open question about the name of the hook: to be resolved at the LaTeX 2ε end.

```
\use_none:n
  {
    \cs_if_exist:NT \@makecol@hook
      {
        \tl_put_right:Nn \@makecol@hook
          {
            \box_if_empty:NF \@cclv
              {
                \vbox_set:Nn \@cclv
                  {
                    \__kernel_backend_postscript:n
                      {
                        pdf.globaldict /pdf.brokenlink.rect ~ known
                          { pdf.bordertracking.continue }
                        if
                      }
                    \vbox_unpack_drop:N \@cclv
                    \__kernel_backend_postscript:n
                      { pdf.bordertracking.endpage }
                  }
              }
          }
        \tl_set:Nn \l__pdf_breaklink_pdfmark_tl { pdf.pdfmark }
        \cs_set_eq:NN \__pdf_breaklink_postscript:n \__kernel_backend_postscript:n
        \cs_set_eq:NN \__pdf_breaklink_usebox:N \hbox_unpack:N
      }
  }
```

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

`\__pdf_backend_link_last:`

The same as annotations, but with a custom integer.

```
\cs_new:Npn \__pdf_backend_link_last:
  { { pdf.obj \int_use:N \g__pdf_backend_link_int } }
```

(*End definition for* `\__pdf_backend_link_last:`.)

`\__pdf_backend_link_margin:n`

Convert to big points and pass to PostScript.

```
\cs_new_protected:Npn \__pdf_backend_link_margin:n #1
  {
    \__kernel_backend_postscript:x
      {
        /pdf.linkmargin { \dim_to_decimal:n {#1} ~ pdf.pt.dvi } def
      }
  }
```

(*End definition for* `\__pdf_backend_link_margin:n`.)

`\__pdf_backend_destination:nn`
`\__pdf_backend_destination:nnnn`
`\__pdf_backend_destination_aux:nnnn`

Here, we need to turn the zoom into a scale. We also need to know where the current anchor point actually is: worked out in PostScript. For the rectangle version, we have a bit more PostScript: we need two points. fitr without rule spec doesn't work, so it falls back to `/Fit` here.

```
\cs_new_protected:Npn \__pdf_backend_destination:nn #1#2
  {
    \__kernel_backend_postscript:n { pdf.dest.anchor }
    \__pdf_backend_pdfmark:x
      {
        /View
        [
          \str_case:nnF {#2}
            {
              { xyz }   { /XYZ ~ pdf.dest.point ~ null }
              { fit }   { /Fit }
              { fitb }  { /FitB }
              { fitbh } { /FitBH ~ pdf.dest.y }
              { fitbv } { /FitBV ~ pdf.dest.x }
              { fith }  { /FitH ~ pdf.dest.y }
              { fitv }  { /FitV ~ pdf.dest.x }
              { fitr }  { /Fit }
            }
            {
              /XYZ ~ pdf.dest.point ~ \fp_eval:n { (#2) / 100 }
            }
        ]
        /Dest ( \exp_not:n {#1} ) cvn
        /DEST
      }
  }
\cs_new_protected:Npn \__pdf_backend_destination:nnnn #1#2#3#4
  {
    \exp_args:Ne \__pdf_backend_destination_aux:nnnn
      { \dim_eval:n {#2} } {#1} {#3} {#4}
  }
\cs_new_protected:Npn \__pdf_backend_destination_aux:nnnn #1#2#3#4
  {
    \vbox_to_zero:n
      {
        \__kernel_kern:n {#4}
        \hbox:n { \__kernel_backend_postscript:n { pdf.save.ll } }
        \tex_vss:D
      }
    \__kernel_kern:n {#1}
    \vbox_to_zero:n
      {
        \__kernel_kern:n { -#3 }
        \hbox:n { \__kernel_backend_postscript:n { pdf.save.ur } }
        \tex_vss:D
      }
    \__kernel_kern:n { -#1 }
    \__pdf_backend_pdfmark:n
      {
        /View
        [
          /FitR ~
            pdf.llx ~ pdf.lly ~ pdf.dest2device ~
            pdf.urx ~ pdf.ury ~ pdf.dest2device
        ]
        /Dest ( #2 ) cvn
        /DEST
      }
  }
```

(*End definition for* `\__pdf_backend_destination:nn`, `\__pdf_backend_destination:nnnn`, *and* `\__pdf_backend_destination_aux:nnnn`.)

### 6.2.4 Structure

`\__pdf_backend_compresslevel:n`
`\__pdf_backend_compress_objects:n`

Doable for the usual `ps2pdf` method.

```
\cs_new_protected:Npn \__pdf_backend_compresslevel:n #1
  {
    \int_compare:nNnT {#1} = 0
      {
        \__kernel_backend_literal_postscript:n
          {
            /setdistillerparams ~ where
             { pop << /CompressPages ~ false >> setdistillerparams }
            if
          }
      }
  }
\cs_new_protected:Npn \__pdf_backend_compress_objects:n #1
  {
    \bool_if:nF {#1}
      {
        \__kernel_backend_literal_postscript:n
          {
            /setdistillerparams ~ where
             { pop << /CompressStreams ~ false >> setdistillerparams }
            if
          }
      }
  }
```

(*End definition for* `\__pdf_backend_compresslevel:n` *and* `\__pdf_backend_compress_objects:n`.)

`\__pdf_backend_version_major_gset:n`
`\__pdf_backend_version_minor_gset:n`

```
\cs_new_protected:Npn \__pdf_backend_version_major_gset:n #1
  {
    \cs_gset:Npx \__pdf_backend_version_major: { \int_eval:n {#1} }
  }
\cs_new_protected:Npn \__pdf_backend_version_minor_gset:n #1
  {
    \cs_gset:Npx \__pdf_backend_version_minor: { \int_eval:n {#1} }
  }
```

(*End definition for* `\__pdf_backend_version_major_gset:n` *and* `\__pdf_backend_version_minor_gset:n`.)

`\__pdf_backend_version_major:`
`\__pdf_backend_version_minor:`

Data not available!

```
\cs_new:Npn \__pdf_backend_version_major: { -1 }
\cs_new:Npn \__pdf_backend_version_minor: { -1 }
```

(*End definition for* `\__pdf_backend_version_major:` *and* `\__pdf_backend_version_minor:`.)

### 6.2.5 Marked content

`\__pdf_backend_bdc:nn`
`\__pdf_backend_emc:`

Simple wrappers.

```
\cs_new_protected:Npn \__pdf_backend_bdc:nn #1#2
  { \__pdf_backend_pdfmark:n { /#1 ~ #2 /BDC } }
\cs_new_protected:Npn \__pdf_backend_emc:
  { \__pdf_backend_pdfmark:n { /EMC } }
```

(*End definition for* `\__pdf_backend_bdc:nn` *and* `\__pdf_backend_emc:`.)

```
⟨/dvips⟩
```

## 6.3 LuaTEX and pdfTEX backend

```
⟨*luatex | pdftex⟩
```

### 6.3.1 Annotations

`\__pdf_backend_annotation:nnnn`

Simply pass the raw data through, just dealing with evaluation of dimensions.

```
\cs_new_protected:Npn \__pdf_backend_annotation:nnnn #1#2#3#4
  {
⟨*luatex⟩
    \tex_pdfextension:D annot ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfannot:D
⟨/pdftex⟩
      width  ~ \dim_eval:n {#1} ~
      height ~ \dim_eval:n {#2} ~
      depth  ~ \dim_eval:n {#3} ~
      {#4}
  }
```

(*End definition for* `\__pdf_backend_annotation:nnnn`.)

`\__pdf_backend_annotation_last:`

A tiny amount of extra data gets added here; we use `x`-type expansion to get the space in the right place and form. The "extra" space in the LuaTEX version is *required* as it is consumed in finding the end of the keyword.

```
\cs_new:Npx \__pdf_backend_annotation_last:
  {
    \exp_not:N \int_value:w
⟨*luatex⟩
      \exp_not:N \tex_pdffeedback:D lastannot ~
⟨/luatex⟩
⟨*pdftex⟩
      \exp_not:N \tex_pdflastannot:D
⟨/pdftex⟩
      \c_space_tl 0 ~ R
  }
```

(*End definition for* `\__pdf_backend_annotation_last:`.)

