Architectural guides for Bonn — book production with ConTEXt

Henning Hraban Ramm

"Architekturführer der Werkstatt Baukultur Bonn"

At the University of Bonn, there is a group of scholars who care about the modernist buildings that were built after the second world war, when Bonn was Germany's capital. They do research, offer guided tours and also publish a series of little architectural guides. My publisher colleagues were already involved with them when they studied German literature in Bonn, so we took over this series when we founded Dreiviertelhaus publishers in 2017. As it happened, I took over the design even earlier, since their designer had no time after becoming a mother.

The design is rather simple, so I decided to do it in ConTEXt (instead of InDesign). But the structure of every booklet is unique, since they have a wide variety in the contents: Some volumes are about only one building, others about an ensemble or a housing estate or a themed collection such as "buildings of the university"; some are by a single author, others collect contributions by several authors. That means I must adapt the table of contents and the structure of titles in every other volume. (Figure 1 is an example.)

Figure 1: symbolic #0 guide

Design and layout

Since we use many photos from archives, most pages are black-and-white, as well as the front covers. But the booklets are printed in color, because we also show current pictures, and sometimes color is important, for example with stained glass windows in churches or other artwork.

For the cover, we try to use one image front-to-back, but it's just not possible for every volume to find a landscape photo where the important part of a building is on the right-hand side. Since my setup expects one image, I glue different photos for front and back together in such cases. (Figure 2.)

Interior images are often full page, sometimes even over a double page spread.

Images that cover the full width or height of a page need to be a few millimeters bigger to avoid problems in paper trimming; this is called *bleed*. This affects not only full page images, but everything that touches paper edges. We also have images that stay within the type area.

Most images have captions. On full page images, the caption is moved into the image and gets a background shadow to increase readability. (Figure 3.)

Figure 2: cover images (back and front)

Henning Hraban Ramm
doi.org/10.47397/tb/44-2/tb137ramm-books

Figure 3: example spreads with half-page and full-page images

Maps

There's another tricky subject in these architectural guides, namely city maps.

For my architectural guide on the Kyrgyz capital Bishkek, I got experience with processing OpenStreetMap data for custom maps.

I'm using Maperitive,[1] because it allows for batch processing. Maperitive is written in .NET, and I run it on my Mac with Mono. It's horribly slow, the programming interface is severely underdocumented, and the latest version is from 2018, but it's still the best choice and I somehow manage to get what I want.

What I want is also a custom style with very subdued colors, nearly black and white, and not many location markers for shops etc. Maperitive uses style sheets that are somewhat similar to CSS, so you have selectors and style instructions.

The output is SVG, and I use Inkscape to convert it to PDF for inclusion. ConTEXt LMTX doesn't need this any more and can process SVG on its own via a MetaPost conversion. But when I made the latest

[1] `https://maperitive.net`

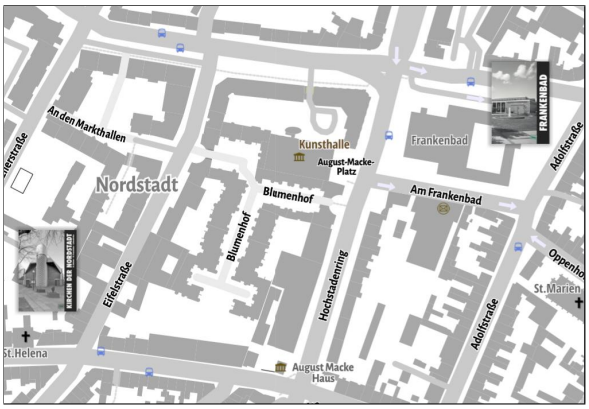


Figure 4: city map, with advertising for related guides

architectural guide in 2019, this was not yet possible. Also, I want to postprocess the images, e.g. deleting unnecessary labels.

You may have read in *TUGboat* 42:3 that ConTEXt can process OpenStreetMap data on its own, also via a MetaPost conversion. This is true, but unfortunately not more than a proof of concept. It can't handle labels, like street names, so it's quite useless for a city map. The colors are ugly, too — that would be easy to change, and I promised to provide a theme, but the rendering is just not flexible enough: All paths can only be drawn as single lines, while if you look at other OpenStreetMap renderers, streets usually have a fill and an outline, and for railway tracks you need a thick white line with a dashed black line on it. My programming skills don't suffice to fix that.

So I stick to my proven workflow for the time being. (Figure 4.)

Setup

I wrote the setup for these architectural guides mostly in 2015, and since then, plenty has happened — not only has ConTEXt moved to LuaMetaTEX, but also I've learned a lot and can do a bit better, so I found my old code a bit embarrassing and refactored it, just in time for the upcoming guides that we hope to publish in 2023/24.

I will leave out all the setups with regard to language, fonts and colors.

