```
1438 -- 6
1439 local blur_radius,blur_sigma = 0, 2
1440 local frame_size = blur_sigma+1
1441 local optimize_shadow
1442      format("+clone            lor
1443 -channel Alpha
1444
1445
1446 -- IM SHADOW W
1447 --  1. create pe
1448 --  2. create pen
1449 --  3. create umbra
1450 --  4. create umbra
1451 --  5. combine transpa
1452 --  6. adds full transp
1453 --     corrects the bord
1454 --     and also blurs poss bly visible transiti
1455 --  7. save shadow
1456
1457 batch[#batch+1] = IM_command .. cl ..
1458      lp .. penumbra_mask
1459      lp .. "+clone
```

drops

The shady side of ConTEXt

(August 24, 2017)

Contents

dropshadow

»drops« is a small extension for ConTEXt, that allows you to add drop shadows to a wide range of shapes. This includes simple squared areas (so called *boxshadows*) as well as more complex objects. All you need is a working installation of ImageMagick (IM), which is required to create the shadow graphics. The supported color spaces are CMYK, RGB and Grayscale, the file formats are limited to `PNG` and `JPG`.

As mentioned, the module is based on the shadow[1] functionality of ImageMagick. The default IM shadow has only two main arguments[2], *opacity* and *sigma*. The first one sets the opacity (0=transparent, 100=opaque) of the shadow, the second one defines the size of the transition area (given opacity to zero opacity).
The transition itself is of linear type, which sadly limits its direct usability for realistic looking shadows. You have no further influence on it and a post-processing of the shadow for a non-linear transition is difficult and leads to other problems.

After several tests I favoured the usage of a combined shadow with two parameter independent shadow areas. One for the umbra area and one for the penumbra area. That's the main idea behind it (and the key to understand the parameters).
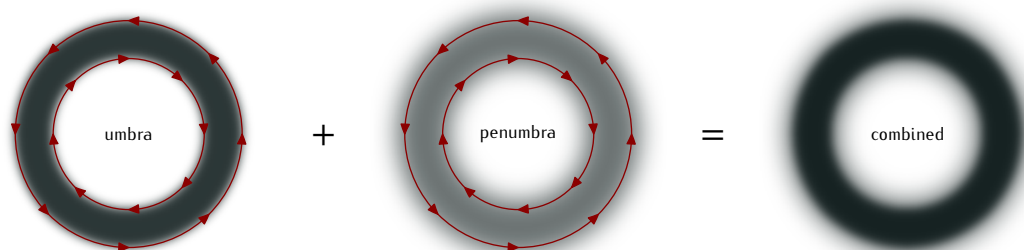
Figure 1   Combining umbra and penumbra shadow

The graphics in figure **1** show the single shadow areas and their combination. The red arrows show the paths of the invisible shadow source, a torus.

Also: not everything is expected to be working right now, expansion problems may occur. This module needs testing. Suggestions for improvements and bug reports[3] are welcome.

---

[1] see **http://www.imagemagick.org/Usage/blur/#shadow** for the details
[2] the *offset* is ignored here, because the shadow positioning is done by ConTEXt
[3] including reports about typos or crimes against the English language in this manual

1 Installation

1. If you are using ConTeXt standalone, you can install the module by using

   `first-setup.sh --modules="t-drops"`

   For a manual installation unzip the archive to '$TEXMFHOME' or '$TEXMFLOCAL' and regenerate TeX's database by calling 'context --generate'.

   To verify the installation check if 'texmfstart --locate t-drops.mkiv' returns a meaningful path.
   The execution of external programs (**write18**[4] feature) must also be allowed to make this module work properly. Without it, the batch file for the creation of the graphics can not be executed automatically. You can still copy the created batch file into your working directory (it has to be called from there) and execute it manually.

2. A recent version[5] of **ImageMagick**[6] (IM) is needed for this module.

   You can use the environment variable '$IMCONV' to point to a local installation. The variable should contain the complete path of the IM binary.

   — **Windows**
   The installer packet should set the '$PATH' variable automatically, so that the IM commands are found. In case of the non installer packet you have to add the IM binaries path manually.

   Pitfalls: the »convert.exe« problem (IM6 only)

   There is an equal named program on Windows OS, that converts the file system of a given partition from `fat32` to `ntfs`. If this program is called instead of the IM binary, you have to edit the '$PATH' variable. The entry for IM must stand before the 'c:\windows*' entries.
   I highly recommend »Redmond Path[7]« for this job on older Windows versions (< 10). Editing this long environment variable by hand is both unclear and cumbersome.
   On newer Windows 10 this is no longer a problem, but you have to finish the IM path with a backslash.