`\__pdf_backend_link_begin_goto:nnw`
`\__pdf_backend_link_begin_user:nnw`
`\__pdf_backend_link_begin:nnnw`
`\__pdf_backend_link_end:`

Links are all created using the same internals.

```
\cs_new_protected:Npn \__pdf_backend_link_begin_goto:nnw #1#2
  { \__pdf_backend_link_begin:nnnw {#1} { goto~name } {#2} }
\cs_new_protected:Npn \__pdf_backend_link_begin_user:nnw #1#2
```

```
  { \__pdf_backend_link_begin:nnnw {#1} { user } {#2} }
\cs_new_protected:Npn \__pdf_backend_link_begin:nnnw #1#2#3
  {
⟨*luatex⟩
    \tex_pdfextension:D startlink ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfstartlink:D
⟨/pdftex⟩
      attr {#1}
      #2 {#3}
  }
\cs_new_protected:Npn \__pdf_backend_link_end:
  {
⟨*luatex⟩
    \tex_pdfextension:D endlink \scan_stop:
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfendlink:D
⟨/pdftex⟩
  }
```

(*End definition for* `\__pdf_backend_link_begin_goto:nnw` *and others.*)

`\__pdf_backend_link_last:` Formatted for direct use.

```
\cs_new:Npx \__pdf_backend_link_last:
  {
    \exp_not:N \int_value:w
⟨*luatex⟩
      \exp_not:N \tex_pdffeedback:D lastlink ~
⟨/luatex⟩
⟨*pdftex⟩
      \exp_not:N \tex_pdflastlink:D
⟨/pdftex⟩
      \c_space_tl 0 ~ R
  }
```

(*End definition for* `\__pdf_backend_link_last:`.)

`\__pdf_backend_link_margin:n` A simple task: pass the data to the primitive.

```
\cs_new_protected:Npn \__pdf_backend_link_margin:n #1
  {
⟨*luatex⟩
    \tex_pdfvariable:D linkmargin
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdflinkmargin:D
⟨/pdftex⟩
      \dim_eval:n {#1} \scan_stop:
  }
```

(*End definition for* `\__pdf_backend_link_margin:n`.)

__pdf_backend_destination:nn
__pdf_backend_destination:nnnn

A simple task: pass the data to the primitive. The `\scan_stop:` deals with the danger of an unterminated keyword. The zoom given here is a percentage, but we need to pass it as *per mille*. The rectangle version is also easy as everything is build in.

```
\cs_new_protected:Npn \__pdf_backend_destination:nn #1#2
  {
⟨*luatex⟩
    \tex_pdfextension:D dest ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfdest:D
⟨/pdftex⟩
        name {#1}
        \str_case:nnF {#2}
          {
            { xyz }   { xyz }
            { fit }   { fit }
            { fitb }  { fitb }
            { fitbh } { fitbh }
            { fitbv } { fitbv }
            { fith }  { fith }
            { fitv }  { fitv }
            { fitr }  { fitr }
          }
          { xyz ~ zoom \fp_eval:n { #2 * 10 } }
        \scan_stop:
  }
\cs_new_protected:Npn \__pdf_backend_destination:nnnn #1#2#3#4
  {
⟨*luatex⟩
    \tex_pdfextension:D dest ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfdest:D
⟨/pdftex⟩
    name {#1}
    fitr ~
      width  \dim_eval:n {#2} ~
      height \dim_eval:n {#3} ~
      depth  \dim_eval:n {#4} \scan_stop:
  }
```

(*End definition for* `\__pdf_backend_destination:nn` *and* `\__pdf_backend_destination:nnnn`*.*)

### 6.3.2 Catalogue entries

__pdf_backend_catalog_gput:nn
__pdf_backend_info_gput:nn

```
\cs_new_protected:Npn \__pdf_backend_catalog_gput:nn #1#2
  {
⟨*luatex⟩
    \tex_pdfextension:D catalog
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfcatalog:D
⟨/pdftex⟩
```

```
      { / #1 ~ #2 }
  }
\cs_new_protected:Npn \__pdf_backend_info_gput:nn #1#2
  {
⟨*luatex⟩
    \tex_pdfextension:D info
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfinfo:D
⟨/pdftex⟩
      { / #1 ~ #2 }
  }
```

(*End definition for* `\__pdf_backend_catalog_gput:nn` *and* `\__pdf_backend_info_gput:nn`.)

### 6.3.3 Objects

`\g__pdf_backend_object_prop`

For tracking objects to allow finalisation.

```
\prop_new:N \g__pdf_backend_object_prop
```

(*End definition for* `\g__pdf_backend_object_prop`.)

`\__pdf_backend_object_new:n`
`\__pdf_backend_object_ref:n`

Declaring objects means reserving at the PDF level plus starting tracking.

```
\cs_new_protected:Npn \__pdf_backend_object_new:n #1
  {
⟨*luatex⟩
    \tex_pdfextension:D obj ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_pdfobj:D
⟨/pdftex⟩
      reserveobjnum ~
      \int_const:cn
        { c__pdf_object_ \tl_to_str:n {#1} _int }
⟨*luatex⟩
        { \tex_pdffeedback:D lastobj }
⟨/luatex⟩
⟨*pdftex⟩
        { \tex_pdflastobj:D }
⟨/pdftex⟩
  }
\cs_new:Npn \__pdf_backend_object_ref:n #1
  { \int_use:c { c__pdf_object_ \tl_to_str:n {#1} _int } ~ 0 ~ R }
```

(*End definition for* `\__pdf_backend_object_new:n` *and* `\__pdf_backend_object_ref:n`.)

`\__pdf_backend_object_write:nnn`
`\__pdf_backend_object_write:nnx`
`\__pdf_backend_object_write:nn`
`\__pdf_exp_not_i:nn`
`\__pdf_exp_not_ii:nn`

Writing the data needs a little information about the structure of the object.

```
\cs_new_protected:Npn \__pdf_backend_object_write:nnn #1#2#3
  {
⟨*luatex⟩
    \tex_immediate:D \tex_pdfextension:D obj ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_immediate:D \tex_pdfobj:D
⟨/pdftex⟩
```

```
      useobjnum ~
      \int_use:c
        { c__pdf_object_ \tl_to_str:n {#1} _int }
      \__pdf_backend_object_write:nn {#2} {#3}
  }
\cs_new:Npn \__pdf_backend_object_write:nn #1#2
  {
      \str_case:nn {#1}
        {
          { array } { { [ ~ \exp_not:n {#2} ~ ] } }
          { dict }  { { << ~ \exp_not:n {#2} ~ >> } }
          { fstream }
            {
              stream ~ attr ~ { \__pdf_exp_not_i:nn #2 } ~
                file ~ { \__pdf_exp_not_ii:nn #2 }
            }
          { stream }
            {
              stream ~ attr ~ { \__pdf_exp_not_i:nn #2 } ~
                { \__pdf_exp_not_ii:nn #2 }
            }
        }
  }
\cs_generate_variant:Nn \__pdf_backend_object_write:nnn { nnx }
\cs_new:Npn \__pdf_exp_not_i:nn #1#2 { \exp_not:n {#1} }
\cs_new:Npn \__pdf_exp_not_ii:nn #1#2 { \exp_not:n {#2} }
```

(*End definition for* `\__pdf_backend_object_write:nnn` *and others.*)

`\__pdf_backend_object_now:nn`
`\__pdf_backend_object_now:nx`

Much like writing, but direct creation.

```
\cs_new_protected:Npn \__pdf_backend_object_now:nn #1#2
  {
⟨*luatex⟩
    \tex_immediate:D \tex_pdfextension:D obj ~
⟨/luatex⟩
⟨*pdftex⟩
    \tex_immediate:D \tex_pdfobj:D
⟨/pdftex⟩
      \__pdf_backend_object_write:nn {#1} {#2}
  }
\cs_generate_variant:Nn \__pdf_backend_object_now:nn { nx }
```

(*End definition for* `\__pdf_backend_object_now:nn`*.*)

`\__pdf_backend_object_last:`

Much like annotation.

```
\cs_new:Npx \__pdf_backend_object_last:
  {
    \exp_not:N \int_value:w
⟨*luatex⟩
      \exp_not:N \tex_pdffeedback:D lastobj ~
⟨/luatex⟩
⟨*pdftex⟩
      \exp_not:N \tex_pdflastobj:D
⟨/pdftex⟩
      \c_space_tl 0 ~ R
  }
```

(*End definition for* `\__pdf_backend_object_last:`.)