Simple page layout First we define the page size. That's easy:

```
\setuppapersize[A6]
```

The page layout is quite simple, we have no page header and usually don't need footnotes.

If you set up a layout in ConTEXt, you should always define the parameters `backspace` and `width` first, then `topspace` and `height`. The latter includes

header and footer. You can leave the other areas like margins and edges alone if you don't need them.

Header and footer setting reflect that we don't need page headers and the footer only for page numbers. We need double pages to get the page numbers in the outer footer, otherwise we couldn't distinguish left and right pages.

```
\setuplayout[
  backspace=12.5mm, width=80mm,
  topspace=12.5mm, height=125mm, % text+footer
  header=0mm,footer=10mm,
]
\setuppagenumbering[
  alternative=doublesided,
]
```

Bleed and trim Most of our images cover the full page width, and that means they must *bleed*. 3 mm is a traditional value; in this small format, 1 mm probably would be enough. If our printshop tells me to change it, I want to change it in only one place.

```
\definemeasure[Bleed][3mm]
\definemeasure[Trim][7.5mm]
\setuplayout[
  marking=on, % cut marks
  location=middle,
  bleedoffset=\measure{Bleed},
  trimoffset=-\measure{Trim},
]
```

With regard to printing, we activate cut marks and center the page on the sheet. The trim offset is the difference between sheet and page size as a negative value. The bleed offset is from the page outward as a positive value. It's the same on all sides.

If you would check the outcome so far, you couldn't find these boxes in the PDF. The activation is strangely coupled to some PDF viewer settings:

```
\setupinteractionscreen[
  option={doublesided,bookmark},
    % necessary for Trim/BleedBox:
  width=max,height=max,
]
```

This should work now. But what's the sheet size? We only defined the paper size! Let's fix this:

```
\setuppapersize[A6][A6,oversized]
```

The `oversized` option adds 7.5 mm around the A6 page. We could also define that size explicitly or use the envelope size C6 instead. (Figure 5.)

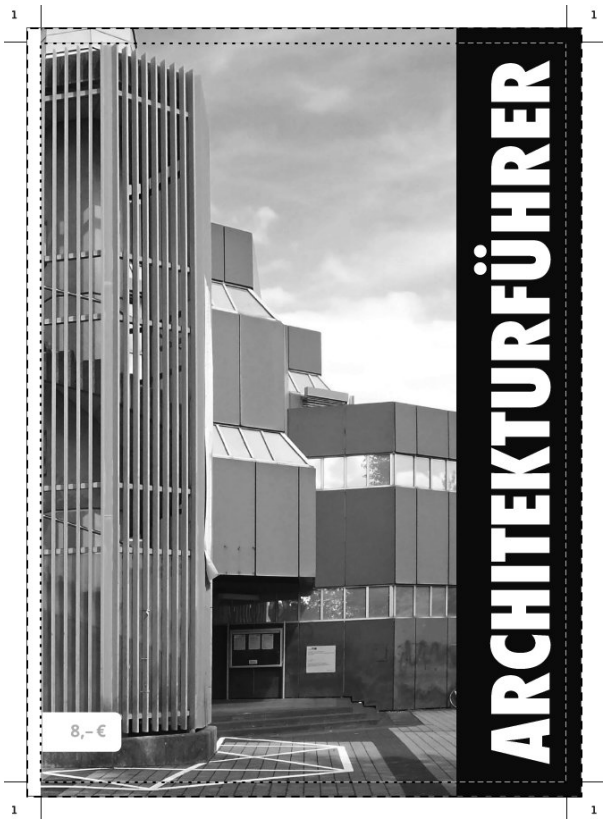


Figure 5: title page with crop marks, trim box (inner) and bleed box (outer)

Preview and print mode While we need bleed, trim and cut marks in the PDF for the printshop, they might confuse the authors in the preview version. They're also not needed for an ebook.

So let's mode-ify the settings. It turns out we only need one mode, 'print'; if activated, it fixes the page size, bleed and trim; it can also turn off interaction (links, etc.).

Another topic where it makes sense to distinguish between preview and print mode is image resolution. It makes no sense to send draft PDFs with high resolution images, and some pictures could use some downsampling even in print mode.

```
% preview (correction copies)
\startnotmode[print]
  \setuppapersize[A6]
  \def\Resolution{96}
  \setupinteraction[state=start]
  \setupexternalfigures[
    conversion=lowres.jpg,
  ]
  % no bleed/trim settings
\stopnotmode

% print version
```

```
\startmode[print]
  \setuppapersize[A6][A6,oversized]
  \def\Resolution{200}
  \setupinteraction[state=stop]
  \setupexternalfigures[
    conversion=hires.jpg,
  ]
  % setuplayout with bleed/trim as above
\stopmode

\setupexternalfigures[
  directory={img},
  resolution={\Resolution},
]
\loadluafile[grph-downsample]
```

This resolution stuff is not (or not yet) a feature of ConTEXt, but handled by some Lua functions that call GraphicsMagick during the TEX run to reduce the image size.

