pxpic

draw pixel pictures

Jonathan P. Spratte*

2021-12-12 v1.3

Abstract

With pxpic you draw pictures pixel by pixel. It was inspired by a lovely post by Paulo Cereda, among other things (most notably a beautiful duck) showcasing the use of characters from the Mario video games by Nintendo in LaTeX.

Contents

*jspratte@yahoo.de

1 Documentation

1.1 Drawing pictures

pxpic supports different input modes, all of them have the same basic parsing behaviour. A ⟨*pixel list*⟩ contains the pixel colours. The image is built line wise from top left to bottom right. Each row of pixels should be a single TEX argument (so either just one token, or a group delimited by {}), and within each line each pixel in turn should be a single TEX argument (so either just one token, or a group delimited by {}). Spaces and hence single newlines in the sources between ⟨*pixel list*⟩ elements are ignored. The different modes are explained in subsubsection 1.2.2. The only disallowed token in the ⟨*pixel list*⟩ is the control sequence \pxpic@end (plus the usual restrictions of TEX so no unbalanced braces, no macros defined as \outer).

There is a small caveat however: pxpic draws each pixel individually, and there is really no space between them, however some PDF viewers fail to display such adjacent lines correctly and leave small gaps (basically the same issue which packages like colortbl suffer from as well). In print this shouldn't be an issue, but some rasterisation algorithms employed by viewers and conversion tools have this deficit.

Another thing I should mention: The pictures you can draw with pxpic can't be arbitrary large. Due to the design decision of the output as a single \hbox and the way the output routine works, pictures are limited by TEX's memory size to roughly 440×440 pixels in pdfLATEX with the default settings in TEX Live. The size is unlimited in LuaLATEX, due to dynamic memory allocation. In XELATEX the size should be even smaller than in pdfLATEX.

\pxpic `\pxpic[⟨options⟩]{⟨pixel list⟩}`

⟨*options*⟩ might be any options as listed in subsection 1.2, and ⟨*pixel list*⟩ is a list of pixels as described above. \pxpic parses the ⟨*pixel list*⟩ and draws the corresponding picture. The result is contained in an \hbox and can be used wherever TEX expects an \hbox. As a result, when you're in vertical mode a \pxpic will form a text line, to prevent this you can use \leavevmode before it. The \pxpic will be bottom aligned by default (see the options b, c, and t), you can further tweak this using \raisebox (or, if you want, TEX's \raise and \lower primitives).

1.1.1 Examples

Since the above explanation of the ⟨*pixel list*⟩ syntax might've been a bit cryptic, and a good documentation should contain examples (this doesn't claim this documentation is *good*), well, here are some examples (you might need to take a look at subsection 1.2 and subsubsection 1.2.2 to fully understand the examples). Examples in this section will use the following \pxpicsetup:

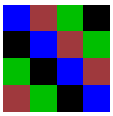
```
\pxpicsetup
  {
     mode    = px
    ,colours = {k=black, r=[HTML]{9F393D}, g=green!75!black, b=[rgb]{0,0,1}}
    ,skip    = .
    ,size    = 10pt
  }
```

We can draw a small cross rather easily:

```
\pxpic
  {
    {.k}
    {kkk}
    {.k}
  }
```

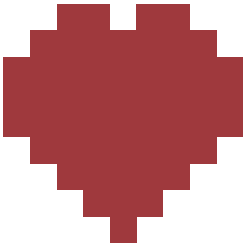
A small multicoloured grid:

```
\pxpic
  {
    {brgk}
    {kbrg}
    {gkbr}
    {rgkb}
  }
```



A heart (shamelessly copied example from PixelArt):

```
\pxpic
  {
    {..rr.rr}
    {.rrrrrrr}
    {rrrrrrrrr}
    {rrrrrrrrr}
    {rrrrrrrrr}
    {.rrrrrrr}
    {..rrrrr}
    {...rrr}
    {....r}
  }
```

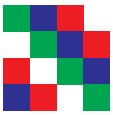


Using `mode=rgb` to draw a short coloured line:

```
\pxpic[mode=rgb]{{{1,0,1}{1,1,0}{0,1,1}}}
```

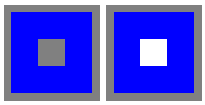
A multicoloured grid using skips and `mode=cmy`:

```
\pxpic[mode=cmy]
  {
    {{1,0,1} {1,1,0} {0,1,1} {}      }
    {{}      {1,0,1} {1,1,0} {0,1,1}}
    {{0,1,1} {}      {1,0,1} {1,1,0}}
    {{1,1,0} {0,1,1} {}      {1,0,1}}
  }
```



Showing the difference between a skipped and a white pixel:

```
\pxpicsetup{colours = {w=white}}
\colorbox{gray}{\pxpic{{bbb}{b.b}{bbb}}}
\colorbox{gray}{\pxpic{{bbb}{bwb}{bbb}}}
```