`\__pdf_backend_pageobject_ref:n`

The usual wrapper situation; the three spaces here are essential.

```
\cs_new:Npx \__pdf_backend_pageobject_ref:n #1
  {
    \exp_not:N \int_value:w
⟨*luatex⟩
      \exp_not:N \tex_pdffeedback:D pageref
⟨/luatex⟩
⟨*pdftex⟩
      \exp_not:N \tex_pdfpageref:D
⟨/pdftex⟩
          \c_space_tl #1 \c_space_tl \c_space_tl \c_space_tl 0 ~ R
  }
```

(*End definition for* `\__pdf_backend_pageobject_ref:n`.)

### 6.3.4 Structure

`\__pdf_backend_compresslevel:n`
`\__pdf_backend_compress_objects:n`
`\__pdf_backend_objcompresslevel:n`

Simply pass data to the engine.

```
\cs_new_protected:Npn \__pdf_backend_compresslevel:n #1
  {
    \tex_global:D
⟨*luatex⟩
      \tex_pdfvariable:D compresslevel
⟨/luatex⟩
⟨*pdftex⟩
      \tex_pdfcompresslevel:D
⟨/pdftex⟩
        \int_value:w \int_eval:n {#1} \scan_stop:
  }
\cs_new_protected:Npn \__pdf_backend_compress_objects:n #1
  {
    \bool_if:nTF {#1}
      { \__pdf_backend_objcompresslevel:n { 2 } }
      { \__pdf_backend_objcompresslevel:n { 0 } }
  }
\cs_new_protected:Npn \__pdf_backend_objcompresslevel:n #1
  {
    \tex_global:D
⟨*luatex⟩
      \tex_pdfvariable:D objcompresslevel
⟨/luatex⟩
⟨*pdftex⟩
      \tex_pdfobjcompresslevel:D
⟨/pdftex⟩
        #1 \scan_stop:
  }
```

(*End definition for* `\__pdf_backend_compresslevel:n`, `\__pdf_backend_compress_objects:n`, *and* `\__pdf_backend_objcompresslevel:n`.)

__pdf_backend_version_major_gset:n
__pdf_backend_version_minor_gset:n

The availability of the primitive is not universal, so we have to test at load time.

```
\cs_new_protected:Npx \__pdf_backend_version_major_gset:n #1
  {
⟨*luatex⟩
    \int_compare:nNnT \tex_luatexversion:D > { 106 }
      {
        \exp_not:N \tex_global:D \tex_pdfvariable:D majorversion
          \exp_not:N \int_eval:n {#1} \scan_stop:
      }
⟨/luatex⟩
⟨*pdftex⟩
    \cs_if_exist:NT \tex_pdfmajorversion:D
      {
        \exp_not:N \tex_global:D \tex_pdfmajorversion:D
          \exp_not:N \int_eval:n {#1} \scan_stop:
      }
⟨/pdftex⟩
  }
\cs_new_protected:Npn \__pdf_backend_version_minor_gset:n #1
  {
    \tex_global:D
⟨*luatex⟩
      \tex_pdfvariable:D minorversion
⟨/luatex⟩
⟨*pdftex⟩
      \tex_pdfminorversion:D
⟨/pdftex⟩
        \int_eval:n {#1} \scan_stop:
  }
```

(*End definition for* __pdf_backend_version_major_gset:n *and* __pdf_backend_version_minor_gset:n.)

__pdf_backend_version_major:
__pdf_backend_version_minor:

As above.

```
\cs_new:Npx \__pdf_backend_version_major:
  {
⟨*luatex⟩
    \int_compare:nNnTF \tex_luatexversion:D > { 106 }
      { \exp_not:N \tex_the:D \tex_pdfvariable:D majorversion }
      { 1 }
⟨/luatex⟩
⟨*pdftex⟩
    \cs_if_exist:NTF \tex_pdfmajorversion:D
      { \exp_not:N \tex_the:D \tex_pdfmajorversion:D }
      { 1 }
⟨/pdftex⟩
  }
\cs_new:Npn \__pdf_backend_version_minor:
  {
    \tex_the:D
⟨*luatex⟩
      \tex_pdfvariable:D minorversion
⟨/luatex⟩
⟨*pdftex⟩
      \tex_pdfminorversion:D
⟨/pdftex⟩
  }
```

(*End definition for* __pdf_backend_version_major: *and* __pdf_backend_version_minor:.)

### 6.3.5 Marked content

__pdf_backend_bdc:nn
__pdf_backend_emc:

Simple wrappers. May need refinement: see https://chat.stackexchange.com/transcript/message/49970158#49970158.

```
\cs_new_protected:Npn \__pdf_backend_bdc:nn #1#2
  { \__kernel_backend_literal_page:n { /#1 ~ #2 ~ BDC } }
\cs_new_protected:Npn \__pdf_backend_emc:
  { \__kernel_backend_literal_page:n { EMC } }
```

(*End definition for* __pdf_backend_bdc:nn *and* __pdf_backend_emc:.)

```
⟨/luatex | pdftex⟩
```

## 6.4 dvipdfmx backend

```
⟨*dvipdfmx | xetex⟩
```

__pdf_backend:n
__pdf_backend:x

A generic function for the backend PDF specials: used where we can.

```
\cs_new_protected:Npx \__pdf_backend:n #1
  { \__kernel_backend_literal:n { pdf: #1 } }
\cs_generate_variant:Nn \__pdf_backend:n { x }
```

(*End definition for* __pdf_backend:n.)

### 6.4.1 Catalogue entries

__pdf_backend_catalog_gput:nn
__pdf_backend_info_gput:nn

```
\cs_new_protected:Npn \__pdf_backend_catalog_gput:nn #1#2
  { \__pdf_backend:n { put ~ @catalog << /#1 ~ #2 >> } }
\cs_new_protected:Npn \__pdf_backend_info_gput:nn #1#2
  { \__pdf_backend:n { docinfo << /#1 ~ #2 >> } }
```

(*End definition for* __pdf_backend_catalog_gput:nn *and* __pdf_backend_info_gput:nn.)

### 6.4.2 Objects

\g__pdf_backend_object_int
\g__pdf_backend_object_prop

For tracking objects to allow finalisation.

```
\int_new:N \g__pdf_backend_object_int
\prop_new:N \g__pdf_backend_object_prop
```

(*End definition for* \g__pdf_backend_object_int *and* \g__pdf_backend_object_prop.)

__pdf_backend_object_new:n
__pdf_backend_object_ref:n

Objects are tracked at the macro level, but we don't have to do anything at this stage.

```
\cs_new_protected:Npn \__pdf_backend_object_new:n #1
  {
    \int_gincr:N \g__pdf_backend_object_int
    \int_const:cn
      { c__pdf_object_ \tl_to_str:n {#1} _int }
      { \g__pdf_backend_object_int }
  }
\cs_new:Npn \__pdf_backend_object_ref:n #1
  { @pdf.obj \int_use:c { c__pdf_object_ \tl_to_str:n {#1} _int } }
```

(*End definition for* `\__pdf_backend_object_new:n` *and* `\__pdf_backend_object_ref:n`.)