---

[4] `http://wiki.contextgarden.net/write18`
[5] 6.9.9–6 – 7.0.6–9 should work, 6.7.6–0 or lower will fail
[6] `http://www.imagemagick.org` ; search for your OS under 'Binary Releases'
[7] `http://www.softpedia.com/get/System/System-Miscellaneous/Redmond-Path.shtml`

As last resort you can use the environment variable '`$IMCONV`'. Just specify the complete command path to your IM convert command in it.

Beware: older Windows versions than Vista need a complete restart of the OS to make the change happen!

— **Unix/Linux**
Check if the ImageMagick (IM) packet is installed (type '`identify rose:`' into a shell). Install it, if the test fails.

If your IM version is too old, you can build the actual version from source. A detailed description can be found **here**[8]. I'll only give you some extra information here, which may save you some time.
The main obstactle on our way to a fully featured IM version are the (probably missing) *library headers*, that are needed for the different graphic formats, compression algorithmns and other stuff. You have to install several '`lib*-dev`' packets to get the same functionality, as in the binary releases of ImageMagick.

NAME	delegate
autotrace	Autotrace
bz2	BZLIB
cairo2	--- (requirement for svg2)
freetype6	FreeType
jasper	JPEG-2000
perl	PERL (admin rights needed)
png12	PNG
svg2	RSVG
tiff5	TIFF, JBIG, JPG, ZLIB
webp	WEBP
wmf	WMF

Figure 2   delegates and their library dependencies

Here is a list of all the packets and their corresponding 'delegates', that I used on »Linux Mint[9]«. The naming scheme of these packets is always of the form '`libNAME-dev`', where '`NAME`' is the only variing part (e.g. for NAME=autotrace

---

[8] `http://www.imagemagick.org/script/advanced-unix-installation.php`
[9] `http://www.linuxmint.com/`

the packet name is '`libautotrace-dev`'). Some of the packets have a version number at the end of their name (which can vary). Skip the number when you search for a specific packet in the packet manager.

Example: search for '`libfreetype`' or '`libfreetype*-dev`';
The summary at the end of the '`configure`' output tells you about the delegate setting and its value. Some of the *delegates are disabled by default* and have to be enabled by hand. In my case with the call

```
./configure --with-autotrace=yes --with-modules=yes \
--with-perl=yes --with-rsvg=yes --with-wmf=yes \
--with-webp=yes --with-xml=yes
```

If the *default installation path* of your Linux distribution differs from '`/usr/lo-cal`', you have to set the correct path manually. For »Linux Mint« this is just '`/usr`'. Look at the files list in your packet manager to get the default path (e.g. at your outdated ImageMagick packet).
Adding '`--prefix=/../usr`' to the prior '`configure`' command will set the correct installation path in my case.

It is also possible to install a *local version* of ImageMagick in case you have no admin rights. I used the local path '`--prefix=/../home/indiego/im agemagick7`' to install the alpha 7.0 version of ImageMagick. The delegate '`PERL`' needs admin rights, so it was deactivated beforehand. This local installation can then be used in combination with the environment variable '`$IMCONV`'.

Configuring should be clear now and there was no need on my distribution to set up any compiler options. Nothing to tell here.

Now, before you install the new version, you should use your packet manager to uninstall any existing version of ImageMagick. This step is not needed in case of a local installation.
A final '`sudo make install`' should do the rest. Then type '`identify -list configure`' in a terminal, and check, if everything is correct. To uninstall this version again, type '`sudo make uninstall`' from the same folder.

And that's it.

2 Usage

Simply add

\usemodule[drops]
\setupcolors[pagecolormodel**=**auto] **%** gray rgb cmyk auto none

to the top area of your document. Setting the page color model is very important, because we have to tell the PDF viewer to use the matching color space, when it calculates the transparent graphics with the underlying content. Without a proper setting the default transparency color space (CMYK) is used and this is probably not what you want for your transparent RGB graphics. Choose 'auto' if you are unsure about the correct setting (this document also uses the 'auto' option).