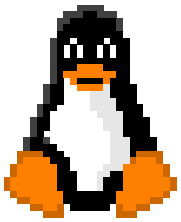


A biggish example: Tux.[1] I put two rows of pixels per code line to reduce the size a

[1] Source: https://www.reddit.com/r/linux/comments/hwpm9j/tux_pixel_art_v10/

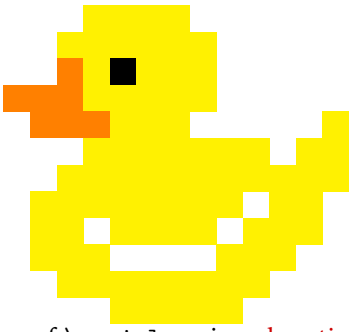
bit and the code is displayed tinily.

```
\pxpic
  [
     size     = 2.5pt
    ,colours = {:=orange,
                '=black!10,
                g=black!75,
                O=orange!80!black}
  ]
  {
    {..........gggg}              {.........gggkkggg}
    {........ggkkkkkkkg}          {........gkkkkkkkkg}
    {........kkkkkkkkkk}          {.......gkk.kkkk.kkk}
    {.......gk.k.kk.k.kk}         {.......gk.k.kk.k.kk}
    {.......gkkkkkkkkkk}          {.......gkkk::::kkkk}
    {.......gk::::::::kk}         {.......gk.:kkkk:kkk}
    {......gkk..::::'kkkk}        {......gkk....''''kkkk}
    {......gkk....''''kkk}        {.....gkk.....''''kkkk}
    {.....gk.......''''kkk}       {....gkk.......''''kkkk}
    {....gkk........''''kkkk}     {....gk.........''kkkk}
    {...gkkg........''''kkkk}     {...gkkkk.......''''kkkk}
    {..OOOOOkkk.....''''kOOOO}    {..O::::kkk.....''''O:::O}
    {.O::::::kk.....''''::::::O}  {.O::::::kkk...''kk::::::O}
    {O::::::::kkk...'kkk::::::::O}{O::::::::kkk...'kkk::::::::O}
    {.O::::::kkkkkkkk::::::::O}   {..OO::::::kkkkkkkk::::::OO}
    {....OO::::O.kkkkkk.O::::OO}  {......OOO.........OOO}
  }
```



Just for Paulo, a duck. Also, showing that the colour definitions in `mode=px` can be arbitrary tokens or multiple letters:

```
\pxpic[colours = {oo=orange,
                  \ylw=yellow,
                  \blk=black},
       skip     = \skp]
  {
    {\skp\skp\skp\ylw\ylw\ylw\ylw}
    {\skp\skp\ylw\ylw\ylw\ylw\ylw\ylw}
    {\skp\skp{oo}\ylw\blk\ylw\ylw\ylw}
    {{oo}{oo}{oo}\ylw\ylw\ylw\ylw\ylw}
    {\skp{oo}{oo}{oo}\ylw\ylw\ylw\skp\skp\skp\skp\skp\ylw}
    {\skp\skp\skp\ylw\ylw\ylw\ylw\ylw\ylw\ylw\skp\ylw\ylw}
    {\skp\skp\ylw\ylw\ylw\ylw\ylw\ylw\ylw\ylw\ylw\ylw\ylw}
    {\skp\ylw\ylw\ylw\ylw\ylw\ylw\ylw\ylw\skp\ylw\ylw}
    {\skp\ylw\ylw\skp\ylw\ylw\ylw\ylw\skp\ylw\ylw\ylw}
    {\skp\ylw\ylw\ylw\skp\skp\skp\skp\ylw\ylw\ylw}
    {\skp\skp\ylw\ylw\ylw\ylw\ylw\ylw\ylw\ylw}
    {\skp\skp\skp\skp\ylw\ylw\ylw\ylw\ylw}
  }
```



Another example might be the definition of `\pxpiclogo` in subsection 2.2.

Who still needs `picture`-mode or complicated packages like pstricks or Ti*k*Z with such pretty pictures?

1.2 Setting options

To control its behaviour pxpic uses a key=value interface powered by expkv. Options can be set either in the optional argument of `\pxpic` or with

`\pxpicsetup` `\pxpicsetup{`⟨*options*⟩`}`

Sets the ⟨*options*⟩ locally to the current TEX group.

Package options are not supported.

The available options are

`colors=`⟨*colour list*⟩
: Define pixel colours for `mode=px`, see subsubsection 1.2.1 for a description of the value's syntax. No pixel definitions are made by the package.

`colours`
: *see* `colors`.

`color-list=⟨choice⟩`
: loads a previously through `\pxpicnewcolorlist` defined colour list. No colour lists are defined by the package.

`colour-list` *see* `color-list`.