Color conversion to greyscale *is* already included in ConTEXt and works the same way, but here we don't need a greyscale mode.

Image dimensions For our image calculations besides resolution, we need a few basic dimensions.

```
\definemeasure[maxWidth]
  [\paperwidth + \measured{Bleed}]
\definemeasure[maxHeight]
  [\paperheight + 2\measured{Bleed}]
\definemeasure[doubleWidth]
  [2\measured{maxWidth}]
% offsets of images from the type area
\definemeasure[topOffset]
  [\topspace+\headerheight+\measured{Bleed}]
\definemeasure[bottomOffset]
  [\bottomheight+\footerheight+\measured{Bleed}]
```

Where you would use `\newdimen` and `\dimexpr` in ε-TEX, you should use `\definemeasure` in ConTEXt. My companion article "Calculating covers" in this issue (pp. 176–179) explains dimension calculations.

Layers for image placement If you want to place elements in specific locations, the ConTEXt way is to use layers.

For images, it makes sense to use full page layers, but we need to distinguish right and left pages.

```
\definelayer[bgpicleft][
  x=-\measure{Bleed},y=-\measure{Bleed},
  width=\measure{maxWidth},
  height=\measure{maxHeight},
] % incl. bleed
\definelayer[bgpicright][
  x=0mm,y=-\measure{Bleed},
  width=\measure{maxWidth},
  height=\measure{maxHeight},
] % incl. bleed
\setupbackgrounds[leftpage]
  [background=bgpicleft]
\setupbackgrounds[rightpage]
  [background=bgpicright]
```

After definition, we must assign the layers as backgrounds. It's possible to use several layers for one area: `background` takes a list, left to right is top to bottom.

Cover layers For the cover, we need additional layers, and we can already set up the black bar as a text background. (Figure 6.)

```
\definelayer[titlebar][
  x=83mm,y=-\measure{Bleed},
  width=25mm,
  height=\measure{maxHeight},
]
\setupframed[frame=off,offset=overlay]
\setlayerframed[titlebar][
  background=color,
  backgroundcolor=titlebarcolor,
  width=25mm,
  height=\measure{maxHeight},
]{\strut}
```

Image placement

Sorry, I won't show you the implementation of my macros — it's long, convoluted, and ugly.

Full page images The placement command for a full page image looks like this:

Figure 6: #13 HICOG settlements

```
\startpostponing[15]
\pagefig
  [fig:10544-08]% reference
  [rh]% placement code
  {Kurpark, 1950er Jahre}% caption
  {DA01_10544-08}% image file
\stoppostponing
```

"Postponing" moves content to a specific page, the page number can be absolute or relative (+1). Due to expansion and buffering issues it's not possible to include this in a macro.

The \pagefig macro is my own; it takes a reference, a placement code, a caption and the filename of an image. But what does it do?

- decide if we're on a right or left page
- start an empty "makeup" (special layout page)
- place the picture on the layer for the left/right page
- clip the picture to fit (placement code defines if height or width are leading)
- place the caption in the footer (usually white on a dark shadow)
- place debugging information (e.g. filename) in the trim area

The code for a double page image looks nearly the same:

```
\startpostponing[+0]
\doublepagefig
  [fig:11390-29]
  [lh]
  {Blick von Osten}
  {DA01_11390-29}
\stoppostponing
```

This instance was placed between chapters and uses "immediate" postponing (+0).

The macro works similarly to the previous one, except we place the left half of the picture on the layer for the left page and the right half on the right page, each in its own makeup. (A multi-page makeup would confuse the page numbering.) The placement code defines the location of the caption.

Half-page images The call for an image that does not cover a whole page looks like this:

```
\topfig
  [fig:9251]
  [rw]
  {Großer Saal}
  {IMG_9251}
```

I love a consistent interface. But the macro works differently:

- decide if we're on a right or left page
- calculate the actual image dimensions with a Lua function
- decide where to clip (top/bottom) according to placement code
- calculate how much to clip so that the picture fits the line grid
- place it as a float, but move it into the trim area

Why the calculations? I'm using grid setting, even if this is rather questionable with these picture-heavy booklets. But it implies that all images should "sit" on a grid line, i.e. a baseline of body text. ConTEXt couldn't do that on its own. (Only recently, Hans Hagen extended the options for float placement; it might be easier now.) Also, the top border of an image should align with the x-height of a text line, but that doesn't matter in this case.

The image has a fixed width, namely the page width plus bleed. With proportional scaling, we know its maximum height. We subtract the space above the type area (4 values) plus bleed. The remainder modulo the line height is what we need to cut.

