The TEXPower bundle
Documentation*

Stephan Lehmke
mailto:Stephan.Lehmke@udo.edu

Hans Fr. Nordhaug
mailto:hansfn@users.sourceforge.net

May 22, 2007

Contents

*Documentation for TEXPower v0.2 of April 8, 2005.

The TEXPower bundle contains style and class files for creating dynamic online presentations with LATEX.

The heart of the bundle is the package `texpower.sty` which implements some commands for presentation effects. This includes setting page transitions, color highlighting and displaying pages incrementally.

For finding out how to achieve special effects (as shown in the **??**), please look at the comments inside the files ending with `example.tex` and `demo.tex` and read this manual to find out what's going on.

For your own first steps with TEXPower, the simple demo file `simpledemo.tex` is the best starting place. There, some basic applications of the dynamic features provided by the `texpower` package are demonstrated. You can make your own dynamic presentations by modifying that demo to your convenience.

`simpledemo.tex` uses the `article` document class for maximum compatibility. There are other simple demos named `slidesdemo.tex`, `foilsdemo.tex`, `seminardemo.tex`, `pp4sldemo.tex`, `pdfslidemo.tex`, `pdfscrdemo.tex`, `prosperdemo.tex`, and `ifmslidemo.tex` which demonstrate how to combine TEXPower with the most popular presentation-making document classes and packages.

The other, more sophisticated examples demonstrate the expressive power of the

texpower package. Look at the commented code of these examples to find out how to achieve special effects and create your own presentation effects with TEXPower.

1 Usage and general options

The texpower package is loaded by putting

```
\usepackage{texpower}
```

into the preamble of a document.

There are no specific restrictions as to which document classes can be used.

It should be stressed that TEXPower is **not** (currently) a complete presentation package. It just adds dynamic presentation effects (and some other gimmicks specifically interesting for dynamic presentations) and should always be combined with a document class dedicated to designing presentations (or a package like pdfslide).

Some of the presentation effects created by texpower require special capabilities of the viewer which is used for presenting the resulting document. The target for the development of texpower has so far been **Adobe Acrobat® Reader**, which means the document should (finally) be produced in pdf format. The produced pdf documents should display well in **GSview** also, but that viewer doesn't support page transitions and duration.

There are no specific restrictions as to which way the pdf format is produced. All demos and examples have been tested with pdfLATEX and standard LATEX, using dvips and **Adobe Acrobat® Distiller** or dvips and ps2pdf (from the **Ghostscript suite**) for generating pdf.

1.1 General options

option: display. Enable 'dynamic' features. If not set, it is assumed that the document is to be printed, and all commands for dynamic presentations, like \pause or \stepwise have no effect.

option: printout (default). Disable 'dynamic' features. As this is the default behaviour, setting this option explicitly is useful only if the option display is set by default for instance in the tpoptions.cfg file (see section 1.5).

option: verbose. Output some administrative info.

Some font options are listed in section 1.3.

1.2 Side effects of page contents duplication

In the implementation of the \pause and \stepwise commands, it is neccessary to duplicate some material on the page.

This way, not only 'visible' page contents will be duplicated, but also some 'invisible' control code stored in **whatsits** (see the TEXbook for an explanation of this concept). Duplicating whatsits can lead to undesirable side effects.

For instance, a `\section` command creates a whatsit for writing the table of contents entry. Duplicating this whatsit will also duplicate the toc entry. So, whatsit items effecting file access are inhibited when duplicating page material.

The current version of `texpower` is a little smarter when handling whatsits. Some commands (related to writing to files and hyperlinks) are made stepwise-aware. This means that links can point to the actual subpage where the anchor is and not to the last (sub)page of an incremental page. However, if you want the old behaviour just use

option: `oldfiltering` switches on the old (pre 0.2) very aggressive/robust filtering of whatsits.

The `oldfiltering` can be turned on and off inside the document using `\oldfilteringon/off`. This command is useful if `texpower` isn't smart enough...

A second type of whatsits is created by TEX's `\special` command which is used for instance for color management. Some drivers, like `dvips` and `textures`, use a color stack which is controlled by `\special` items included in the dvi file. When page contents are duplicated, then these `\special`s are also duplicated, which can seriously mess up the color stack.

`texpower` implements a 'color stack correction' method by maintaining a stack of color corrections, which should counteract this effect. Owing to potential performance problems, this method is turned off by default.

option: `fixcolorstack` switches on color stack correction. Use it if you experience strange color switches in your document.

1.3 Setting the base font

`texpower` offers no options for setting the base font of the document. Use the `tpslifonts` package in stead. Read more in section 1.9.

Further, there are packages like `cmbright` or `beton` which change the whole set of fonts to something less fragile than cmr.

1.4 Switches

There are some boolean registers provided and set automatically by `texpower`.

boolean: `psspecialsallowed` True if PostScript® specials may be used.

`texpower` tries to find out whether or not PostScript® specials may be used in the current document. For instance, pdfLATEX can't interpret arbitrary specials. This switch is set automatically and can be used inside a document to enable/disable parts which need PostScript® specials.

boolean: `display` True if `display` option was given.

This switch indicates whether 'dynamic' features of `texpower` are enabled. Use it inside your document to distinguish between the 'presented' and the printed version of your document.

boolean: `TPcolor` True if any of the color highlighting options (see section 5) were given.

This switch indicates whether 'color' features of `texpower` are enabled (compare section 5). You can use it inside your document to distinguish between a 'colored' and a 'monochrome' version of your document.