`gap-hack=⟨dimen⟩`
: To fix the issues with visible gaps in PDF viewers you can introduce some negative kerns to make the pixels overlap (lines overlap to the top, pixels to the left). This option expects a dimension as its value. A positive value will (maybe) close the gaps, a negative value will introduce real gaps. In any case the outermost pixels' borders still coincide with the borders of the surrounding `\hbox`. Take a look at my babbling about this issue in subsection 1.5.

`ht=⟨dimen⟩` Set the height of the pixels.

`mode=⟨choice⟩`
: Set the used mode, see subsubsection 1.2.2 for available modes. Initial value is `px`.

`size=⟨dimen⟩`
: Set both `ht` and `wd`. Initial value is `1.opt`.

`skip=⟨tokens⟩`
: Define ⟨*tokens*⟩ to be a skip (an empty space of width `wd`) in `mode=px`. No skip definitions are made by the package.

`wd=⟨dimen⟩` Set the width of the pixels.

`b` Set the bottom of the `\pxpic` on the surrounding baseline (vertical bottom alignment; this is the default).

`c` Set the centre of the `\pxpic` on the surrounding baseline (vertical centre alignment).

`t` Set the top of the `\pxpic` on the surrounding baseline (vertical top alignment).

1.2.1 Colour syntax

In the value of the `colours` option you'll have to use the following syntax. Use a comma separated key=value list in which each key corresponds to a new pixel name for `mode=px`, and each value to the used colour. If the colour starts with an opening bracket use the complete value as is behind `\color`, else use the whole value as the first mandatory argument to `\color` with a set of braces added. For example to define `r` as the named colour `red`, and `x` as the colour `#abab0f` (in the HTML colour model) use:

```
colours = {r=red, x=[HTML]{abab0f}}
```

1.2.2 Available modes

`px` As already mentioned, pxpic supports different modes of input. The easiest to use `mode` is `px`, in which each element of the ⟨*pixel list*⟩ has been previously defined as either a coloured pixel (using the `colour` option) or as a skipped pixel (using the `skip` option, resulting in a fully transparent pixel). Each element will be `\detokenized`, so (within TEX's limitations) the name of a pixel can be arbitrary. This is the initial mode pxpic uses. But other options are available as well.

named — Another mode is named, in which each element of the ⟨*pixel list*⟩ should be a named colour (or colour expression) known to xcolor. Each element will be used like so: {\color{⟨*element*⟩}\px}. An exception is an element which is empty ({}), which will be a skipped pixel.

rgb, cmy, cmyk, hsb, Hsb, tHsb, gray, RGB, HTML, HSB, Gray, wave — The modes rgb, cmy, cmyk, hsb, Hsb, tHsb, gray, RGB, HTML, HSB, Gray, and wave correspond to the different colour models supported by xcolor. With these modes each element of the ⟨*pixel list*⟩ will be the values in these colour models, so they'll be used like so: {\color[⟨*mode*⟩]{⟨*element*⟩}\px}. An exception is an element which is empty ({}), which will be a skipped pixel.

You can define additional modes selectable with the mode option using the macros \pxpicnewmode or \pxpicsetmode.

1.3 Other customisation macros

\pxpicnewmode
\pxpicsetmode

\pxpicnewmode{⟨*name*⟩}{⟨*definition*⟩}

You can define your own modes with \pxpicnewmode. Inside ⟨*definition*⟩ #1 is the currently parsed item in the \pxpic ⟨*pixel list*⟩. You can output a pixel using \px, and skip a pixel using \pxskip. The pixel will use the currently active colour (so if you want to draw a red pixel you could use {\color{red}\px}). \pxpicnewmode will throw an error if you try to define a mode which already exists, \pxpicsetmode has no checks on the name.

\pxpicnewcolorlist
\pxpicsetcolorlist
\pxpicaddcolorlist

\pxpicnewcolorlist{⟨*name*⟩}{⟨*colour list*⟩}

This defines a colour list (to be used with the colour-list option). The syntax of ⟨*colour list*⟩ is the same as for the colours option. The pixels aren't directly defined, but only by the use of colour-list=⟨*name*⟩. So

pxpicnewcolorlist{*example*}{*r=red , b=blue , g=green , k=black , w=white*}
pxpicsetup{*colour-list=example*}

would have the same effect as

pxpicsetup{*colours={r=red , b=blue , g=green , k=black , w=white}*}

but a colour-list is more efficient if used multiple times. The new variant will only throw an error if the colour list ⟨*name*⟩ is already defined. The set variant has no such tests, and the add variant will add additional colours to an existing list.

\pxpicforget

\pxpicforget{⟨*px*⟩}

Undefines the ⟨*px*⟩ definition for use in mode=px (or skip symbol) added with the colours (or skip) option.

1.4 Other macros

\px
\pxskip

Inside of a \pxpic the macro \px draws a pixel (of the currently active colour), and \pxskip leaves out a pixel (so this one pixel is fully transparent). Use this in the ⟨*definition*⟩ of a mode in \pxpicnewmode.