\setupdrops[<drops parameters>]

```
\setupdrops [..,..=..,..]
                          *
*   background             =   screen color none foreground NAME
    backgroundcolor        =   NAME
    direction              =   NUMBER
    fileformat             =   jpg png
    mppath                 =   NAME
    offset                 =   DIMENSION
    pdistance              =   DIMENSION
    penumbra               =   NUMBER
    psigma                 =   DIMENSION
    radius                 =   DIMENSION
    resolution             =   NUMBER
    rotation               =   NUMBER
    shadowbackgroundcolor  =   NAME
    shadowcolor            =   NAME
    udistance              =   DIMENSION
    umbra                  =   NUMBER
    usigma                 =   DIMENSION
```

The \setupdrops command defines the global defaults for the \drops macro. Calling it without any argument resets all parameters to their initial value.

\drops[<drops parameters>][<framed parameters>]{content}

\drops *[..,..=..,..] [..,..=..,..] {...}*
                    OPT   OPT
1   inherits: \setupdropssetup = the name(s) of one (or more) predefined
             setup(s)
2   inherits: \setupframed
3   CONTENT

to be used in setup definitions only (see t–drops–setups.mkiv)

\LocalStyle{<list of 'key**=**value' pairs**>**}**\crlf**

control and debugging

\dropson \dropsoff % enables/disables module
\dropsdo % always create new graphics if needed; the default
\dropsdone % never create new graphics, but use available ones

handy for preset developement and debugging
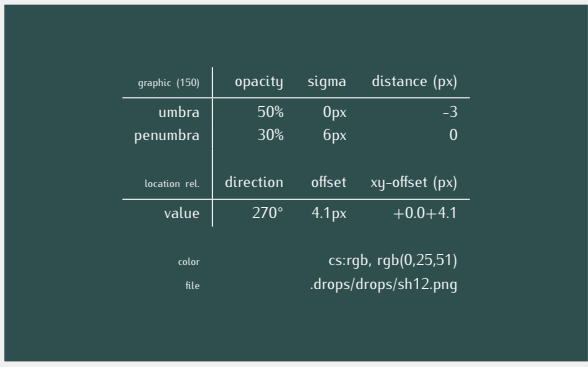
% shows bounding box of object and shadow as wire frame
\dropsshowframes
% shows a table with the most important parameters
\drops[setup**=**drops:blackboard][width**=**80mm,height**=**50mm,
  foregroudcolor**=**white]{**\dropsshowtable**}
\dropshideframes

| graphic (150)	opacity	sigma	distance (px)
umbra	50%	0px	–3
penumbra	30%	6px	0
location rel.	direction	offset	xy-offset (px)
value	270°	4.1px	+0.0+4.1
color		cs:rgb, rgb(0,25,51)	
file		.drops/drops/sh12.png	

3 Examples

Let's start with the setting of a common shadow `direction` and `shadowcolor` for the following examples.

```
\setupdrops[direction=270,shadowcolor={azure:V20}] % shadow at bottom
```

3.1 boxshadow examples

Here is a well known example that makes use of the default 'boxshadow' setting. More about it in the overlay examples on the following pages.

```
\drops[radius=2.54mm]
{\framed[offset=2.5mm,strut=no,
 frame=off]{\externalfigure[cow]}}
```
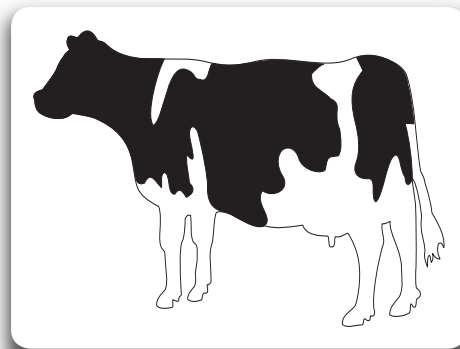
The next example shows the usage of a predefined setup, that was optimized for a 'flat' look (photo on paper). The additional white frame is neeeded to get a reliable contrast.

```
\defineframed[photoframed]
  [offset=2.3mm,depth=0mm,strut=no,
   frame=off]


\drops[setup={drops:flat:dark},
  udistance=-0.34mm,pdistance=0mm,
  offset=0.87mm]
  {\photoframed{\externalfigure[mill]}}
```

Beware: the settings from a `setup` always override other settings (parameter order is ignored)!