`\__pdf_backend_object_write:nnn`
`\__pdf_backend_object_write:nnx`
`\__pdf_backend_object_write_array:nn`
`\__pdf_backend_object_write_dict:nn`
`\__pdf_backend_object_write_fstream:nn`
`\__pdf_backend_object_write_stream:nn`
`\__pdf_backend_object_write_stream:nnnn`

This is where we choose the actual type.

```
\cs_new_protected:Npn \__pdf_backend_object_write:nnn #1#2#3
  {
    \use:c { __pdf_backend_object_write_ #2 :nn }
      { \__pdf_backend_object_ref:n {#1} } {#3}
  }
\cs_generate_variant:Nn \__pdf_backend_object_write:nnn { nnx }
\cs_new_protected:Npn \__pdf_backend_object_write_array:nn #1#2
  {
    \__pdf_backend:x
      { obj ~ #1 ~ [ ~ \exp_not:n {#2} ~ ] }
  }
\cs_new_protected:Npn \__pdf_backend_object_write_dict:nn #1#2
  {
    \__pdf_backend:x
      { obj ~ #1 ~ << ~ \exp_not:n {#2} ~ >> }
  }
\cs_new_protected:Npn \__pdf_backend_object_write_fstream:nn #1#2
  { \__pdf_backend_object_write_stream:nnnn { f } {#1} #2 }
\cs_new_protected:Npn \__pdf_backend_object_write_stream:nn #1#2
  { \__pdf_backend_object_write_stream:nnnn { } {#1} #2 }
\cs_new_protected:Npn \__pdf_backend_object_write_stream:nnnn #1#2#3#4
  {
    \__pdf_backend:x
      {
        #1 stream ~ #2 ~
          ( \exp_not:n {#4} ) ~ << \exp_not:n {#3} >>
      }
  }
```

(*End definition for* `\__pdf_backend_object_write:nnn` *and others.*)

`\__pdf_backend_object_now:nn`
`\__pdf_backend_object_now:nx`

No anonymous objects with `dvipdfmx` so we have to give an object name.

```
\cs_new_protected:Npn \__pdf_backend_object_now:nn #1#2
  {
    \int_gincr:N \g__pdf_backend_object_int
    \exp_args:Nnx \use:c { __pdf_backend_object_write_ #1 :nn }
      { @pdf.obj \int_use:N \g__pdf_backend_object_int }
      {#2}
  }
\cs_generate_variant:Nn \__pdf_backend_object_now:nn { nx }
```

(*End definition for* `\__pdf_backend_object_now:nn`.)

`\__pdf_backend_object_last:`

```
\cs_new:Npn \__pdf_backend_object_last:
 { @pdf.obj \int_use:N \g__pdf_backend_object_int }
```

(*End definition for* `\__pdf_backend_object_last:`.)

`\__pdf_backend_pageobject_ref:n`

Page references are easy in `dvipdfmx`/XƎTEX.

```
\cs_new:Npn \__pdf_backend_pageobject_ref:n #1
  { @page #1 }
```

(*End definition for* `\__pdf_backend_pageobject_ref:n`.)

### 6.4.3 Annotations

\g__pdf_backend_annotation_int

Needed as objects which are not annotations could be created.

```
\int_new:N \g__pdf_backend_annotation_int
```

(*End definition for* `\g__pdf_backend_annotation_int`.)

__pdf_backend_annotation:nnnn

Simply pass the raw data through, just dealing with evaluation of dimensions.

```
\cs_new_protected:Npn \__pdf_backend_annotation:nnnn #1#2#3#4
  {
    \int_gincr:N \g__pdf_backend_object_int
    \int_gset_eq:NN \g__pdf_backend_annotation_int \g__pdf_backend_object_int
    \__pdf_backend:x
      {
        ann ~ @pdf.obj \int_use:N \g__pdf_backend_object_int \c_space_tl
        width  ~ \dim_eval:n {#1} ~
        height ~ \dim_eval:n {#2} ~
        depth  ~ \dim_eval:n {#3} ~
        << /Type /Annot #4 >>
      }
  }
```

(*End definition for* `\__pdf_backend_annotation:nnnn`.)

__pdf_backend_annotation_last:

```
\cs_new:Npn \__pdf_backend_annotation_last:
 { @pdf.obj \int_use:N \g__pdf_backend_annotation_int }
```

(*End definition for* `\__pdf_backend_annotation_last:`.)

\g__pdf_backend_link_int

To track annotations which are links.

```
\int_new:N \g__pdf_backend_link_int
```

(*End definition for* `\g__pdf_backend_link_int`.)

__pdf_backend_link_begin_goto:nnw
__pdf_backend_link_begin_user:nnw
__pdf_backend_link_begin:n
__pdf_backend_link_end:

All created using the same internals.

```
\cs_new_protected:Npn \__pdf_backend_link_begin_goto:nnw #1#2
  { \__pdf_backend_link_begin:n { #1 /Subtype /Link /A << /S /GoTo /D ( #2 ) >> } }
\cs_new_protected:Npn \__pdf_backend_link_begin_user:nnw #1#2
  { \__pdf_backend_link_begin:n {#1#2} }
\cs_new_protected:Npx \__pdf_backend_link_begin:n #1
  {
    \exp_not:N \int_gincr:N \exp_not:N  \g__pdf_backend_link_int
    \__pdf_backend:x
      {
         bann ~
         @pdf.lnk
         \exp_not:N \int_use:N \exp_not:N  \g__pdf_backend_link_int
         \c_space_tl
         <<
           /Type /Annot
           #1
         >>
      }
  }
\cs_new_protected:Npn \__pdf_backend_link_end:
  { \__pdf_backend:n { eann } }
```

(*End definition for* `\__pdf_backend_link_begin_goto:nnw` *and others.*)

`\__pdf_backend_link_last:`

Available using the backend mechanism with a suitably-recent version.

```
\cs_new:Npn \__pdf_backend_link_last:
  { @pdf.lnk \int_use:N \g__pdf_backend_link_int }
```

(*End definition for* `\__pdf_backend_link_last:`.)

`\__pdf_backend_link_margin:n`

Pass to `dvipdfmx`.

```
\cs_new_protected:Npn \__pdf_backend_link_margin:n #1
  { \__kernel_backend_literal:x { dvipdfmx:config~g~ \dim_eval:n {#1} } }
```

(*End definition for* `\__pdf_backend_link_margin:n`.)

`\__pdf_backend_destination:nn`
`\__pdf_backend_destination:nnnn`
`\__pdf_backend_destination_aux:nnnn`

Here, we need to turn the zoom into a scale. The method for `FitR` is from Alexander Grahn: the idea is to avoid needing to do any calculations in TEX by using the backend data for `@xpos` and `@ypos`. `/FitR` without rule spec doesn't work, so it falls back to `/Fit` here.

```
\cs_new_protected:Npn \__pdf_backend_destination:nn #1#2
  {
    \__pdf_backend:x
      {
        dest ~ ( \exp_not:n {#1} )
        [
          @thispage
          \str_case:nnF {#2}
            {
              { xyz }   { /XYZ ~ @xpos ~ @ypos ~ null }
              { fit }   { /Fit }
              { fitb }  { /FitB }
              { fitbh } { /FitBH }
              { fitbv } { /FitBV ~ @xpos }
              { fith }  { /FitH ~ @ypos }
              { fitv }  { /FitV ~ @xpos }
              { fitr }  { /Fit }
            }
            { /XYZ ~ @xpos ~ @ypos ~ \fp_eval:n { (#2) / 100 } }
        ]
      }
  }
\cs_new_protected:Npn \__pdf_backend_destination:nnnn #1#2#3#4
  {
    \exp_args:Ne \__pdf_backend_destination_aux:nnnn
      { \dim_eval:n {#2} } {#1} {#3} {#4}
  }
\cs_new_protected:Npn \__pdf_backend_destination_aux:nnnn #1#2#3#4
  {
    \vbox_to_zero:n
      {
        \__kernel_kern:n {#4}
        \hbox:n
          {
            \__pdf_backend:n { obj ~ @pdf_ #2 _llx ~ @xpos }
            \__pdf_backend:n { obj ~ @pdf_ #2 _lly ~ @ypos }
          }
        \tex_vss:D
      }
    \__kernel_kern:n {#1}
    \vbox_to_zero:n
      {
        \__kernel_kern:n { -#3 }
        \hbox:n
          {
            \__pdf_backend:n
              {
                dest ~ (#2)
                [
                  @thispage
                  /FitR ~
                    @pdf_ #2 _llx ~ @pdf_ #2 _lly ~
                    @xpos ~ @ypos
                ]
              }
          }
        \tex_vss:D
      }
    \__kernel_kern:n { -#1 }
  }
```