\pxpicHT
\pxpicWD

These two are `dimen` registers storing the height and width of the pixels.

\pxpiclogo

`\pxpiclogo[`⟨*size*⟩`]`

This draws the logo of pxpic. The ⟨*size*⟩ controls the pixel size.

1.5 Miscellaneous

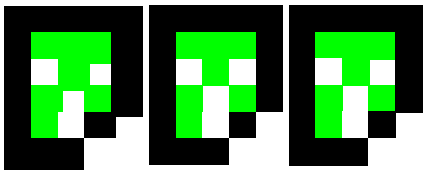
If you find bugs or have suggestions I'll be glad to hear about it, you can either open a ticket on Github (https://github.com/Skillmon/ltx_pxpic) or email me (see the first page).

A similar package is PixelArt, which, as of writing this, is described as a "working draft" by its author. pxpic wasn't intended as a direct competitor (I already started coding pxpic when I learned about PixelArt's existence), but I took inspiration from the "Bugs, Ideas, Undefined behaviours" section of PixelArt's documentation for the syntax of `mode=px`.

Regarding the gap issue: The pixels are output touching each other with no real gap, however some PDF viewers and tools will display such a gap. To make things even worse, the effect depends on the viewers current magnification. pxpic has the `gap-hack` option to provide some crude hack that might fix the issue, at the cost that the pixels on the far right and bottom are bigger than they were specified to be. Also pixels next to skipped pixels have a different size (skipped pixels don't cover pixels to their left or top as they are transparent). You'll want to find a good trade-off value if you want to use `gap-hack`, that mitigates the effect but isn't too big (to make the errors less obvious). You can play with the value and decide for yourself what's the lesser evil. Or you do like me, don't use `gap-hack` and blame the viewers. Here are examples in which you can compare (the `gap-hack` is chosen way too big in this example and skips are used close to white pixels on purpose, but it illustrates the effects; the third output, not shown in the code, uses a

more reasonable `gap-hack=.2pt`):

```
\pxpicsetup
  {
     colours={k=black,g=green,w=white}
    ,skip=.
    ,size=10pt
    ,t
  }
\pxpic[gap-hack=2pt]
  {
    {kkkkk}
    {kgggk}
    {kwg.k}
    {kg.gk}
    {kgwk}
    {kkkw}
  }
\pxpic
  {
    {kkkkk}
    {kgggk}
    {kwg.k}
    {kg.gk}
    {kgwk}
    {kkkw}
  }
```



2 Implementation

Report who we are

```
\ProvidesPackage{pxpic}[2021-12-12 v1.3 draw pixel pictures]
```

and load dependencies

```
\RequirePackage{xcolor}
\RequirePackage{expkv}
```

\pxpicHT
\pxpicWD

These two variables store the height and width of a pixel.

```
\@ifdefinable\pxpicHT{\newdimen\pxpicHT}
\@ifdefinable\pxpicWD{\newdimen\pxpicWD}
\pxpicHT=\p@
\pxpicWD=\pxpicHT
```

(*End definition for* `\pxpicHT` *and* `\pxpicWD`. *These variables are documented on page* 7.)

\pxpic@kern

To fix the visible gaps in some PDF viewers if the user chooses so with the `gap-hack` option we introduce some `\kern`s of the length stored in this register.

```
\@ifdefinable\pxpic@kern{\newdimen\pxpic@kern}
\pxpic@kern=\z@
```

(*End definition for* `\pxpic@kern`.)

\pxpic@inner@box
\pxpic@after@inner@box

To get different vertical alignments we nest one of `\vbox`, `\vtop`, and a lowered `\vbox` inside the outer `\hbox`. The macro `\pxpic@inner@box` will store this information, and since lowering can only be done after the box was set (the alternative would be `\vcenter`, which ends up on a different height), we need to be able to put the box output with `\lower` after the box was collected, which is why we need `\pxpic@after@inner@box`. We default to bottom alignment.

```
\@ifdefinable\pxpic@inner@box{\let\pxpic@inner@box\vbox}
\@ifdefinable\pxpic@after@inner@box{\let\pxpic@after@inner@box\@empty}
```

(*End definition for* `\pxpic@inner@box` *and* `\pxpic@after@inner@box`.)

2.1 Options

We define the options using **expkv** directly (no fancy options are involved and these are just a few anyway).