Boxshadows can also be used to enhance the readabilty of (smaller) text elements on any background. This technique is not recommended for big head lines though.

```
\startuniqueMPgraphic{blacktowhite}
    linear_shade(OverlayBox,0,white,black);
\stopuniqueMPgraphic
\defineoverlay[blacktowhite][\uniqueMPgraphic{blacktowhite}]

\startsetups drops:readability
  \LocalStyle{direction=180,offset=0.3mm,background=,radius=4mm,
    umbra=50,penumbra=45,udistance=0.7mm,pdistance=1.2mm}%
\stopsetups

\def\JustListening{~Nadine Shah – Love Your Dum and Mad~}%
\framed[offset=8mm,background=blacktowhite]
{\vbox{%
    \hbox{\JustListening}
    \blank[medium]
    \drops[shadowcolor=white,setup={drops:readability}][]
    {\hbox{\JustListening}}
    \blank[big]
    \hbox{\startcolor[white]\JustListening\stopcolor}
    \blank[medium]
    \drops[shadowcolor=black,setup={drops:readability}]
    [foregroudcolor=white] % framed part
    {\hbox{\JustListening}}
}}
```
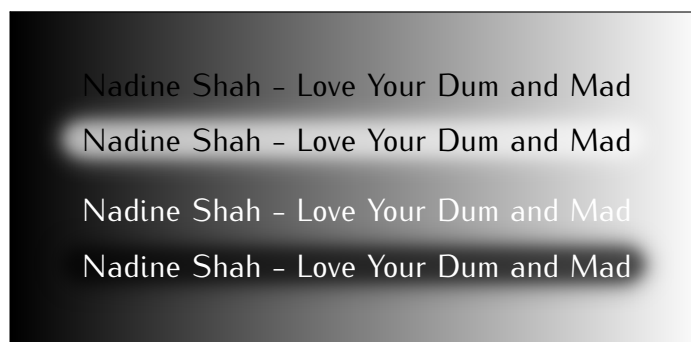

Figure 3   It's best when the text is readable and when the difference in brightness doesn't catch your eye. The latter can't be accomplished in this extreme example.

Using **overlays** is also possible. The module uses the default background value 'color' to support the easy creation of the standard boxshadows.
The `backgroundcolor` parameter defines the color for that kind of box.

```
\defineoverlay[myboxshadow] % enclose \drops in curly braces
[{\drops[backgroundcolor=snow2,radius=2.54mm]
 [width=\overlaywidth,height=\overlayheight,empty=yes]{}}]

\defineframed[myboxshadow]
[background=myboxshadow,% overlay as background
 framecolor=white,rulethickness=2.5pt,offset=2.5mm,
 radius=2.1mm,corner=round,strut=no]
```

```
\myboxshadow{\externalfigure[cow]}
```
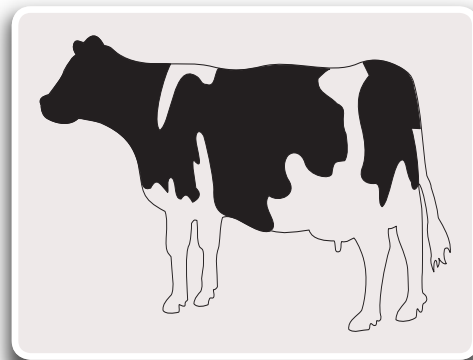
Using overlays means, that we are using the `background` mechanism of the \framed macro. If you use overlays on (bitmap) graphics, setting a suitable 'offset' and 'strut=no' is very important. The default settings of these parameters are optimized for text based content and not for graphics.

Without a given offset a small amount is automatically added to each side of the overlay box. Your graphic and its shadow will grow in size.
The 'strut=no' setting is even more important here. Without it the dimension \strutdepth is added to the total height of the overlay box, resulting in a noticable size addon at the bottom. But the crux is, that this value is directly derived from the current font size. Any change of the active font (size) can also change the size of all such overlay boxes. The consequences should be clear.

The next example shows how you can ged rid of that default background and use your own (well, here also 'color'). The background can be set in the \drops related parameter part (1) or in the \framed parameter part (2) of the command. The latter overwrites all other settings.

```
\defineoverlay[shadowonly]
  [{\drops[][width=\overlaywidth,height=\overlayheight,
    background=,empty=yes]{}}]

\hbox\bgroup
  \framed[background=shadowonly,offset=2.5mm,strut=no]
    {\externalfigure[cow]}
  \hskip12mm
  \framed[background={shadowonly,color},backgroundcolor=snow2,
    offset=2.5mm,strut=no]
    {\externalfigure[cow]}
\egroup
```
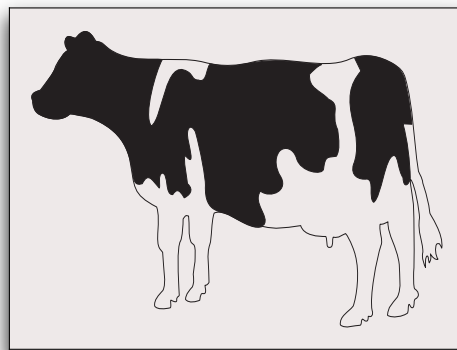
3.2 mppath examples