(*End definition for* `\__pdf_backend_destination:nn`*,* `\__pdf_backend_destination:nnnn`*, and* `\__pdf_backend_destination_aux:nnnn`*.*)

### 6.4.4 Structure

__pdf_backend_compresslevel:n
__pdf_backend_compress_objects:n

Pass data to the backend: these are a one-shot.

```
\cs_new_protected:Npn \__pdf_backend_compresslevel:n #1
  { \__kernel_backend_literal:x { dvipdfmx:config~z~ \int_eval:n {#1} } }
\cs_new_protected:Npn \__pdf_backend_compress_objects:n #1
  {
    \bool_if:nF {#1}
      { \__kernel_backend_literal:n { dvipdfmx:config~C~0x40 } }
  }
```

(*End definition for* `\__pdf_backend_compresslevel:n` *and* `\__pdf_backend_compress_objects:n`*.*)

__pdf_backend_version_major_gset:n
__pdf_backend_version_minor_gset:n

We start with the assumption that the default is active.

```
\cs_new_protected:Npn \__pdf_backend_version_major_gset:n #1
  {
    \cs_gset:Npx \__pdf_backend_version_major: { \int_eval:n {#1} }
    \__kernel_backend_literal:x { pdf:majorversion~ \__pdf_backend_version_major: }
  }
\cs_new_protected:Npn \__pdf_backend_version_minor_gset:n #1
  {
    \cs_gset:Npx \__pdf_backend_version_minor: { \int_eval:n {#1} }
    \__kernel_backend_literal:x { pdf:minorversion~ \__pdf_backend_version_minor: }
  }
```

(*End definition for* `\__pdf_backend_version_major_gset:n` *and* `\__pdf_backend_version_minor_gset:n`*.*)

__pdf_backend_version_major:
__pdf_backend_version_minor:

We start with the assumption that the default is active.

```
\cs_new:Npn \__pdf_backend_version_major: { 1 }
\cs_new:Npn \__pdf_backend_version_minor: { 5 }
```

(*End definition for* `\__pdf_backend_version_major:` *and* `\__pdf_backend_version_minor:`.)

### 6.4.5 Marked content

__pdf_backend_bdc:nn
__pdf_backend_emc:

Simple wrappers. May need refinement: see https://chat.stackexchange.com/transcript/message/49970158#49970158.

```
\cs_new_protected:Npn \__pdf_backend_bdc:nn #1#2
  { \__kernel_backend_literal_page:n { /#1 ~ #2 ~ BDC } }
\cs_new_protected:Npn \__pdf_backend_emc:
  { \__kernel_backend_literal_page:n { EMC } }
```

(*End definition for* `\__pdf_backend_bdc:nn` *and* `\__pdf_backend_emc:`.)

```
⟨/dvipdfmx | xetex⟩
```

## 6.5 dvisvgm backend

```
⟨*dvisvgm⟩
```

### 6.5.1 Annotations

__pdf_backend_annotation:nnnn

```
\cs_new_protected:Npn \__pdf_backend_annotation:nnnn #1#2#3#4 { }
```

(*End definition for* `\__pdf_backend_annotation:nnnn`.)

__pdf_backend_annotation_last:

```
\cs_new:Npn \__pdf_backend_annotation_last: { }
```

(*End definition for* `\__pdf_backend_annotation_last:`.)

__pdf_backend_link_begin_goto:nnw
__pdf_backend_link_begin_user:nnw
__pdf_backend_link_begin:nnnw
__pdf_backend_link_end:

```
\cs_new_protected:Npn \__pdf_backend_link_begin_goto:nnw #1#2 { }
\cs_new_protected:Npn \__pdf_backend_link_begin_user:nnw #1#2 { }
\cs_new_protected:Npn \__pdf_backend_link_begin:nnnw #1#2#3 { }
\cs_new_protected:Npn \__pdf_backend_link_end: { }
```

(*End definition for* `\__pdf_backend_link_begin_goto:nnw` *and others.*)

__pdf_backend_link_last:

```
\cs_new:Npx \__pdf_backend_link_last: { }
```

(*End definition for* `\__pdf_backend_link_last:`.)

__pdf_backend_link_margin:n

A simple task: pass the data to the primitive.

```
\cs_new_protected:Npn \__pdf_backend_link_margin:n #1 { }
```

(*End definition for* `\__pdf_backend_link_margin:n`.)

__pdf_backend_destination:nn
__pdf_backend_destination:nnnn

```
\cs_new_protected:Npn \__pdf_backend_destination:nn #1#2 { }
\cs_new_protected:Npn \__pdf_backend_destination:nnnn #1#2#3#4 { }
```

(*End definition for* `\__pdf_backend_destination:nn` *and* `\__pdf_backend_destination:nnnn`.)

### 6.5.2 Catalogue entries

__pdf_backend_catalog_gput:nn
__pdf_backend_info_gput:nn

No-op.

```
\cs_new_protected:Npn \__pdf_backend_catalog_gput:nn #1#2 { }
\cs_new_protected:Npn \__pdf_backend_info_gput:nn #1#2 { }
```

(*End definition for* `\__pdf_backend_catalog_gput:nn` *and* `\__pdf_backend_info_gput:nn`.)

### 6.5.3 Objects

__pdf_backend_object_new:n
__pdf_backend_object_ref:n
__pdf_backend_object_write:nnn
__pdf_backend_object_write:nx
__pdf_backend_object_now:nn
__pdf_backend_object_now:nx
__pdf_backend_object_last:
__pdf_backend_pageobject_ref:n

All no-ops here.

```
\cs_new_protected:Npn \__pdf_backend_object_new:nn #1 { }
\cs_new:Npn \__pdf_backend_object_ref:n #1 { }
\cs_new_protected:Npn \__pdf_backend_object_write:nnn #1#2#3 { }
\cs_new_protected:Npn \__pdf_backend_object_write:nnx #1#2#3 { }
\cs_new_protected:Npn \__pdf_backend_object_now:nn #1#2 { }
\cs_new_protected:Npn \__pdf_backend_object_now:nx #1#2 { }
\cs_new:Npn \__pdf_backend_object_last: { }
\cs_new:Npn \__pdf_backend_pageobject_ref:n #1 { }
```

(*End definition for* `\__pdf_backend_object_new:n` *and others.*)

### 6.5.4 Structure

__pdf_backend_compresslevel:n
__pdf_backend_compress_objects:n

These are all no-ops.

```
\cs_new_protected:Npn \__pdf_backend_compresslevel:n #1 { }
\cs_new_protected:Npn \__pdf_backend_compress_objects:n #1 { }
```

(*End definition for* `\__pdf_backend_compresslevel:n` *and* `\__pdf_backend_compress_objects:n`.)

__pdf_backend_version_major_gset:n
__pdf_backend_version_minor_gset:n

Data not available!

```
\cs_new_protected:Npn \__pdf_backend_version_major_gset:n #1 { }
\cs_new_protected:Npn \__pdf_backend_version_minor_gset:n #1 { }
```

(*End definition for* `\__pdf_backend_version_major_gset:n` *and* `\__pdf_backend_version_minor_gset:n`.)

__pdf_backend_version_major:
__pdf_backend_version_minor:

Data not available!

```
\cs_new:Npn \__pdf_backend_version_major: { -1 }
\cs_new:Npn \__pdf_backend_version_minor: { -1 }
```

(*End definition for* `\__pdf_backend_version_major:` *and* `\__pdf_backend_version_minor:`.)