The first few options are straight forward. We use **expkv**'s name space to actually store the `skip` and `px` definitions, hence we use `\ekvdefNoVal` in the code of `skip`.

```
\protected\ekvdef{pxpic}{size}
  {\pxpicHT=\dimexpr#1\relax\pxpicWD=\pxpicHT}
\protected\ekvdef{pxpic}{ht}{\pxpicHT=\dimexpr#1\relax}
\protected\ekvdef{pxpic}{wd}{\pxpicWD=\dimexpr#1\relax}
\protected\ekvdef{pxpic}{gap-hack}{\pxpic@kern=\dimexpr#1\relax}
\protected\ekvdef{pxpic}{skip}{\ekvdefNoVal{pxpic@px}{#1}{\pxskip}}
```

The `colours` option is parsed using `\ekvparse` and `\pxpic@setcolor`.

```
\protected\ekvdef{pxpic}{colors}{\ekvparse\pxpic@err@noval\pxpic@setcolor{#1}}
\ekvletkv{pxpic}{colours}{pxpic}{colors}
```

And the mode just checks whether the mode macro is defined and lets the auxiliary macro \pxpic@parse@px to the defined mode.

```
\protected\ekvdef{pxpic}{mode}
  {%
    \@ifundefined{pxpic@parse@px@#1}%
      {\pxpic@err@unknown@mode{#1}}%
      {%
        \expandafter\let\expandafter\pxpic@parse@px
          \csname pxpic@parse@px@#1\endcsname
      }%
  }
```

A similar check is done for the colour-list option.

```
\protected\ekvdef{pxpic}{color-list}
  {%
    \@ifundefined{pxpic@colorlist@#1}%
      {\pxpic@err@unknown@colorlist{#1}}
      {\csname pxpic@colorlist@#1\endcsname}%
  }
\ekvletkv{pxpic}{colour-list}{pxpic}{color-list}
```

The alignment options set the internals \pxpic@inner@box and \pxpic@after@inner@box.

```
\protected\ekvdefNoVal{pxpic}{b}
  {%
    \let\pxpic@inner@box\vbox
    \let\pxpic@after@inner@box\@empty
  }
\protected\ekvdefNoVal{pxpic}{c}
  {%
    \def\pxpic@inner@box{\setbox0=\vbox}%
    \def\pxpic@after@inner@box{\lower.5\ht0\box0}%
  }
\protected\ekvdefNoVal{pxpic}{t}
  {%
    \let\pxpic@inner@box\vtop
    \let\pxpic@after@inner@box\@empty
  }
```

2.2 User macros

\pxpic
\pxpic@

\pxpic expands directly to an opened \hbox, the auxiliary \pxpic@ checks for the optional argument and inserts the rest of the code. We need to set \baselineskip to \pxpicHT so that the pixels are stacked vertically without gaps. \pxpic@parse will parse the ⟨*pixel list*⟩ until \pxpic@end is hit. The final \egroup closes the \hbox. The row-wise output is done via a \vbox in which each pixel row will be wrapped inside an \hbox. The \kern negates a negative \kern in \pxpic@parse so that the first line isn't moved.

```
\@ifdefinable\pxpic{\protected\def\pxpic{\hbox\bgroup\pxpic@}}
\newcommand\pxpic@[2][]
  {%
    \pxpicsetup{#1}%
    \pxpic@inner@box
      {%
        \let\px\pxpic@px
```

```
        \let\pxskip\pxpic@skip
        \advance\pxpicHT\pxpic@kern
        \advance\pxpicWD\pxpic@kern
        \baselineskip=\pxpicHT
        \kern\pxpic@kern
        \pxpic@parse#2\pxpic@end
      }%
    \pxpic@after@inner@box
    \egroup
  }
```

(*End definition for* `\pxpic` *and* `\pxpic@`*. These functions are documented on page 2.*)

`\pxpicsetup` Just directly defined to call expkv's parser for the pxpic set.

```
\ekvsetdef\pxpicsetup{pxpic}
```

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

`\pxpiclogo` The logo is just a biggish pixel picture. The `\lower` will move it down a bit so that it appears correctly aligned on the baseline. Since the logo should be part of a normal sentence in most usages we put `\leavevmode` before it. Also we make sure that the `mode` and `px` definitions are correct and the output is bottom aligned.

```
\newcommand*\pxpiclogo[1][.13ex]
  {%
    \begingroup
      \pxpicHT=\dimexpr#1\relax
      \pxpicWD=\pxpicHT
      \pxpic@kern=\z@
      \leavevmode
      \lower3.2\pxpicHT\pxpic
        [b,mode=px,colours={o=[HTML]{9F393D},g=black!75},skip=.]
        {
          {.........................................g}
          {.........................................gggg}
          {.oooo.....................gggg..........ggg}
          {.ooooo...oo......oo...oo....ggggg...gg......g.........g}
          {.oooooooooooo...ooooo..oooo..ggggggggggg...ggggg...ggggggg}
          {..ooooo..oooo.ooooooooooo....ggggg..gggg.ggggggg.gggggggggg}
          {...oooo..oooo....ooooo........gggg..gggg...gggg..gggg.ggg}
          {...oooo..oooo.....oooo........gggg..gggg...gggg..gggg}
          {.oooooo..oooo.....ooooo.....ggggggg..gggg...gggg..gggg}
          {ooooooooooooo...ooooooooo...ggggggggggggg....gggg..gggggggggg}
          {o.oooooooo....ooooo.oooooo.g.gggggggg......ggggg..ggggggg}
          {...ooo.o......o.oo...oo.......ggg.g.........gg......ggg}
          {...ooo.......................ggg}
          {...ooo.......................ggg}
          {....o.........................g}
        }%
    \endgroup
  }
```