The module also supports arbitrary paths (currently with some limitations).
In the first step we define the desired path in a MetaPost graphic and pass it via
`passvariable` to a specific `<path_id>`, here 'ivy'.

```
\startuseMPgraphic{drops:ivy}
    path p;
    p:= (176.86,1.21)
        ..controls (166.41,16.38) and (164.91,36.27)..(152.79,50.54)
        % ...
        ..controls (183.13,8.3) and (181.7,-3.82)
        ..cycle;

    w:= xpart urcorner boundingbox p;
    f:= 60mm/w;
    p:= p scaled f;

    draw p withpen pencircle scaled 2 withcolor white;
    fill p withcolor \MPcolor{gray85};
    label("\CONTEXT",(xpart center(p),1.22*ypart(center(p))))
        withcolor \MPcolor{gray40};

    passvariable("ivy",p);
    setbounds currentpicture to boundingbox p;
\stopuseMPgraphic
```

After the graphic definition the `<path_id>` is still unknown to ConTEXt, because the
graphic hasn't been used yet. We simply put the graphic into a predefined `\box`,
where it can not interfere with other stuff. All but the path is ignored this way. After
that we can use the `<path_id>` as value for the parameter `mppath`.
To avoid the unwanted color background from the default 'boxshadow' mechanism we
also have to clear the `background` parameter.

```
\setbox\Pathbox\hbox{\useMPgraphic{drops:ivy}}%
\drops[mppath=ivy,background=,offset=1mm,resolution=300,
       umbra=80,penumbra=70,usigma=0.34mm,psigma=1.5mm]
 {\useMPgraphic[drops:ivy]}%
```

And this is how the result looks like.

Figure 4   An ivy leaf

You can also use graphics with multiple paths (all must be closed though), just use `passarrayvariable` in this case. A typical example code would look like this …

```
path p[]; % an array of paths
p0:= fullsquare xyscaled(w,h); % unused sub path
p1:= fullcircle xyscaled(w,h); % outer paths: counter clockwise
p2:= reverse(p1 scaled 0.66); % inner paths: clockwise

% passarrayvariable(<path_id>,<path_array>,<from>,<to>,<increment>);
% use sub paths from p1 to p2, loop increment is 1
passarrayvariable("torus",p,1,2,1);
setbounds currentpicture to boundingbox p1;
```

Using **overlays** in combination with `mppath` is a but more tricky. The `<path_id>` of the MP graphic has to be unique for every different overlay path. You can handle a few different paths by simply predefining them, but if their number exceeds, the path id has to be created dynamically at runtime.

To make it worse, this must be done twice. Once in MetaPost, when we pass the path to LuaTEX, and the second time when we use that id in the overlay definition. The best and cleanest way is to define only one instance (a lua function), that handles both cases. Just look into the 'overlay' example for the details (and also for the hiding devils).
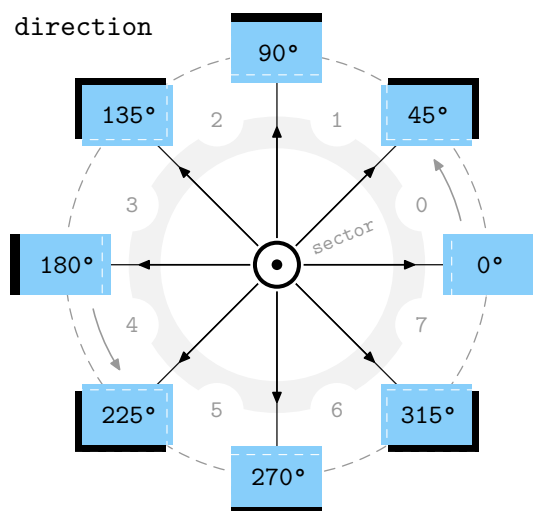
4 Parameters

The »drops« parameters are divided into two groups: graphic dependent and graphic independent. All graphics dependent parameters directly influence the appearance of the shadow graphic. Any change of a graphic dependent parameter may force the recreation of that graphic.