__pdf_backend_bdc:nn
__pdf_backend_emc:

More no-ops.

```
\cs_new_protected:Npn \__pdf_backend_bdc:nn #1#2 { }
\cs_new_protected:Npn \__pdf_backend_emc: { }
```

(*End definition for* `\__pdf_backend_bdc:nn` *and* `\__pdf_backend_emc:`.)

```
⟨/dvisvgm⟩
```

## 6.6 PDF Page size (media box)

For setting the media box, the split between backends is somewhat different to other areas, thus we approach this separately. The code here assumes a recent LaTeX 2ε: that is ensured at the level above.

```
⟨*dvipdfmx | dvips⟩
```

\__pdf_backend_pagesize_gset:nn

This is done as a backend literal, so we deal with it using the shipout hook.

```
\cs_new_protected:Npn \__pdf_backend_pagesize_gset:nn #1#2
  {
    \__kernel_backend_first_shipout:n
      {
        \__kernel_backend_literal:e
          {
⟨*dvipdfmx⟩
            pdf:pagesize ~
              width  ~ \dim_eval:n {#1} ~
              height ~ \dim_eval:n {#2}
⟨/dvipdfmx⟩
⟨*dvips⟩
            papersize = \dim_eval:n {#1} , \dim_eval:n {#2}
⟨/dvips⟩
          }
      }
  }
```

(*End definition for* `\__pdf_backend_pagesize_gset:nn`.)

```
⟨/dvipdfmx | dvips⟩
```

```
⟨*luatex | pdftex | xetex⟩
```

\__pdf_backend_pagesize_gset:nn

Pass to the primitives.

```
\cs_new_protected:Npn \__pdf_backend_pagesize_gset:nn #1#2
  {
    \dim_gset:Nn \tex_pagewidth:D  {#1}
    \dim_gset:Nn \tex_pageheight:D {#2}
  }
```

(*End definition for* `\__pdf_backend_pagesize_gset:nn`.)

```
⟨/luatex | pdftex | xetex⟩
```

```
⟨*dvisvgm⟩
```

\__pdf_backend_pagesize_gset:nn

A no-op.

```
\cs_new_protected:Npn \__pdf_backend_pagesize_gset:nn #1#2 { }
```

(*End definition for* `\__pdf_backend_pagesize_gset:nn`.)

```
⟨/dvisvgm⟩
```

```
⟨/package⟩
```

# 7 **l3backend-opacity** Implementation

```
⟨*package⟩
⟨@@=opacity⟩
```

Although opacity is not color, it needs to be managed in a somewhat similar way: using a dedicated stack if possible. Depending on the backend, that may not be possible. There is also the need to cover fill/stroke setting as well as more general running opacity. It is easiest to describe the value used in terms of opacity, although commonly this is referred to as transparency.

```
⟨*dvips⟩
```

`\__opacity_backend_select:n`
`\__opacity_backend_select_aux:n`
`\__opacity_backend_fill:n`
`\__opacity_backend_stroke:n`
`\__opacity_backend:nnn`
`\__opacity_backend:xnn`

No stack so set values directly. The need to deal with Distiller and Ghostscript separately means we use a common auxiliary: the two systems require different PostScript for transparency. This is of course not quite as efficient as doing one test for setting all transparency, but it keeps things clearer here. Thanks to Alex Grahn for the detail on testing for GhostScript.

```
\cs_new_protected:Npn \__opacity_backend_select:n #1
  {
    \exp_args:Nx \__opacity_backend_select_aux:n
      { \fp_eval:n { min(max(0,#1),1) } }
  }
\cs_new_protected:Npn \__opacity_backend_select_aux:n #1
  {
    \__opacity_backend:nnn {#1} { fill }   { ca }
    \__opacity_backend:nnn {#1} { stroke } { CA }
  }
\cs_new_protected:Npn \__opacity_backend_fill:n #1
  {
    \__opacity_backend:xnn
      { \fp_eval:n { min(max(0,#1),1) } }
      { fill }
      { ca }
  }
\cs_new_protected:Npn \__opacity_backend_stroke:n #1
  {
    \__opacity_backend:xnn
      { \fp_eval:n { min(max(0,#1),1) } }
      { stroke }
      { CA }
  }
\cs_new_protected:Npn \__opacity_backend:nnn #1#2#3
  {
    \__kernel_backend_postscript:n
      {
        product ~ (Ghostscript) ~ search
          {
            pop ~ pop ~ pop ~
            #1 ~ .set #2 constantalpha
          }
          {
            pop ~
            mark ~
            /#3 ~ #1
            /SetTransparency ~
            pdfmark
          }
        ifelse
      }
  }
\cs_generate_variant:Nn \__opacity_backend:nnn { x }
```

(*End definition for* `\__opacity_backend_select:n` *and others.*)

```
⟨/dvips⟩
```

```
⟨*dvipdfmx | luatex | pdftex | xetex⟩
```

`\c__opacity_backend_stack_int`

Set up a stack, where that is applicable.

```
\bool_lazy_and:nnT
  { \cs_if_exist_p:N \pdfmanagement_if_active_p: }
  { \pdfmanagement_if_active_p:}
  {
⟨*luatex | pdftex⟩
    \__kernel_color_backend_stack_init:Nnn \c__opacity_backend_stack_int
      { page ~ direct } { /opacity 1 ~ gs }
⟨/luatex | pdftex⟩
    \pdfmanagement_add:nnn { Page / Resources / ExtGState }
      { opacity 1 } { << /ca ~ 1 /CA ~ 1 >> }
  }
```

(*End definition for* `\c__opacity_backend_stack_int`.)

`\l__opacity_backend_fill_tl`
`\l__opacity_backend_stroke_tl`

We use `tl` here for speed: at the backend, this should be reasonable.

```
\tl_new:N \l__opacity_backend_fill_tl
\tl_new:N \l__opacity_backend_stroke_tl
```

(*End definition for* `\l__opacity_backend_fill_tl` *and* `\l__opacity_backend_stroke_tl`.)

`\__opacity_backend_select:n`
`\__opacity_backend_select_aux:n`
`\__opacity_backend_reset:`

Other than the need to evaluate the opacity as an `fp`, much the same as color.

```
\cs_new_protected:Npn \__opacity_backend_select:n #1
 {
   \exp_args:Nx \__opacity_backend_select_aux:n
     { \fp_eval:n { min(max(0,#1),1) } }
 }
\cs_new_protected:Npn \__opacity_backend_select_aux:n #1
  {
    \tl_set:Nn \l__opacity_backend_fill_tl {#1}
    \tl_set:Nn \l__opacity_backend_stroke_tl {#1}
    \pdfmanagement_add:nnn { Page / Resources / ExtGState }
      { opacity #1 }
      { << /ca ~ #1 /CA ~ #1 >> }
⟨*dvipdfmx | xetex⟩
    \__kernel_backend_literal_pdf:n
⟨/dvipdfmx | xetex⟩
⟨*luatex | pdftex⟩
    \__kernel_color_backend_stack_push:nn \c__opacity_backend_stack_int
⟨/luatex | pdftex⟩
      { /opacity #1 ~ gs }
    \group_insert_after:N \__opacity_backend_reset:
  }
\bool_lazy_and:nnF
  { \cs_if_exist_p:N \pdfmanagement_if_active_p: }
  { \pdfmanagement_if_active_p:}
  {
    \cs_gset_protected:Npn \__opacity_backend_select_aux:n #1 { }
  }
\cs_new_protected:Npn \__opacity_backend_reset:
 {
⟨*dvipdfmx | xetex⟩
   \__kernel_backend_literal_pdf:n
      { /opacity1 ~ gs }
⟨/dvipdfmx | xetex⟩
⟨*luatex | pdftex⟩
    \__kernel_color_backend_stack_pop:n \c__opacity_backend_stack_int
⟨/luatex | pdftex⟩
 }
```

(*End definition for* `\__opacity_backend_select:n`*,* `\__opacity_backend_select_aux:n`*, and* `\__opacity_backend_reset:`*.*)

`\__opacity_backend_fill:n`
`\__opacity_backend_stroke:n`
`\__opacity_backend_fillstroke:nn`
`\__opacity_backend_fillstroke:xx`