(*End definition for* `\pxpiclogo`*. This function is documented on page 7.*)

\pxpicforget

Straight forward, just let the px macro to an undefined macro.

```
\newcommand\pxpicforget[1]
  {\expandafter\let\csname\ekv@name{pxpic@px}{#1}N\endcsname\pxpic@undef}
```

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

\pxpicnewmode
\pxpicsetmode

These are pretty simple as well, the `new` variant will use `\newcommand` which will do the testing for us, the `set` variant uses `\def`.

```
\protected\long\def\pxpicnewmode#1#2%
  {\expandafter\newcommand\csname pxpic@parse@px@#1\endcsname[1]{#2}}
\protected\long\def\pxpicsetmode#1#2%
  {\long\expandafter\def\csname pxpic@parse@px@#1\endcsname##1{#2}}
```

(*End definition for* `\pxpicnewmode` *and* `\pxpicsetmode`. *These functions are documented on page* 6.)

\pxpicnewcolorlist
\pxpicsetcolorlist
\pxpicaddcolorlist
\pxpic@setcolorlist
\pxpic@addcolorlist

The colour list is first parsed with `\ekvparse` inside an `\edef`. `\ekvparse` will prevent the parsed list from further expanding, leaving each list element and `\pxpic@experr@noval` or `\pxpic@setcolor@colorlist` before it. In a second `\edef` these will be expanded, `\pxpic@experr@noval` throwing an error for each element missing a colour definition, and `\pxpic@setcolor@colorlist` testing for an opening bracket (which we do expandably) and leaving the correct definition protected against further expansion. The `add` variant uses a temporary macro for the parsing part and adds the result to the list holding macro. The second expansion step in `set` and both in `add` are done inside a group to revert any definition (also those letting tokens to `\relax` by `\csname`) made at this point except for the list macro itself.

```
\protected\def\pxpicnewcolorlist#1%
  {%
    \@ifundefined{pxpic@colorlist@#1}
      {\pxpicsetcolorlist{#1}}
      {\pxpic@err@defined@colorlist{#1}\@gobble}%
  }
\protected\def\pxpicsetcolorlist#1%
  {\expandafter\pxpic@setcolorlist\csname pxpic@colorlist@#1\endcsname}
\protected\long\def\pxpic@setcolorlist#1#2%
  {%
    \edef#1{\ekvparse\pxpic@experr@noval\pxpic@setcolor@colorlist{#2}}%
    \begingroup\edef#1{\endgroup\protected\def\unexpanded{#1}{#1}}%
    #1%
  }
\protected\def\pxpicaddcolorlist#1%
  {%
    \@ifundefined{pxpic@colorlist@#1}
      {\pxpic@err@unknown@colorlist{#1}\@gobble}
      {\expandafter\pxpic@addcolorlist\csname pxpic@colorlist@#1\endcsname}%
  }
\protected\long\def\pxpic@addcolorlist#1#2%
  {%
    \begingroup
      \edef\pxpic@tmp
        {\ekvparse\pxpic@experr@noval\pxpic@setcolor@colorlist{#2}}%
      \edef\pxpic@tmp
        {%
          \endgroup
```

```
            \protected\def\unexpanded#1{\unexpanded\expandafter{#1}\pxpic@tmp}%
          }%
      \pxpic@tmp
    }
```

(*End definition for* `\pxpicnewcolorlist` *and others. These functions are documented on page* 6.)

2.3 Parser

`\pxpic@ifend` `\pxpic@ifempty` `\pxpic@ifbracket`

These are three helper macros. The first just gobbles everything until the next `\pxpic@end`, and we borrow a fast test for an empty argument from **expkv**. The last can be used to check for an opening bracket if used like `\pxpic@ifbracket\pxpic@end #1.\pxpic@end[]\pxpic@end`.

```
\long\def\pxpic@ifend#1\pxpic@end{}
\let\pxpic@ifempty\ekv@ifempty
\long\def\pxpic@ifbracket#1\pxpic@end[#2]\pxpic@end{\pxpic@ifempty{#2}}
```

(*End definition for* `\pxpic@ifend`, `\pxpic@ifempty`, *and* `\pxpic@ifbracket`.)

`\pxpic@openbrace`

For some weirder TEX programming it is sometimes necessary to insert an unmatched opening brace. This code does exactly that if it's expanded twice. It is put into a single macro so that one can `\expandafter` it easier.

```
\newcommand*\pxpic@openbrace{\expandafter{\iffalse}\fi}
```

(*End definition for* `\pxpic@openbrace`.)