4.1 graphic independent

Unlike the default ImageMagick shadows »drops« shadows are not »positioned« (by an amount of blank pixels for the needed shift). All positioning is done by ConTEXt instead, so you can use the same graphic for all positions. Playing around with different values for `direction` and `offset` doesn't require the generation of new graphics. This way you are also not bound to whole numbers (pixel) for the shadow offset(s), you can place it wherever you want.

direction   The `direction` defines where (in relation to the source box) the shadow is placed. Just enter the wanted degree as positive or negative number (but without '°'). Any number will do, you are not limited to the (extremal) values shown in the graphic. The default value is '–45' (315°).

Figure 5   shadow direction and resulting position

offset   The final x/y-offset of the shadow is automatically calculated from the parameters `direction` and `offset`.

Believe it or not, but this is (one of) the most important parameters. It indirectly defines the distance between the shadow source and

the background (e.g. the paper) and also the relative angle of the light to the object. In combination with the `direction` parameter and the shadow's appearance (luminosity, color) it sets the ambiance of the whole scene.

An example to make this clear. Let's say you want to add a drop shadow to a photo lying flat on the paper. The distance in this case is very short, so unless you have an extreme light position, the shadow offset is also small. If you use a bigger offset here, your photo is automatically hovering over the paper.

`rotation` This parameter is only needed, if the object is rotated and you want the shadow to stay at the given direction. The `rotation` value is simply subtracted from the `direction` value to compensate the rotation. Be sure that this value is set to zero, if you do not rotate the object.

`background,` These two are the known `\framed` parameters. Very handy to `background-` create boxshadows with text content.
`color` defaults: `background=color`, `backgroundcolor=white`

4.2 graphic dependent

Let me say it again. Any change of a graphic dependent parameter can (and probably will) force a recreation of the graphic!
Apart from that, there is one more important thing to say. As ImageMagick deals with pixels, all graphic dependent dimension parameters are rounded to the next full pixel number. This can lead to the situation, that you change a dimension, but no new graphic is created, because the pixel number hasn't changed. Keep that in mind when you change the dimensions.

px	mm	px	mm	px	mm	px	mm	px	mm
1	0.17	6	1.02	11	1.86	16	2.71	21	3.56
2	0.34	7	1.19	12	2.03	17	2.88	22	3.73
3	0.51	8	1.35	13	2.20	18	3.05	23	3.89
4	0.68	9	1.52	14	2.37	19	3.22	24	4.06
5	0.85	10	1.69	15	2.54	20	3.39	25	4.23

Table 1   conversion table for the default resolution (150ppi)

It is very handy to use 'px' based dimensions while playing around with the parameters. No extra conversion is needed and any change will have an effect on the graphic. But be aware, that the size of a pixel is not a fixed dimension. Any `resolution` change will also change the final size of a pixel based dimension. It's best to convert them to a fixed dimension (like 'mm') after you have found the right settings.

`penumbra,` `umbra`	Defines the opacity value of the shadow area. The final shadow graphic is a combination of both shadow areas: the darker `umbra` area and the lighter `penumbra` area. When the shadows are combined, the maximal opacity value (0=transparent, 100=opaque) of every pixel position is choosen for the final graphic. As we work with two parameter independent shadows here, each area has their own `*sigma` and `*distance` parameter. All umbra related parameters start with the letter 'u', the penumbra related parameters with the letter 'p'.
`psigma,` `usigma`	The `*sigma` values define how much space is used to blend over from the given shadow opacity (`umbra,penumbra`) to zero opacity (full transparency). The size of the graphic is increased by $2 \times$ sigma on each side (the space is needed for a blur operation). No negative values are allowed here. Even a sigma value of zero results in a shadow, that has a small transition area and is slightly bigger than the source object. A higher sigma value always leads to a wider shadow, but you can compensate that by using a negative `*distance` parameter. This parameter directly influence the size of the shadow mask (the source for the shadow creation).
`pdistance,` `udistance`	The `*distance` parameters define the offset of the shadow mask paths (umbra, penumbra) from the given original path (`mppath`). These parameters are only needed, if you want to vary the size of a sub shadow. For positive distances the new path is drawn outside (bigger) of the original path, for negative ones it is drawn inside (smaller). If things go the other way round your path does not follow the rule, that outline paths are drawn counter-clockwise, inline paths clockwise. Just 'reverse(<path>);' the MP path in this case.