For separate fill and stroke, we need to work out if we need to do more work or if we can stick to a single setting.

```
\cs_new_protected:Npn \__opacity_backend_fill:n #1
  {
    \__opacity_backend_fill_stroke:xx
      { \fp_eval:n { min(max(0,#1),1) } }
      \l__opacity_backend_stroke_tl
  }
\cs_new_protected:Npn \__opacity_backend_stroke:n #1
  {
    \__opacity_backend_fill_stroke:xx
      \l__opacity_backend_fill_tl
      { \fp_eval:n { min(max(0,#1),1) } }
  }
\cs_new_protected:Npn \__opacity_backend_fill_stroke:nn #1#2
  {
    \str_if_eq:nnTF {#1} {#2}
      { \__opacity_backend_select_aux:n {#1} }
      {
        \tl_set:Nn \l__opacity_backend_fill_tl {#1}
        \tl_set:Nn \l__opacity_backend_stroke_tl {#2}
        \pdfmanagement_add:nnn { Page / Resources / ExtGState }
          { opacity.fill #1 }
          { << /ca ~ #1 >> }
        \pdfmanagement_add:nnn { Page / Resources / ExtGState }
          { opacity.stroke #1 }
          { << /CA ~ #2 >> }
⟨*dvipdfmx | xetex⟩
    \__kernel_backend_literal_pdf:n
⟨/dvipdfmx | xetex⟩
⟨*luatex | pdftex⟩
    \__kernel_color_backend_stack_push:nn \c__opacity_backend_stack_int
⟨/luatex | pdftex⟩
```

```
        { /opacity.fill #1 ~ gs /opacity.stroke #2 ~ gs }
        \group_insert_after:N \__opacity_backend_reset:
      }
  }
\cs_generate_variant:Nn \__opacity_backend_fill_stroke:nn { xx }
```

(*End definition for* `\__opacity_backend_fill:n`*,* `\__opacity_backend_stroke:n`*, and* `\__opacity_backend_fillstroke:nn`*.*)

```
⟨/dvipdfmx | luatex | pdftex | xetex⟩
```

```
⟨*dvisvgm⟩
```

`\__opacity_backend_select:n`
`\__opacity_backend_fill:n`
`\__opacity_backend_stroke:n`
`\__opacity_backend:nn`

Once again, we use a scope here. There is a general opacity function for SVG, but that is of course not set up using the stack.

```
\cs_new_protected:Npn \__opacity_backend_select:n #1
  { \__opacity_backend:nn {#1} { } }
\cs_new_protected:Npn \__opacity_backend_fill:n #1
  { \__opacity_backend:nn {#1} { fill- } }
\cs_new_protected:Npn \__opacity_backend_stroke:n #1
  { \__opacity_backend:nn { {#1} } { stroke- } }
\cs_new_protected:Npn \__opacity_backend:nn #1#2
  { \__kernel_backend_scope:x { #2 opacity = " \fp_eval:n { min(max(0,#1),1) } " } }
```

(*End definition for* `\__opacity_backend_select:n` *and others.*)

```
⟨/dvisvgm⟩
```

```
⟨/package⟩
```

# 8 **l3backend-header** Implementation

```
⟨*dvips & header⟩
```

`color.sc`

Empty definition for color at the top level.

```
/color.sc { } def
```

(*End definition for* `color.sc`*. This function is documented on page* **??**.)

`TeXcolorseparation`
`separation`

Support for separation/spot colors: this strange naming is so things work with the color stack.

```
TeXDict begin
/TeXcolorseparation { setcolor } def
end
```

(*End definition for* `TeXcolorseparation` *and* `separation`*. These functions are documented on page* **??**.)

`pdf.globaldict`

A small global dictionary for backend use.

```
true setglobal
/pdf.globaldict 4 dict def
false setglobal
```

(*End definition for* `pdf.globaldict`*. This function is documented on page* **??**.)

**pdf.cvs**
**pdf.dvi.pt**
**pdf.pt.dvi**
**pdf.rect.ht**

Small utilities for PostScript manipulations. Conversion to DVI dimensions is done here to allow for `Resolution`. The total height of a rectangle (an array) needs a little maths, in contrast to simply extracting a value.

```
/pdf.cvs { 65534 string cvs } def
/pdf.dvi.pt { 72.27 mul Resolution div } def
/pdf.pt.dvi { 72.27 div Resolution mul } def
/pdf.rect.ht { dup 1 get neg exch 3 get add } def
```

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

**pdf.linkmargin**
**pdf.linkdp.pad**
**pdf.linkht.pad**

Settings which are defined up-front in `SDict`.

```
/pdf.linkmargin { 1 pdf.pt.dvi } def
/pdf.linkdp.pad { 0 } def
/pdf.linkht.pad { 0 } def
```

(*End definition for* `pdf.linkmargin`*,* `pdf.linkdp.pad`*, and* `pdf.linkht.pad`*. These functions are documented on page* **??**.)

**pdf.rect**
**pdf.save.ll**
**pdf.save.ur**
**pdf.save.linkll**
**pdf.save.linkur**
**pdf.llx**
**pdf.lly**
**pdf.urx**
**pdf.ury**

Functions for marking the limits of an annotation/link, plus drawing the border. We separate links for generic annotations to support adding a margin and setting a minimal size.

```
/pdf.rect
  { /Rect [ pdf.llx pdf.lly pdf.urx pdf.ury ] } def
/pdf.save.ll
  {
    currentpoint
    /pdf.lly exch def
    /pdf.llx exch def
  }
    def
/pdf.save.ur
  {
    currentpoint
    /pdf.ury exch def
    /pdf.urx exch def
  }
    def
/pdf.save.linkll
  {
    currentpoint
    pdf.linkmargin add
    pdf.linkdp.pad add
    /pdf.lly exch def
    pdf.linkmargin sub
    /pdf.llx exch def
  }
    def
/pdf.save.linkur
  {
    currentpoint
    pdf.linkmargin sub
    pdf.linkht.pad sub
    /pdf.ury exch def
    pdf.linkmargin add
    /pdf.urx exch def
  }
    def
```

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

`pdf.dest.anchor`
`pdf.dest.x`
`pdf.dest.y`
`pdf.dest.point`
`pdf.dest2device`
`pdf.dev.x`
`pdf.dev.y`
`pdf.tmpa`
`pdf.tmpb`
`pdf.tmpc`
`pdf.tmpd`

For finding the anchor point of a destination link. We make the use case a separate function as it comes up a lot, and as this makes it easier to adjust if we need additional effects. We also need a more complex approach to convert a co-ordinate pair correctly when defining a rectangle: this can otherwise be out when using a landscape page. (Thanks to Alexander Grahn for the approach here.)

```
/pdf.dest.anchor
  {
    currentpoint exch
    pdf.dvi.pt 72 add
    /pdf.dest.x exch def
    pdf.dvi.pt
    vsize 72 sub exch sub
    /pdf.dest.y exch def
  }
    def
/pdf.dest.point
  { pdf.dest.x pdf.dest.y } def
/pdf.dest2device
  {
    /pdf.dest.y exch def
    /pdf.dest.x exch def
    matrix currentmatrix
    matrix defaultmatrix
    matrix invertmatrix
    matrix concatmatrix
    cvx exec
    /pdf.dev.y exch def
    /pdf.dev.x exch def
    /pdf.tmpd exch def
    /pdf.tmpc exch def
    /pdf.tmpb exch def
    /pdf.tmpa exch def
    pdf.dest.x pdf.tmpa mul
      pdf.dest.y pdf.tmpc mul add
      pdf.dev.x add
    pdf.dest.x pdf.tmpb mul
     pdf.dest.y pdf.tmpd mul add
     pdf.dev.y add
  }
    def
```

(*End definition for* `pdf.dest.anchor` *and others. These functions are documented on page* **??**.)