`\pxpic@parse` `\pxpic@done`

The parsing loop is pretty simple, first check whether we're done, else open a new `\hbox` (which will form a row in the `\vbox` placed by `\pxpic@`) in which the inner parsing loop is run. Then call the next iteration. If we're done just gobble the remainder of the current iteration. First we introduce our `\kern` which might fix the gap issue. Another `\kern` is done at the start of each `\hbox` to compensate the unnecessary `\kern` done by the first `\pxpic@parseline`.

```
\newcommand\pxpic@parse[1]
  {%
    \pxpic@ifend#1\pxpic@done\pxpic@end
    \kern-\pxpic@kern
    \hbox{\kern\pxpic@kern\pxpic@parseline#1\pxpic@end}%
    \pxpic@parse
  }
\long\def\pxpic@done\pxpic@end\kern-\pxpic@kern\hbox#1\pxpic@parse{}
```

(*End definition for* `\pxpic@parse` *and* `\pxpic@done`.)

`\pxpic@parseline` `\pxpic@linedone`

The line parsing loop also checks whether we're done, if not we place a pixel using the current definition of `\pxpic@parse@px` (which will be set by the current `mode`) and afterwards call the next iteration. If we're done we gobble the remainder of the current iteration and control goes back to `\pxpic@parse`. Before each pixel we introduce a negative `\kern` to maybe fix the gap issue by letting the pixels overlap a bit.

```
\newcommand\pxpic@parseline[1]
  {%
    \pxpic@ifend#1\pxpic@linedone\pxpic@end
    \kern-\pxpic@kern
    \pxpic@parse@px{#1}%
```

```
    \pxpic@parseline
  }
\long\def\pxpic@linedone
    \pxpic@end\kern-\pxpic@kern\pxpic@parse@px#1\pxpic@parseline
  {}
```

(*End definition for* `\pxpic@parseline` *and* `\pxpic@linedone`.)

2.4 Modes

The modes define how a single element of the ⟨*pixel list*⟩ is parsed.

`\pxpic@parse@px@px` `\pxpic@parse@px` In the px mode we check whether the pixel is defined (using the name space of **expl**^{kv}), if so call it, else throw an error and skip. Since this is also the initial `mode` we `\let` the auxiliary macro `\pxpic@parse@px` to this mode here.

```
\newcommand\pxpic@parse@px@px[1]
  {%
    \ekvifdefinedNoVal{pxpic@px}{#1}
      {\csname\ekv@name{pxpic@px}{#1}N\endcsname}%
      {%
        \pxpic@err@unknown@px{#1}%
        \pxskip
      }%
  }
\let\pxpic@parse@px\pxpic@parse@px@px
```

(*End definition for* `\pxpic@parse@px@px` *and* `\pxpic@parse@px`.)

`\pxpic@parse@px@named` `named` just checks whether the skip is empty. If so skip, else call `\color` with the element and output a pixel.

```
\newcommand\pxpic@parse@px@named[1]
  {%
    \pxpic@ifempty{#1}
      {\pxskip}
      {{\@declaredcolor{#1}\px}}%
  }
```

(*End definition for* `\pxpic@parse@px@named`.)

`\pxpic@parse@px@rgb` `\pxpic@parse@px@cmy` `\pxpic@parse@px@cmyk` `\pxpic@parse@px@hsb` `\pxpic@parse@px@Hsb` `\pxpic@parse@px@tHsb` `\pxpic@parse@px@gray` `\pxpic@parse@px@RGB` `\pxpic@parse@px@HTML` `\pxpic@parse@px@HSB` `\pxpic@parse@px@Gray` `\pxpic@parse@px@wave` The colour model modes are all the same in principle. They test for an empty element to introduce a skip, else they call `\color` with the respective colour model and output a pixel. We use the auxiliary `\pxpic@tmp` to do all those definitions and undefine it afterwards.

```
\def\pxpic@tmp#1%
  {%
    \pxpicnewmode{#1}%
      {%
        \pxpic@ifempty{##1}
          {\pxskip}
          {{\@undeclaredcolor[#1]{##1}\px}}%
      }%
  }
\pxpic@tmp{rgb}
\pxpic@tmp{cmy}
```

```
\pxpic@tmp{cmyk}
\pxpic@tmp{hsb}
\pxpic@tmp{Hsb}
\pxpic@tmp{tHsb}
\pxpic@tmp{gray}
\pxpic@tmp{RGB}
\pxpic@tmp{HTML}
\pxpic@tmp{HSB}
\pxpic@tmp{Gray}
\pxpic@tmp{wave}
\let\pxpic@tmp\pxpic@undef
```

(*End definition for* `\pxpic@parse@px@rgb` *and others.*)

2.5 Pixel and Skip

`\pxpic@px` `\pxpic@skip`

The actual definition of pixels and skips is stored in macros to which the frontend macros `\px` and `\pxskip` will be let inside of `\pxpic`.

```
\newcommand\pxpic@px{\vrule height\pxpicHT width\pxpicWD depth\z@}
\newcommand\pxpic@skip{\kern\pxpicWD}
```

(*End definition for* `\pxpic@px` *and* `\pxpic@skip`.)