1.5 Configuration files

`texpower` loads three configuration files (if present):

file: `tpoptions.cfg` is loaded before options are processed. Can be used to set default options in a system-specific way. See the comments inside the file `tpoptions.cfg` which is part of the TEXPower bundle for instructions.

file: `tpsettings.cfg` is loaded at the end of `texpower`. Here, you can do some system-specific settings. See the comments inside the file `tpsettings.cfg` which is part of the TEXPower bundle for instructions.

file: `tpcolors.cfg` is loaded if `TPcolor` is true. The file defines the standard colors/colorsets (see section 5). See the comments inside the file `tpcolors.cfg` which is part of the TEXPower bundle for instructions.

1.6 Miscellaneous commands

Some important commands that don't fit in the latter sections:

`\oldfilteringon` reverts to the old (pre v0.2) aggressive/robust filtering of whatsits.

`\oldfilteringoff` turns on the new better treatment of whatsits.

`\currentpagevalue{⟨value⟩}` sets how to find the number of the current page, `\value{page}` is default. Used to name the hyper target on the first subpage of every page. Also used in the TeXPower navigation buttons.

`\pausesafecounter{⟨counter⟩}` is used to add counters that are to be restored to their original value after `\pause`. The `page` counter is always restored. In addition the `slide` counter is restored if the `seminar` class is used. If you need more counters to be restored after `\pause`, use `\pausesafecounter`.

1.7 Page Anchors

For each physical page TEXPower (when in display mode) makes a number of subpages - this is the dynamics. For convenience TEXPower defines an anchor to the first subpage of physical page n, `firstpage.n`. The standard page anchor for physical page n, `page.n`, points to the last subpage of physical page n. If you want to link to any other subpage just insert a `\hyperlink` in the standard way assuming you haven't turned on the old filtering (1.2).

1.8 Dependencies on other packages

`textpower` always loads the packages `ifthen` and `calc`, as the extended command syntax provided by these is indispensable for the macros to work. They are in the `base` and `tools` area of the LATEX distribution, respectively, so I hope they are available on all systems.

Furthermore, `texpower` loads the package `color` if any color-specific options are set (see section 5).

Further packages are *not* loaded automatically by `texpower` to avoid incompatibilities, although some features of `texpower` are enabled *only* if a certain package is loaded. If you wish to use these features, you are responsible for loading the respective package yourself.

If some necessary package is *not* loaded, `texpower` will issue a warning and disable the respective features.

The following packages are neccessary for certain features of `texpower`:

package: `hyperref` is neccessary for page transition effects to work (see section 4).

In particular, the `\pageDuration` (see section 4.2) command only works if the version of hyperref loaded is at least v6.70a (where the `pdfpageduration` key was introduced).

Commands which work only when `hyperref` is loaded are marked with **h** in the description.

package: `soul` is neccessary for the implementation of the commands `\hidetext` and `\highlighttext` (see section 3.6).

Commands which work only when `soul` is loaded are marked with **s** in the description.

1.9 What else is part of the TEXPower bundle?

Besides the package `texpower` (which is described here), there are four more packages, `tpslifonts`, `fixseminar`, `automata` and `tplists`, and one document class, `powersem`, in the TEXPower bundle. Except for `tpslifonts` and `tplists` these files have no documentation of their own. They will be described in this section until they are turned into `dtx` files producing their own documentation.

See the file `00readme.txt` which is part of the TEXPower bundle for a short description of all files.

The document class `powersem`

This is planned to provide a more 'modern' version of `seminar` which can be used for creating dynamic presentations.

Currently, this document class doesn't do much more than load `seminar` and apply some fixes, but it is planned to add some presentation-specific features (like navigation panels).

There are three new options which are specific for `powersem`, all other options are passed to `seminar`:

option: `display` Turns off all features of `seminar` (notes, vertical centering of slides) which can disturb dynamic presentations.

option: `calcdimensions` `seminar` automatically calculates the slide dimensions `\slidewidth` and `\slideheight` only for the default `letter` and for its own option `a4`. For all the other paper sizes which are possible with the `KOMA` option, the slide dimensions are not calculated automatically.

The `calcdimensions` option makes `powersem` calculate the slide dimensions automatically from paper size and margins.

option: `truepagenumbers` The truepagenumbers option makes powersem count pages with the counter page, independently of the counter slide. This enables proper working of TeXPowers navigation buttons (some of which calculate relative page numbers) even when the counter slide is reset frequently (for slide numberings of the type `<l>.<n>.<m>`).

option: `KOMA` Makes `seminar` load `scrartcl` (from the KOMA-Script bundle) instead of `article` as its base class. All new features of `scrartcl` are then available also for slides.

option: `UseBaseClass` Makes `seminar` load the class `\baseclass` (initially `article`) instead of `article` as its base class.

option: `reportclass` Makes `seminar` load the class `\baseclass` (initially `report`) instead of `article`.

option: `bookclass` Makes `seminar` load the class `\baseclass` (initially `book`) instead of `article`.

There is one change in `powersem` which will lead to incompatibilities with `seminar`. `seminar` has the unfortunate custom of *not* exchanging `\paperwidth` and `\paperheight` when making landscape slides, as for instance `typearea` and `geometry` do.

This leads to problems with setting the paper size for `pdf` files, as done for instance by the `hyperref` package.

`powersem` effectively turns off `seminar`'s papersize management and leaves this to the base class (with the pleasant side effect that you can use e. g. `\documentclass[KOMA,a0paper]{powersem}` for making posters).

In consequence, the `portrait` option of `seminar` is turned on by `powersem` to avoid confusing `seminar`. You have to explicitly use the `landscape` option (and a base class or package which understands this option) to get landscape slides with powersem.

The package `fixseminar`

Unfortunately, there are some fixes to seminar which can *not* be applied in `powersem` because they have to be applied after `hyperref` is loaded (if this package should be loaded).

The package `fixseminar` applies these fixes, so this package should be loaded after `hyperref` (if `hyperref` is loaded at all, otherwise `fixseminar` can be loaded anywhere in the preamble).