It would have also been possible to just move the image, without clipping it.

The simplified float placement looks like this:

```
\startplacefigure[
 location={top,high},
 reference={#1},
 title={#3},
]
\offset[
 topoffset=-\topOffset,
 leftoffset=\measure{leftOffset},
]{%
  \clip[
   x=0mm,y=\topCut,
   width=\measure{maxWidth},
   height=\measure{calculatedImgHeight},
  ]{%
   \externalfigure[#4][width=\measure{maxWidth}]
  }%
}%
\stopplacefigure
```

Shadow captions

The shadow behind captions in full-page images is a MetaPost graphic: A number of stacked rounded rectangles of slightly increasing size, set to a high transparency in "multiply" mode, so that the main shape becomes dark and the borders get blurry.

This graphic is set as an overlay and used as a background to the (invisible) caption frame.

```
\startuniqueMPgraphic{mpshadow}
mw := BodyFontSize/3;
ox := -0.5 ;  % offset x
oy := -0.5 ;  % offset y
bx := 1.5mw ; % bleed x (height of the shadow)
by := 1.5mw ; % bleed y (width of the shadow)
```

Figure 7: multiline caption with a subtle shadow

```
rx := 3mw ;   % max. corner radius x
ry := 2mw ;   % max. corner radius y
steps := 10 ; % number of shadow layers,
              % 10 is a good value
hue := 0.015 ; % 0.02 is a good value
ycorr := 1mw ; % difference between overlay
               % height and shadow height
for step = 1 upto steps:
  part := (step-1)/steps;
  xstep := bx * part ; % current part of bleed
  ystep := by * part ;
  crx := (rx + rx*part)/2; % current radius
  cry := (ry + ry*part)/2;
  % points of the rounded rectangle
  xa := -xstep + ox;
  xb := -xstep + ox + crx;
  xc :=  xstep + ox - crx + \overlaywidth;
  xd :=  xstep + ox       + \overlaywidth;
  ya := -ystep + ycorr + oy;
  yb := -ystep + ycorr + oy + cry;
  yc :=  ystep - ycorr + oy - cry
                           + \overlayheight;
  yd :=  ystep - ycorr + oy + \overlayheight;

  fill (xb, ya)---(xc, ya)...(xd, yb)---
    (xd, yc)...(xc, yd)---(xb, yd)...
    (xa, yc)---(xa, yb)...cycle
    withcolor transparent(1, hue, black) ;
endfor;
setbounds currentpicture to OverlayBox ;
\stopuniqueMPgraphic
\defineoverlay[shadow][\useMPgraphic{mpshadow}]
% ...
\inframed[frame=off,
  background=shadow,
  foregroundcolor=white,
]{This is my caption.}
```

This was first made for a shadow behind images, and it works well for text when there's only one line or if you can make all lines the same width. The example is one of the few where that wasn't possible, but I was never satisfied with this solution.

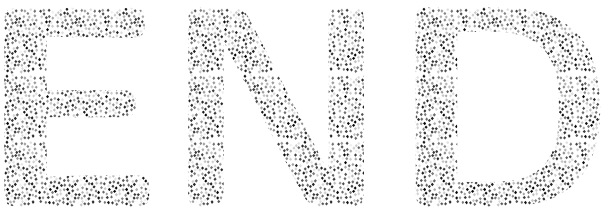
The outline approach Just recently I found out how to make a shadow that adapts to the font shape. This uses a LuaMetaFun extension for font outlines. Again, we stack elements with a low opacity, this time with an increasing outline "rulethickness".

The shadow color is somewhat irregular due to overlapping outlines between letters or letter elements. Maybe it's possible to combine the paths.

```
\definecolor[tshade][t=.05,a=1,k=1]
\starttexdefinition ShadowText #1
\startMPcode
steps := 10 ; % number of shadow layers
rulesize := BodyFontSize/steps/3;
for step = 1 upto steps:
  draw lmt_outline [
    text = "\vbox{\strut #1}",
    kind = "fillup",
    fillcolor = "tshade",
    rulethickness = (step*rulesize),
  ];
endfor;
% finally, opaque white text on top
draw lmt_outline [
  text = "\vbox{\strut #1}",
  kind = "fillup",
  fillcolor = "white",
  rulethickness = 0,
];
\stopMPcode
\stoptexdefinition
% ...
\ShadowText{Now, doesn't this ...}
```

If you use this with big text, it makes sense to add `randomized 3` to the `lmt_outline` call to make it look a bit more natural.

The LuaMetaFun `lmt` functions were introduced in 2021 and are quite fun to play with. E.g. to fill an `lmt_outline` path with an `lmt_poisson` pattern:



◇ Henning Hraban Ramm
Limburg, Germany
hraban (at) fiee dot net