2.6 Parser for colours

`\pxpic@setcolor` `\pxpic@setcolor@a` `\pxpic@setcolor@b`

First we test whether the colour starts with an opening bracket or not. Depending on that we either just put the colour after `\color`, or put braces around it (as it then is a colour expression for xcolor and just a single argument). `\pxpic@setcolor` defines a `px` in the name space of **expkv** (this has a slight overhead during definition, but **expkv** is fast in checking whether one of its keys is defined or not, and reduces the amount of code in this package).

```
\newcommand\pxpic@setcolor[2]
  {%
    \pxpic@ifbracket\pxpic@end#2.\pxpic@end[]\pxpic@end
      \pxpic@setcolor@a\pxpic@setcolor@b
      {#1}{#2}%
  }
\newcommand\pxpic@setcolor@a[2]
  {%
    \expandafter\def\csname\ekv@name{pxpic@px}{#1}N\endcsname
      {{\@declaredcolor{#2}\px}}%
  }
\newcommand\pxpic@setcolor@b[2]
  {%
    \expandafter\def\csname\ekv@name{pxpic@px}{#1}N\endcsname
      {{\@undeclaredcolor#2\px}}%
}
```

(*End definition for* `\pxpic@setcolor`, `\pxpic@setcolor@a`, *and* `\pxpic@setcolor@b`.)

`\pxpic@setcolor@colorlist`

This macro should leave the correct code in the input stream to define a single pixel. It is to be used inside of `\edef`, hence using `\unexpanded`, which doesn't have an opening brace directly after it so that the `\pxpic@ifbracket` test is fully expanded. Next we

expand \pxpic@setcolor@a/b twice (which will expand the \csname contained in it) and then leave the opening bracket for \unexpanded in the input stream. The code should be used inside a group so that all the implicit definitions to \relax done by \csname are reverted.

```
\newcommand\pxpic@setcolor@colorlist[2]
  {%
    \unexpanded\iffalse{\fi
    \pxpic@ifbracket\pxpic@end#2.\pxpic@end[]\pxpic@end
      {\expandafter\expandafter\expandafter\pxpic@openbrace\pxpic@setcolor@a}
      {\expandafter\expandafter\expandafter\pxpic@openbrace\pxpic@setcolor@b}
      {#1}{#2}%
    }%
  }
```

(*End definition for* \pxpic@setcolor@colorlist.)

2.7 Messages

\pxpic@err@noval
\pxpic@err@unknown@px
\pxpic@err@unknown@mode
\pxpic@err@unknown@colorlist
\pxpic@err@defined@colorlist

These are just some macros throwing errors, nothing special here.

```
\newcommand\pxpic@err@noval[1]
  {\PackageError{pxpic}{Missing colour definition for name `\detokenize{#1}'}{}}
\newcommand\pxpic@err@unknown@px[1]
  {\PackageError{pxpic}{Unknown pixel `\detokenize{#1}'. Skipping}{}}
\newcommand\pxpic@err@unknown@mode[1]
  {\PackageError{pxpic}{Unknown mode `#1'}{}}
\newcommand\pxpic@err@unknown@colorlist[1]
  {\PackageError{pxpic}{Unknown colour list `#1'}{}}
\newcommand\pxpic@err@defined@colorlist[1]
  {\PackageError{pxpic}{Colour list `#1' already defined}{}}
```

(*End definition for* \pxpic@err@noval *and others.*)

\pxpic@experr

This macro can be used to throw an error expandably. For this an undefined control sequence \pxpic␣Error: is used. The group containing \expandafter keeps the definition of \pxpic␣Error: local (it is \relax after the \csname) so that it is undefined when it's used. The \@firstofone is needed to get the readable output (now the undefined macro and actual message are always the same argument).

```
\def\pxpic@experr#1%
  {%
    \long\def\pxpic@experr##1%
      {%
        \expandafter\expandafter\expandafter
        \pxpic@ifend
          \@firstofone{#1##1.}%
        \pxpic@end
      }%
  }
\begingroup\expandafter\endgroup
\expandafter\pxpic@experr\csname pxpic Error:\endcsname
```

(*End definition for* \pxpic@experr.)

\pxpic@experr@noval With the expandable error throwing mechanism out of the way, the following is straight forward again.

```
\newcommand\pxpic@experr@noval[1]
  {\pxpic@experr{Missing colour definition for ‘#1’}}
```

(*End definition for* `\pxpic@experr@noval`.)

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.