It applies two fixes:

- In case `pdflatex` is being run, the lengths `\pdfpageheight` and `\pdfpagewidth` have to be set in a 'magnification-sensitive' way.
- `hyperref` introduces some code at the beginning of every page which can produce spurious vertical space, which in turn disturbs building dynamic pages. This code is 'fixed' so it cannot produce vertical space.

The package `tpslifonts`

Presentations to be displayed 'online' with a video beamer have special needs concerning font configuration owing to low 'screen' resolution and bad contrast caused by possibly bad light conditions combined with color highlighting.

This package tries to cater to these needs by offering a holistic configuration of all document fonts, including text, typewriter, and math fonts. Special features are 'smooth scaling' of Type1 fonts and careful design size selection for optimal readability.

For more information on package options and used fonts (and on implementation) read the documentation coming with the package - check the `tpslifonts` directory.

The package `automata`

Experimental package for drawing automata in the sense of theoretical computer science (using PSTricks) and animating them with TeXPower. Only DFA and Mealy automata are supported so far.

The package `tplists`

Experimental package providing easy dynamic lists. Currently there are stepped, flipped and dimmed versions of itemize and enumerate (and corresponding lists from the `eqlist` and `paralist` package). For more information and an example, compile (and then read) the file `tplists.dtx`.

2 The \pause command

\pause is derived from the \pause command from the package texpause which is part of the PPower4 suite by Klaus Guntermann.

It will ship out the current page, start a new page and copy whatever was on the current page onto the new page, where typesetting is resumed.

This will create the effect of a **pause** in the presentation, i. e. the presentation stops because the current page ends at the point where the \pause command occurred and is resumed at this point when the presenter switches to the next page.

Things to pay attention to

1. \pause should appear in **vertical mode** only, i. e. between paragraphs or at places where ending the current paragraph doesn't hurt.

2. This means \pause is forbidden in all **boxed** material (including tabular), **headers/footers**, and **floats**.

3. \pause shouldn't appear either in environments which have to be *closed* to work properly, like picture, tabbing, and (unfortunately) environments for **aligned math formulas**.

4. \pause does work in all environments which mainly influence paragraph formatting, like center, quote or all **list** environments.

5. \pause doesn't really have problems with automatic page breaking, but beware of *overfull* pages/slides. In this case, it may occur that only the last page(s)/slide(s) of a sequence are overfull, which changes vertical spacing, making lines 'wobble' when switching to the last page/slide of a sequence.

6. The duplication of page material done by \pause can lead to unwanted side effects. See section 1.2 for further explanations. In particular, if you should experience strange color switches when using \pause (and you are **not** using pdftex), turn on color stack correction with the option fixcolorstack. In addition you should be aware of \pausesafecounter, see section 1.6.

A lot of the restrictions for the use of pause can be avoided by using \stepwise (see next section).

3 The \stepwise command

\stepwise{⟨contents⟩} is a command for displaying some part of a LaTeX document (which is contained in ⟨contents⟩) 'step by step'. As of itself, \stepwise doesn't do very much. If ⟨contents⟩ contains one or more constructs of the form \step{⟨stepcontents⟩}, the following happens:

1. The current contents of the page are saved (as with `\pause`).

2. As many pages as there are `\step` commands in ⟨`contents`⟩ are produced.

   Every page starts with what was on the current page when `\stepwise` started.

   The first page also contains everything in ⟨`contents`⟩ which is *not* in ⟨`stepcontents`⟩ for any `\step` command.

   The second page additionally contains the ⟨`stepcontents`⟩ for the *first* `\step` command, and so on, until all ⟨`stepcontents`⟩ are displayed.

3. When all ⟨`stepcontents`⟩ are displayed, `\stepwise` ends and typesetting is resumed (still on the current page).

This will create the effect that the `\step` commands are executed 'step by step'.

Things to pay attention to

1. `\stepwise` should appear in **vertical mode** only, i. e. between paragraphs, just like `\pause`.

2. Don't put `\pause` or nested occurrences of `\stepwise` into ⟨`contents`⟩.

3. Structures where `\pause` does not work (like `tabular` or aligned equations) can go *completely* into ⟨`contents`⟩, where `\step` can be used freely (see **??**).

4. As ⟨`contents`⟩ is read as a macro argument, constructs involving **catcode** changes (like `\verb` or language switches) won't work in ⟨`contents`⟩ **unless** you use the `fragilesteps` environment (3.1).

5. Several instances of `\stepwise` may occur on one page, also combined with `\pause` (outside of ⟨`contents`⟩).

   But beware of page breaks in ⟨`contents`⟩. This will really mess things up.

   Overfull pages/slides are also a problem, just like with `\pause`. See the description of `\pause` (section 2) concerning this and also concerning side effects of duplicating page material.

6. `\step` can go in ⟨`stepcontents`⟩. The order of execution of `\step` commands is just the order in which they appear in ⟨`contents`⟩, independent of nesting within each other.

7. As ⟨`contents`⟩ is executed several times, LaTeX constructs changing **global counters**, accessing **files** etc. are problematic. This concerns sections, numbered equations, labels, hyperlinks and the like.

   Counters are taken care of explicitly by `\stepwise`, so equation numbers are no problem.

   Commands accessing toc files and such (like `\section`) are taken care of by the whatsit suppression mechanism (compare section 1.2).

3.1 `fragilesteps` environment

The `fragilesteps` environment is a wrapper around `\stepwise` that makes it possible to use verbatim. The code for this environment is based on similar code from beamer - an excellent presentation class written by Till Tantau - thanks! Using the `fragilesteps` environment enables the use of the `listings` package to display code line by line. There are some examples in `verbexample.tex`.