The evaluated path is only an approximation and its calculation can fail, especially if the new path intersects somewhere. The more you advance the distance, the higher is the risk to run into problems. This highly depends on the shape of the path.

The penumbra related parameter must be at least as big as the umbra related parameter. The reason for this is, that the umbra shadow is directly shifted and drawn on the (bigger) penumbra canvas.

`pdistance` $\geq$ `udistance`

Also: the sum of the `pdistance` and `psigma` parameters should be greater than the sum of `udistance` and `usigma`. Otherwise you won't get a noticeable penumbra area.

`psigma` + `pdistance` > `usigma` + `udistance`

mppath   This parameter is empty by default. In the default case the build in boxshadow mechanism automatically creates a path for you. This path is always a simple square (with optional rounded edges).

If you want to add a shadow to a different object, you have to define the corresponding `mppath` first. All you have to do for this is to pass a MetaPost path to LuaTEX by calling the MetaPost commands 'passvariable' or 'passarrayvariable'. The first one is for single paths only, the latter for multiple paths.

```
passvariable(<path_id>,<path>);
passarrayvariable(<path_id>,<path_array>,
            <from>,<to>,<increment>);
```

The parameters `<from>`,`<to>` define the start and end index of the used sub paths. The `<increment>` parameter defines the increment value for the sub path loop (normally '1'). The lack of any of these parameters will result in an endless loop.

On the first look this seems trivilar, but the devil is in the details. Some prerequirements must be fullfilled, before we can successfully use the '`<path_id>`' parameter as value for the `mppath` parameter.

- path related: the path has to be closed (start and end point are the same) and intersection free. The path should also fulfill the Convex Hull Property (CHP), meaning that the curve is compleately contained in the convex hull of its control points.
- The final MP graphic is not empty, meaning its bounding box is not zero (empty graphics are simply ignored by ConTEXt). A correct bounding box is a prerequirement for overlays.
- You have to call the path graphic before you can use that path.

See chapter 3.2 for a `mppath` example.

resolution   This parameter sets the resolution/density (in pixels per inch) for the generated shadow graphics. The default value of 150ppi should be suffient for most exercises, as it is a good compromise between quality and size.

Figure 6   Doubling the resolution means to quadrupli–cate the pixel quantity. The size of the graphic stays the same.
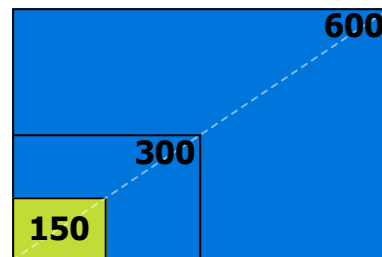
Boxshadows normally don't gain that much from higher resolution values. Using a lower value doesn't save much size, but decreases the quality significantly. Using a higher value increases the file size and the creation time of the graphic, but also increases the shadow quality. Keep in mind that it also takes more time to render a (transparent) high resolution graphic in a viewer.

Higher resolution values are recommended for the usage of user defined shadow paths (`mppath`). A resolution value of 300 or 600 should be sufficient for higher quality requirements.

radius   This is a 'boxshadow' only parameter and its value is ignored, if the parameter `mppath` is set.
Sets the radius for the corners, any positive dimension is valid. If the source box has round edges, the shadow should have them too. The default value is '0mm'.

shadowcolor Sets the color of the shadow. The default shadow color is 'black'. This parameter defines the color temperatur of the light and has a direct influence on the picture's atmosphere.

Natural light for example has a higher blue proportion (azure). By lowering[10] the brightness ('value' in HSV color space) of the 'azure' standard we get a more natural, but also colder looking shadow color. Using a more reddish shadow color gives a softer and warmer looking sight.

```
\definecolor[azure]     [h=007FFF] % hsv(210,100,50)
\definecolor[azure:V20] [h=001933] % hsv(210,100,20)
```

shadowback-groundcolor Mainly needed for JPG. As JPG doesn't support transparency, the shadow graphic has to be 'flattened' before saving. All transparent pixels are calculated with the given color to get the final, non transparent graphic.

The color is also used as background color (bKGD) in PNG. This may help some graphic viewing programs to set the correct background color, when they display the graphic or generate thumbnails.