`pdf.bordertracking`
`pdf.bordertracking.begin`
`pdf.bordertracking.end`
`pdf.leftboundary`
`pdf.rightboundary`
`pdf.brokenlink.rect`
`pdf.brokenlink.skip`
`pdf.brokenlink.dict`
`pdf.bordertracking.endpage`
`pdf.bordertracking.continue`
`pdf.originx`
`pdf.originy`

To know where a breakable link can go, we need to track the boundary rectangle. That can be done by hooking into `a` and `x` operations: those names have to be retained. The boundary is stored at the end of the operation. Special effort is needed at the start and end of pages (or rather galleys), such that everything works properly.

```
/pdf.bordertracking false def
```

```
/pdf.bordertracking.begin
  {
    SDict /pdf.bordertracking true put
    SDict /pdf.leftboundary undef
    SDict /pdf.rightboundary undef
    /a where
      {
        /a
          {
            currentpoint pop
            SDict /pdf.rightboundary known dup
              {
                SDict /pdf.rightboundary get 2 index lt
                  { not }
                if
              }
            if
              { pop }
              { SDict exch /pdf.rightboundary exch put }
            ifelse
            moveto
            currentpoint pop
            SDict /pdf.leftboundary known dup
              {
                SDict /pdf.leftboundary get 2 index gt
                  { not }
                if
              }
            if
              { pop }
              { SDict exch /pdf.leftboundary exch put }
            ifelse
          }
        put
      }
    if
  }
    def
/pdf.bordertracking.end
  {
    /a where { /a { moveto } put } if
    /x where { /x { 0 exch rmoveto } put } if
    SDict /pdf.leftboundary known
      { pdf.outerbox 0 pdf.leftboundary put }
    if
    SDict /pdf.rightboundary known
      { pdf.outerbox 2 pdf.rightboundary put }
    if
    SDict /pdf.bordertracking false put
  }
    def
  /pdf.bordertracking.endpage
{
  pdf.bordertracking
```

```
    {
      pdf.bordertracking.end
      true setglobal
      pdf.globaldict
        /pdf.brokenlink.rect [ pdf.outerbox aload pop ] put
      pdf.globaldict
        /pdf.brokenlink.skip pdf.baselineskip put
      pdf.globaldict
        /pdf.brokenlink.dict
          pdf.link.dict pdf.cvs put
      false setglobal
      mark pdf.link.dict cvx exec /Rect
        [
          pdf.llx
          pdf.lly
          pdf.outerbox 2 get pdf.linkmargin add
          currentpoint exch pop
          pdf.outerbox pdf.rect.ht sub pdf.linkmargin sub
        ]
      /ANN pdf.pdfmark
    }
  if
}
  def
/pdf.bordertracking.continue
  {
    /pdf.link.dict pdf.globaldict
      /pdf.brokenlink.dict get def
    /pdf.outerbox pdf.globaldict
      /pdf.brokenlink.rect get def
    /pdf.baselineskip pdf.globaldict
      /pdf.brokenlink.skip get def
    pdf.globaldict dup dup
    /pdf.brokenlink.dict undef
    /pdf.brokenlink.skip undef
    /pdf.brokenlink.rect undef
    currentpoint
    /pdf.originy exch def
    /pdf.originx exch def
    /a where
      {
        /a
          {
            moveto
            SDict
            begin
            currentpoint pdf.originy ne exch
              pdf.originx ne or
              {
                pdf.save.linkll
                /pdf.lly
                  pdf.lly pdf.outerbox 1 get sub def
                pdf.bordertracking.begin
              }
            if
            end
          }
        put
      }
    if
    /x where
      {
        /x
          {
            0 exch rmoveto
            SDict
            begin
            currentpoint
            pdf.originy ne exch pdf.originx ne or
              {
                pdf.save.linkll
                /pdf.lly
                  pdf.lly pdf.outerbox 1 get sub def
                pdf.bordertracking.begin
              }
            if
            end
          }
        put
      }
    if
  }
    def
```

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

`pdf.breaklink`
`pdf.breaklink.write`
`pdf.count`
`pdf.currentrect`

Dealing with link breaking itself has multiple stage. The first step is to find the `Rect` entry in the dictionary, looping over key–value pairs. The first line is handled first, adjusting the rectangle to stay inside the text area. The second phase is a loop over the height of the bulk of the link area, done on the basis of a number of baselines. Finally, the end of the link area is tidied up, again from the boundary of the text area.

```
/pdf.breaklink
  {
    pop
    counttomark 2 mod 0 eq
      {
        counttomark /pdf.count exch def
          {
           pdf.count 0 eq { exit } if
           counttomark 2 roll
           1 index /Rect eq
             {
               dup 4 array copy
               dup dup
                 1 get
                 pdf.outerbox pdf.rect.ht
                 pdf.linkmargin 2 mul add sub
                 3 exch put
```

```
 dup
   pdf.outerbox 2 get
   pdf.linkmargin add
   2 exch put
 dup dup
   3 get
   pdf.outerbox pdf.rect.ht
   pdf.linkmargin 2 mul add add
   1 exch put
 /pdf.currentrect exch  def
 pdf.breaklink.write
   {
     pdf.currentrect
     dup
       pdf.outerbox 0 get
       pdf.linkmargin sub
       0 exch put
     dup
       pdf.outerbox 2 get
       pdf.linkmargin add
       2 exch put
     dup dup
       1 get
       pdf.baselineskip add
       1 exch put
     dup dup
       3 get
       pdf.baselineskip add
       3 exch put
     /pdf.currentrect exch def
     pdf.breaklink.write
    }
  1 index 3 get
  pdf.linkmargin 2 mul add
  pdf.outerbox pdf.rect.ht add
  2 index 1 get sub
  pdf.baselineskip div round cvi 1 sub
  exch
repeat
pdf.currentrect
dup
  pdf.outerbox 0 get
  pdf.linkmargin sub
  0 exch put
dup dup
  1 get
  pdf.baselineskip add
  1 exch put
dup dup
  3 get
  pdf.baselineskip add
  3 exch put
dup 2 index 2 get  2 exch put
/pdf.currentrect exch def
```

```
       pdf.breaklink.write
       SDict /pdf.pdfmark.good false put
       exit
      }
      { pdf.count 2 sub /pdf.count exch def }
     ifelse
    }
   loop
  }
 if
 /ANN
}
 def
/pdf.breaklink.write
 {
  counttomark 1 sub
  index /_objdef eq
   {
    counttomark -2 roll
    dup wcheck
     {
      readonly
      counttomark 2 roll
     }
     { pop pop }
    ifelse
   }
  if
  counttomark 1 add copy
  pop pdf.currentrect
  /ANN pdfmark
 }
  def
```

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

`pdf.pdfmark`
`pdf.pdfmark.good`
`pdf.outerbox`
`pdf.baselineskip`
`pdf.pdfmark.dict`

The business end of breaking links starts by hooking into `pdfmarks`. Unlike hypdvips, we avoid altering any links we have not created by using a copy of the core `pdfmarks` function. Only mark types which are known are altered. At present, this is purely `ANN` marks, which are measured relative to the size of the baseline skip. If they are more than one apparent line high, breaking is applied.

```
/pdf.pdfmark
 {
  SDict /pdf.pdfmark.good true put
  dup /ANN eq
   {
    pdf.pdfmark.store
    pdf.pdfmark.dict
     begin
      Subtype /Link eq
      currentdict /Rect known and
      SDict /pdf.outerbox known and
      SDict /pdf.baselineskip known and
       {
```

```
                Rect 3 get
                pdf.linkmargin 2 mul add
                pdf.outerbox pdf.rect.ht add
                Rect 1 get sub
                pdf.baselineskip div round cvi 0 gt
                  { pdf.breaklink }
                if
              }
            if
          end
        SDict /pdf.outerbox undef
        SDict /pdf.baselineskip undef
        currentdict /pdf.pdfmark.dict undef
      }
    if
    pdf.pdfmark.good
      { pdfmark }
      { cleartomark }
    ifelse
  }
    def
/pdf.pdfmark.store
  {
    /pdf.pdfmark.dict 65534 dict def
    counttomark 1 add copy
    pop
      {
        dup mark eq
          {
            pop
            exit
          }
          {
            pdf.pdfmark.dict
            begin def end
          }
        ifelse
      }
    loop
}
  def
```

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

⟨/dvips & header⟩

# 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.