3.2 `\boxedsteps` and `\nonboxedsteps`

By default, ⟨`stepcontents`⟩ belonging to a `\step` which is not yet 'active' are ignored altogether. This makes it possible to include e. g. tabulators `&` or line breaks into ⟨`stepcontents`⟩ without breaking anything.

Sometimes, however, this behaviour is undesirable, for instance when stepping through an equation 'from outer to inner', or when filling in blanks in a paragraph. Then, the desired behaviour of a `\step` which is not yet 'active' is to create an appropriate amount of *blank space* where ⟨`stepcontents`⟩ can go as soon as it is activated.

The simplest and most robust way of doing this is to create an empty box (aka `\phantom`) with the same dimensions as the text to be hidden.

This behaviour is toggled by the following commands. See section 3.6 for more sophisticated (albeit more fragile) variants.

`\boxedsteps` makes `\step` create a blank box the size of ⟨`stepcontents`⟩ when inactive and put ⟨`stepcontents`⟩ into a box when active.

`\nonboxedsteps` makes `\step` ignore ⟨`stepcontents`⟩ when inactive and leave ⟨`stepcontents`⟩ alone when active (default).

Things to pay attention to

1. The settings effected by `\boxedsteps` and `\nonboxedsteps` are *local*, i. e. whenever a group closes, the setting is restored to its previous value.

2. Putting stuff into boxes can break things like tabulators (`&`). It can also mess up math spacing, which then has to be corrected manually. Compare the following examples:

$$\left(\frac{a+b}{c}\right) \qquad \left(\frac{a+b}{c}\right) \qquad \left(\frac{a+b}{c}\right)$$

3.3 Custom versions of `\stepwise`

Sometimes, it might happen that vertical spacing is different on every page of a sequence generated by `\stepwise`, making lines 'wobble'. This is usually fixed if you use `\liststepwise` or `\parstepwise` (described below) in stead of `\stepwise`.

There are two custom versions of `\stepwise` which should produce better vertical spacing.

`\liststepwise{⟨contents⟩}` works exactly like `\stepwise`, but adds an 'invisible rule' before ⟨contents⟩. Use for list environments and aligned equations.

`\parstepwise{⟨contents⟩}` works like `\liststepwise`, but `\boxedsteps` is turned on by default. Use for texts where `\step`s are to be filled into blank spaces.

3.4 Starred versions of `\stepwise` commands

Usually, the first page of a sequence produced contains *only* material which is *not* part of any ⟨stepcontents⟩. The first ⟨stepcontents⟩ are displayed on the second page of the sequence.

For special effects (see example **??**), it might be desirable to have the first ⟨stepcontents⟩ active even on the first page of the sequence.

All variants of `\stepwise` have a starred version (e. g. `\stepwise*`) which does exactly that.

3.5 The optional argument of `\stepwise`

Every variant of `\stepwise` takes an optional argument, like this

`\stepwise[⟨settings⟩]{⟨contents⟩}`.

⟨settings⟩ will be placed right before the internal loop which produces the sequence of pages. It can contain settings of parameters which modify the behaviour of `\stepwise` or `\step`. ⟨settings⟩ is placed inside a group so all changes are local to this call of `\stepwise`.

Some internal macros and counters which can be adjusted are explained in the following.

3.6 Customizing the way ⟨stepcontents⟩ is diplayed

Internally, there are three macros (taking one argument each) which control how ⟨stepcontents⟩ is displayed: `\displaystepcontents`, `\hidestepcontents`, and `\activatestep`. Virtually, every `\step{⟨stepcontents⟩}` is replaced by

`\hidestepcontents{⟨stepcontents⟩}`
when this step is not yet active.

`\displaystepcontents{\activatestep{⟨stepcontents⟩}}` when this step is activated *for the first time.*

`\displaystepcontents{⟨stepcontents⟩}`
when this step has been activated before.

By redefining these macros, the behaviour of `\step` is changed accordingly. You can redefine them inside ⟨`contents`⟩ to provide a change affecting one `\step` only, or in the optional argument of `\stepwise` to provide a change for all `\step`s inside ⟨`contents`⟩.

In the **??**, it is demonstrated how special effects can be achieved by redefining these macros.

`\activatestep` is set to `\displayidentical` by default, the default settings of `\hidestepcontents` and `\displaystepcontents` depend on whether `\boxedsteps` or `\nonboxedsteps` (default) is used.

`texpower` offers nine standard definitions.

For interpreting `\displaystepcontents`:

`\displayidentical` Simply expands to its argument. The same as LaTeXs `\@ident`. Used by `\nonboxedsteps` (default).

`\displayboxed` Expands to an `\mbox` containing its argument. Used by `\boxedsteps`.

For interpreting `\hidestepcontents`:

`\hideignore` Expands to nothing. The same as LaTeXs `\@gobble`. Used by `\nonboxedsteps` (default).

`\hidephantom` Expands to a `\phantom` containing its argument. Used by `\boxedsteps`.

`\hidevanish` In a colored document, makes its argument 'vanish' by setting all colors to `\vanishcolor` (defaults to `pagecolor`; compare section 5.7). Note that this will give weird results with structures backgrounds.

For monochrome documents, there is no useful interpretation for this command, so it is disabled.

s `\hidetext` Produces blank space of the same dimensions as the space that would be occupied if its argument would be typeset in the current paragraph. Respects automatic hyphenation and line breaks.

This command needs the `soul` package to work, which is not loaded by `texpower` itself. Consult the documentation of `soul` concerning restrictions on commands implemented using `soul`. If you don't load the `soul` package yourself, there is no useful definition for this command, so it defaults to `\hidephantom`.

`\hidedimmed` In a colored document, displays its argument with dimmed colors (compare section 5.8). Note that this doesn't make the argument completely invisible.

For monochrome documents, there is no useful interpretation for this command, so it is disabled.