The default value for the `shadowbackgroundcolor` is 'white'.

fileformat The supported formats are 'png' and 'jpg'. The latter should only be used, if you have to produce CMYK only documents or if the usage of transparency is forbidden.

The drawbacks of JPG with these kind of graphics are compression artefacts even with higher quality settings and a bigger size when compared to PNG. As there is no tranparency, an overlapping of shadow graphics is also not possible.

default: 'png'

The usage of `setup` can be graphic dependent, it's simply a matter of the used parameters.

setup If the name of a predefined setup is given, this setup will be executed after(!) the normal parameter setting. The consequence is, that any local style parameter in that setup will overwrite prior settings.

---

[10] use **http://www.colorizer.org** for your color experiments

A setup can change every parameter (except `setup` itself), multiple setups can be called at once.

Example: `setup={drops:soft:dark,drops:b}`

You can and should make your own setups, as there is no good working general preset for all cases.

4.3 The `\framed` part

Only a very limited number of settings really make sense here. It was also never intended to support all parameters here. It's more a cheap way (`\framed` is used internally anyway) to get the needed functionality without adding additional parameters. If you don't use overlays you will probably never be in the need to set any parameter here.
On the first look the `offset` parameter collides with its namesake in the `\drops` parameter list. Same name, but different meaning. But the internally used `\framed` 'offset' is always set to 'overlay', so there was no real need for a different name. Just don't use `offset` here!

Current usage: size setting (main focus on overlay definition), setting the `foregroundcolor` for text content

5 Directory structure and files

All »drops« related files are stored in the '.drops' directory of the current path. The graphic files are located in the sub directory '.drops/<tex.jobname>/', the hash file and the ImageMagick batch file are stored in the '.drops' main directory.

```
.drops/                          ; drops main directory
.drops/drops-<tex.jobname>[.bat] ; IM batch file
.drops/drops-<tex.jobname>.lua   ; drops hash file
.drops/path-<pathid>.mvg         ; IM path (external)
.drops/<tex.jobname>/            ; home of all graphic files
```

If a user shadow path exceeds a given maximum length the path is stored as an external file. This way we can get rid of the length limit for command lines (2K on Windows 2000/NT4.0). Keep in mind that two paths are used per IM call.

The book keeping of the created shadow graphics is limited. The IM batch file is called at the end of every run, if a new graphic was added or if a graphic dependent parameter of an old one has changed in that run. There is no automatic cleanup of unused graphics.

Over time those 'zombie' graphics can accumulate in the related sub directory (tex.jobname). The number of such graphics is expected to be low and has also no influence on the resulting PDF. They only cost time (once) when created and some storage space.

Beware: if you change a parameter that is used for the majority of your graphics, like for example `fileformat` or the `shadowcolor`, you raise a horde of zombies. Use your favorite tool and deal with them!

For a manual cleanup you can simply delete all bitmap graphics and the hash file. Don't forget to also delete your `<tex.jobname>.tuc` file, as you will need an extra run to recreate the graphics. In the next ConTeXt run all active graphics are then automatically recreated.

6 Some notes and facts

The shadow graphics don't occupy space, the size of the source object is unchanged. This is a feature (photons do not count) and not a bug. Use the spacing parameters of the object's macro if possible.

I suggest that you use a small test file to experiment with the shadow parameters. The creation of the graphics is time consuming and there is no need to do this with your complete document with all shadow graphics over and over again. The code in `<whatever>/doc/context/third/drops/presets.tex` can be used as starting point. It was used to develope the predefined setups. Anyhow, it's your time, so do as you like.

Use the same lighting conditions to prevent irritations.
Try to create realistic shadow scenes (use your perception of real shadows as reference).

For extra debugging info add '`--trackers=modules.drops`' to your command line.

7 Known bugs and limitations

- `\drops` is not fully expandable;
  Multi page objects, like multi page tables, won't work at all (compilation breaks)

- There are several incompatibilities to older IM versions and I will not support them for various reasons. There was a change in the info string format of the IM6 version lately (to standardize it with IM7), which breaks compatiblity with older drops releases.
  If your IM version is at least 6.9.9-6 things should work as expected.

- the `offset` parameter does not support pixel based dimensions (string.todimen limitation)

- The offset algorithm for the `*distance` parameter doesn't handle paths with spicky angles in a proper way. Currently the offset path segments are connected by extending both lines and by choosing their intersection point. The distance of this point can be a few times bigger than the desired offset this way.