For interpreting `\activatestep`:

`\highlightboxed` If the `colorhighlight` option (see section 5) is set, expands to a box with colored background containing its argument. Otherwise, expands to an `\fbox` containing its argument. It is made sure that the resulting box has the same dimensions as the argument (the outer frame may overlap surrounding text).

There is a new length register `\highlightboxsep` which acts like `\fboxsep` for the resulting box and defaults to `0.5\fboxsep`.

s `\highlighttext` If the `colorhighlight` option (see section 5) is set, puts its argument on colored background. Otherwise, underlines its argument. It is made sure that the resulting text has the same dimensions as the argument (the outer frame may overlap surrounding text).

`\highlightboxsep` is used to determine the extent of the coloured box(es) used as background.

This command needs the `soul` package to work (compare the description of `\hidetext`). If you don't load the `soul` package yourself, there is no useful definition for this command, so it is disabled.

`\highlightenhanced` In a colored document, displays its argument with enhanced colors (compare section 5.8).

For monochrome documents, there is no useful interpretation for this command, so it is disabled.

3.7 Variants of `\step`

There are a couple of custom versions of `\step` which make it easier to achieve special effects needed frequently.

`\bstep` Like `\step`, but is *always* boxed (see section 3.2). `\bstep{⟨stepcontents⟩}` is implemented in principle as `{\boxedsteps\step{⟨stepcontents⟩}}`.

In aligned equations where `\stepwise` is used for being able to put tabulators into ⟨`stepcontents`⟩, but where nested occurrences of `\step` should be boxed to assure correct sizes of growing braces or such, this variant of `\step` is more convenient than using `\boxedsteps` for every nested occurrence of `\step`.

`\switch{⟨ifinactive⟩}{⟨ifactive⟩}` is a variant of `\step` which, instead of making its argument appear, switches between ⟨`ifinactive`⟩ and ⟨`ifactive`⟩ when activated.

In fact, `\step{⟨stepcontents⟩}` is in principle implemented by

```
\switch{\hidestepcontents{⟨stepcontents⟩}}
       {\displaystepcontents{⟨stepcontents⟩}}
```

This command can be used, for instance, to add an `\underbrace` to a formula, which is difficult using `\step`.

Beware of problems when ⟨`ifinactive`⟩ and ⟨`ifactive`⟩ have different dimensions.

`\dstep` A variant of `\step` which takes **no** argument, but simply switches colors to 'dimmed' (compare section 5.8) if not active. Not that the scope of this color change will last until the next outer group closes. This command does nothing in a monochrome document.

`\vstep` A variant of `\step` which takes **no** argument, but simply switches all colors to `\vanishcolor` (defaults to `pagecolor`; compare section 5.7) if not active. Not that the scope of this color change will last until the next outer group closes. This command does nothing in a monochrome document.

`\steponce` Like `\step`, but goes inactive again in the subsequent step.

`\multistep` is a shorthand macro for executing several steps successively. In fact, it would better be called `\multiswitch`, because it's functionality is based on `\switch`, it only acts like a (simplified) `\step` command which is executed 'several times'. The syntax is

```
\multistep[⟨activatefirst⟩]{⟨n⟩}{⟨stepcontents⟩}
```

where ⟨`n`⟩ is the number of steps. Only one instance of ⟨`stepcontents`⟩ is displayed at a time. Inside ⟨`stepcontents`⟩, a counter `substep` can be evaluated which tells the number of the current instance. In the starred form the last instance of ⟨`stepcontents`⟩ stays visible.

`\movie` works like `\multistep`, but between `\steps`, pages are advanced automatically every ⟨`dur`⟩ seconds. The syntax is

```
\movie{⟨n⟩}{⟨dur⟩}[⟨stop⟩]{⟨stepcontents⟩}
```

where ⟨`n`⟩ is the number of steps. The additional optional argument ⟨`stop`⟩ gives the code (default: `\stopAdvancing`) which stops the animation. (`\movie` accepts the same first optional argument as `\multistep` but it was left out above.)

`\overlays` is another shorthand macro for executing several steps successively. In contrast to `\multistep`, it doesn't print things *after* each other, but *over* each other. The syntax is

```
\overlays[⟨activatefirst⟩]{⟨n⟩}{⟨stepcontents⟩}
```

where ⟨`n`⟩ is the number of steps. Inside ⟨`stepcontents`⟩, a counter `substep` can be evaluated which tells the number of the current instance.

`\restep`, `\rebstep`, `\reswitch`, `\redstep`, `\revstep`.

Frequently, it is desirable for two or more steps to appear at the same time, for instance to fill in arguments at several places in a formula at once (see example **??**).

`\restep{⟨stepcontents⟩}` is identical with `\step{⟨stepcontents⟩}`, but is activated at the same time as the previous occurrence of `\step`.

`\rebstep`, `\reswitch`, `\redstep`, and `\revstep` do the same for `\bstep`, `\switch`, `\dstep`, and `\vstep`.

3.8 Optional arguments of `\step`

Sometimes, letting two `\step`s appear at the same time (with `\restep`) is not the only desirable modification of the order in which `\step`s appear. `\step`, `\bstep` and `\switch` take two optional arguments for influencing the mode of activation, like this:

`\step[⟨activatefirst⟩][⟨whenactive⟩]{⟨stepcontents⟩}`.

Both ⟨`activatefirst`⟩ and ⟨`whenactive`⟩ should be conditions in the syntax of the `\ifthenelse` command (see the documentation of the `ifthen` package for details).

⟨`activatefirst`⟩ checks whether this `\step` is to be activated *for the first time*. The default value is `\value{step}=\value{stepcommand}` (see section 3.9 for a list of internal values). By using `\value{step}=`⟨n⟩, this `\step` can be forced to appear as the nth one. See example **??** for a demonstration of how this can be used to make `\step`s appear in arbitrary order.

⟨`whenactive`⟩ checks whether this `\step` is to be considered active *at all*. The default behaviour is to check whether this `\step` has been activated before (this is saved internally for every step). See example **??** for a demonstration of how this can be used to make `\step`s appear and disappear after a defined fashion.

If you know what you're doing...

Both optional arguments allow two syntctical forms:

1. enclosed in square brackets `[]` like explained above.

2. enclosed in braces `()`. In this case, ⟨`activatefirst`⟩ and ⟨`whenactive`⟩ are *not* treated as conditions in the sense of `\ifthenelse`, but as conditionals like those used internally by LaTeX. That means, ⟨`activatefirst`⟩ (when enclosed in braces) can contain arbitrary TeX code which then takes two arguments and expands to one of them, depending on whether the condition is fulfilled or not fulfilled. For instance, `\step[⟨activatefirst⟩]{⟨stepcontents⟩}` could be replaced by `\step(\ifthenelse{⟨activatefirst⟩}){⟨stepcontents⟩}`.

   See example **??** for a simple application of this syntax.

Internally, the default for the treatment of ⟨whenactive⟩ is (\if@first@TP@true), where \if@first@TP@true is an internal condition checking whether this \step has been activated before.

3.9 Finding out what's going on

Inside ⟨settings⟩ and ⟨contents⟩, you can refer to the following internal state variables which provide information about the current state of the process executed by \stepwise:

counter: firststep The number from which to start counting steps (see counter step below). Is 0 by default and 1 for starred versions (section 3.4) of \stepwise. You can set this in ⟨settings⟩ for special effects (see example ??).

counter: totalsteps The total number of \step commands occurring in ⟨contents⟩.

counter: step The number of the current iteration, i. e. the number of the current page in the sequence of pages produced by \stepwise. Runs from \value{firststep} to \value{totalsteps}.

counter: stepcommand The number of the \step command currently being executed.

boolean: firstactivation true if this \step is active for the first time, false otherwise.

boolean: active true if this \step is currently active, false otherwise.

stepcommand, firstactivation, and active are useful only inside ⟨stepcontents⟩.

3.10 \afterstep

It might be neccessary to set some parameters which affect the appearance of the *page* (like page transitions) inside ⟨stepcontents⟩. However, the \step commands are usually placed deeply inside some structure, so that all *local* settings are likely to be undone by groups closing before the page is completed.

\afterstep{⟨settings⟩} puts ⟨settings⟩ right before the end of the page, after the current step is performed.

Things to pay attention to

1. There can be only one effective value for ⟨settings⟩. Every occurrence of \afterstep overwrites this value globally.

2. \afterstep will *not* be executed in ⟨stepcontents⟩ if the corresponding \step is not active, even if ⟨stepcontents⟩ is displayed owing to a redefinition of \hidestepcontents, like in example ??.

3. As ⟨settings⟩ is put immediately before the page break, there is no means of restoring the original value of whatever has been set. So if you set something via \afterstep and want it to be reset in some later step, you have to reset it explicitly with another call of \afterstep.

4 Page transitions and automatic advancing

4.1 Page transitions

I am indepted to Marc van Dongen for allowing me to include a suite of commands written by him and posted to the PPower4 mailing list which set page transitions (using hyperrefs \hypersetup).

These commands work only if the hyperref package is loaded.

The following page transition commands are defined:

h \pageTransitionSplitHO Split Horizontally to the outside.

h \pageTransitionSplitHI Split Horizontally to the inside.

h \pageTransitionSplitVO Split Vertically to the outside.

h \pageTransitionSplitVI Split Vertically to the inside.

h \pageTransitionBlindsH Horizontal Blinds.

h \pageTransitionBlindsV Vertical Blinds.

h \pageTransitionBoxO Growing Box.

h \pageTransitionBoxI Shrinking Box.

h \pageTransitionWipe{⟨angle⟩}

Wipe from one edge of the page to the facing edge.

⟨angle⟩ is a number between 0 and 360 which specifies the direction (in degrees) in which to wipe.

Apparently, only the values 0, 90, 180, 270 are supported.

h \pageTransitionDissolve Dissolve.

h \pageTransitionGlitter{⟨angle⟩}

Glitter from one edge of the page to the facing edge.

⟨angle⟩ is a number between 0 and 360 which specifies the direction (in degrees) in which to glitter.

Apparently, only the values 0, 270, 315 are supported.

h `\pageTransitionReplace` Simple Replace (the default).

Things to pay attention to

1. The setting of the page transition is a property of the *page*, i. e. whatever page transition is in effect when a page break occurs, will be assigned to the corresponding pdf page.

2. The setting of the page transition is undone when a group ends.

   Make sure no LaTeX environment is ended between a `\pageTransition` setting and the next page break. In particular, in ⟨`stepcontents`⟩, `\afterstep` should be used (see example **??**).

3. Setting page transitions works well with `\pause`. Here, `\pause` acts as a page break, i. e. a different page transition can be set before every occurrence of `\pause`.

4.2 Automatic advancing of pages

If you have loaded a sufficiently new version of the `hyperref` package (which allows to set `pdfpageduration`), then the following command is defined which enables automatic advancing of `pdf` pages.

h `\pageDuration{`⟨`dur`⟩`}` causes pages to be advanced automatically every ⟨`dur`⟩ seconds. ⟨`dur`⟩ should be a non-negative fixed-point number.

Depending on the `pdf` viewer, this will happen only in full-screen mode.

See example **??** for a demonstration of this effect.

The same restrictions as for **page transitions** apply. In particular, the page duration setting is undone by the end of a group, i. e. it is useless to set the page duration if a LaTeX environment ends before the next page break.

There is no 'neutral' value for ⟨`dur`⟩ (0 means advance as fast as possible). You can make automatic advancing stop by calling `\pageDuration{}`. `texpower` offers the custom command

h `\stopAdvancing`

to do this.

5 Color management, color emphasis and highlighting

TeXPower tries to find out whether you are making a colored document. This is assumed if

- the `color` package has been loaded before the `texpower` package or
- a color-related option (see sections 5.3 and 5.6) is given to the `texpower` package (in this case, the `color` package is loaded automatically).

If this is the case, TEXPower installs an extensive color management scheme on top of the kernel of the color package.

In the following, some new concepts established by this management scheme are explained. Sections 5.3 and 5.6 list options for color activation, section 5.7 lists some new highlighting commands, and section 5.8 gives the names and meaning of TEXPower's predefined colors.

Note that parts of the kernel of the color package are overloaded for special purposes (getting driver-independent representations of defined colors to be used by `\colorbetween` (5.5), for instance), so it is recommended to execute color definition commands like `\definecolor` *after* the `texpower` package has been loaded (see also the next section on `\defineTPcolor`).

5.1 Standard colors

TEXPower maintains a list of **standard colors** which are recognized and handled by TEXPower's color management. Some commands like `\dimcolors` (see section 5.4) affect *all* standard colors. There are some predefined colors which are in this list from the outset (see section 5.8).

If colors defined by the user are to be recognized by TEXPower, they have to be included in this list. The easiest way is to use the following command for defining them.

`\defineTPcolor{⟨name⟩}{⟨model⟩}{⟨def⟩}` acts like `\definecolor` from the `color` package, but the color ⟨name⟩ is also added to the list of standard colors.

If you want to make a color a standard color which is defined elsewhere (by a document class, say), you can simply add it to the list of standard colors with the command `\addTPcolor{⟨name⟩}`.

5.2 Color sets

Every standard color may be defined in one or several **color sets**. There are two fundamentally different types of color set:

The current color set. This contains the current definition of every standard color which is actually used at the moment. Every standard color should be defined at least in the current color set. The current color set is not distinguished by a special name.

Named color sets. These are 'containers' for a full set of color definitions (for the standard colors) which can be activated by respective commands (see below). The color sets are distinguished by their names. Color definitions in a named color set are not currently available, they have to be made available by activating the named color set.

There are four predefined color sets named `whitebg`, `lightbg`, `darkbg`, `blackbg`, each of which contains a full set of (predefined) standard colors customized for a white, light, dark, black background color, respectively.

There are the following commands for manipulating color sets:

`\usecolorset{⟨name⟩}` Make the color set named ⟨name⟩ the current color set. *All standard colors in the current color set which are also in color set ⟨name⟩ are overwritten.*

The standard color `textcolor` is set automatically after activating color set ⟨name⟩.

`\dumpcolorset{⟨name⟩}` Copy the definitions of *all* standard colors in the current color set into color set named ⟨name⟩. All standard colors in color set ⟨name⟩ will be overwritten.

Using `\defineTPcolor{⟨name⟩}` or `\definecolor{⟨name⟩}` will define the color ⟨name⟩ in the *current* color set. To define a color in color set ⟨cset⟩, use `\defineTPcolor[⟨cset⟩]{⟨name⟩}`.

Things to pay attention to

1. Color sets are not really 'TEX objects', but are distinguished by color name suffixes. This means, a color named `foo` is automatically in the current color set. Executing `\defineTPcolor[⟨cset⟩]{foo}` means executing `\definecolor` for a specific color the name of which is a combination of `foo` and ⟨cset⟩.

   Consequently, `\usecolorset` and `\dumpcolorset` do not copy color sets as composite objects, but simply all colors the names of which are generated from the list of standard colors.

2. The command `\usecolorset{⟨name⟩}` overwrites only those colors which have been defined in color set ⟨name⟩. If a standard color is defined in the current color set, but not in color set ⟨name⟩, it is preserved (but if `\dumpcolorset{⟨name⟩}` is executed later, then it will also be copied back into the color set ⟨name⟩).

5.3 Color Background Options

For activating the predefined color sets, there are shorthands `\whitebackground`, `\lightbackground`, `\darkbackground`, `\blackbackground` which execute `\usecolorset` and additionally set the background color to its current value.

When one of the following options is given, the respective command is executed automatically at the beginning of the document.

option: `whitebackground` **(default)** Set standard colors to match a white background color.

option: `lightbackground` Set standard colors to match a light (but not white) background color.

option: `darkbackground` Set standard colors to match a dark (but not black) background color.

option: `blackbackground` Set standard colors to match a black background color.

5.4 Color variants

In addition to color sets, TEXPower implements a concept of **color variant**. Currently, every color has three variants: **normal**, **dimmed**, and **enhanced**. The normal variant is what is usually seen, text written in the dimmed variant appears "faded into the background" and text written in the enhanced variant appears to "stick out".

When switching variants, for every color one of two cases can occur:

1. A **designated color** for this variant has been defined.

   For color ⟨`color`⟩ the designated name of the **dimmed** variant is `d`⟨`color`⟩, the designated name of the **enhanced** variant is `e`⟨`color`⟩.

   If a color by that name exists at the time the variant is switched to, then variant switching is executed by replacing color ⟨`color`⟩ with the designated color.

2. A **designated color** for this variant has not been defined.

   If a color by the designated name does not exist at the time a color variant is switched to, then variant switching is executed by **automatically** calculating the color variant from the original color.

   The method for calculation depends on the variant:

   dimmed. The dimmed variant is calculated by **interpolating** between `pagecolor` and the color to be dimmed, using the `\colorbetween` command (see 5.5).

   There is a command `\dimlevel` which contains the parameter ⟨`weight`⟩ given to `\colorbetween` (default: `0.7`). This default can be overridden by either redefining `\dimlevel` or giving an alternative ⟨`weight`⟩ as an optional argument to the color dimming command (see below).

   enhanced. The enhanced variant is calculated by **extrapolating** the color to be enhanced (relative to `pagecolor`).

   There is a command `\enhancelevel` which gives the **extent** of the extrapolation (default: `0.5`). The same holds for overriding this default as for `\dimlevel`.

The following commands switch color variants:

`\dimcolor[`⟨`level`⟩`]{`⟨`color`⟩`}` switches color ⟨`color`⟩ to the **dimmed** variant. If given, ⟨`level`⟩ replaces the value of `\dimlevel` in automatic calculation of the dimmed variant (see above).

`\dimcolors[`⟨`level`⟩`]` switches *all* standard colors to the **dimmed** variant. The optional argument ⟨`level`⟩ acts as for `\dimcolor`.

`\enhancecolor[`⟨`level`⟩`]{`⟨`color`⟩`}` switches color ⟨`color`⟩ to the **enhanced** variant. If given, ⟨`level`⟩ replaces the value of `\enhancelevel` in automatic calculation of the enhanced variant (see above).

`\enhancecolors[⟨level⟩]` switches *all* standard colors to the **enhanceed** variant. The optional argument ⟨`level`⟩ acts as for `\enhancecolor`.

Things to pay attention to

1. While automatic calculation of a **dimmed** color will almost always yield the desired result (interpolating between colors by calculating a weighted average is trivial), automatic calculation of an **enhanced** color by 'extrapolating' is tricky at best and will often lead to unsatisfactory results. This is because the idea of making a color 'stronger' is very hard to formulate numerically.

   The following effects of the current algorithm should be kept in mind:

   - if the background color is light, enhancing a color will make it darker;
   - if the background color is dark, enhancing a color will make it lighter;
   - sometimes, the numerical values describing an enhanced color have to be **bounded** to avoid exceeding the allowed range, diminishing the enhancing effect. For instance, if the background color is black and the color to be enhanced is a 'full-powered' yellow, there is no way of enhancing it by simple numeric calculation.

   As a conclusion, for best results it is recommended to provide custom `e` variants of colors to be enhanced. By default, TEXPower does not provide dedicated enhanced colors, but the file `tpsettings.cfg` contains complete sets of enhanced variants for the standard colors in the different color sets, which you can uncomment and experiment with as convenient.

2. Currently, switching to a different color variant is done by simply overwriting the current definitions of all standard colors. This means

   - there is no way of 'undimming' a color once it has been dimmed,
   - a dimmed color can not be enhanced and vice versa.

   Maybe this will be solved in a slightly more clever way in subsequent releases of TEXPower.

   Hence, it is recommended to

   - restrict the **scope** of a global variant switching command like `\dimcolors`, `\enhancecolors` or `\dstep` by enlcosing it into a LATEX group (like `{...}`) or
   - use `\dumpcolorset` before the command to save the current definitions of all colors, to be restored with `\usecolorset`.

     At the very beginning of a `\stepwise` command, TEXPower executes `\dumpcolorset{stwcolors}`, so you can restore the colors anywhere in the argument of `\stepwise` by saying `\usecolorset{stwcolors}`.

3. Some rudimentary attempts are made to keep track of which color is in what variant, to the effect that
   - a color which is not in the normal variant will neither be dimmed nor enhanced;
   - when `\usecolorset` overwrites a color with its normal variant, this is registered.

   Still, it is easy to get in trouble by mixing variant changes with color set changes (say, if not all standard colors are defined in a color set, or if a color set is dumped when not all colors are in normal variant), so it is recommended not to use or dump color sets when outside the normal variant (unless for special applications like undoing a variant change by `\usecolorset{stwcolors}`).

5.5 Miscellaneous color management commands

`\replacecolor[⟨tset⟩]{⟨tcolor⟩}[⟨sset⟩]{⟨scolor⟩}` makes ⟨tcolor⟩ have the same definition as ⟨scolor⟩ (if ⟨scolor⟩ is defined at all), where ⟨tcolor⟩ and ⟨scolor⟩ are color names as given in the first argument of `\definecolor`. If (one of) ⟨tset⟩ and ⟨sset⟩ are given, the respective color is taken from the respective color set, otherwise from the current color set.

If ⟨scolor⟩ is not defined (in color set ⟨sset⟩), ⟨tcolor⟩ is left alone.

`\colorbetween[⟨weight⟩]{⟨src1⟩}{⟨src2⟩}{⟨target⟩}` calculates a 'weighted average' between two colors. ⟨src1⟩ and ⟨src2⟩ are the names of the two colors. ⟨weight⟩ (default: 0.5) is a fixed-point number between 0 and 1 giving the 'weight' for the interpolation between ⟨src1⟩ and ⟨src2⟩. ⟨target⟩ is the name to be given to the resulting mixed color.

If ⟨weight⟩ is 1, then ⟨target⟩ will be identical to ⟨src1⟩ (up to color model conversions, see below), if ⟨weight⟩ is 0, then ⟨target⟩ will be identical to ⟨src2⟩, if ⟨weight⟩ is 0.5 (default), then ⟨target⟩ will be exactly in the middle between ⟨src1⟩ and ⟨src2⟩.

`\colorbetween` supports the following color models: `rgb`, `RGB`, `gray`, `cmyk`, `hsb`. If both colors are of the same model, the resulting color is also of the respective model. If ⟨src1⟩ and ⟨src2⟩ are from *different* models, then ⟨target⟩ will *always* be an `rgb` color. The only exception is the `hsb` color model: As I don't know how to convert `hsb` to `rgb`, mixing `hsb` with another color model will always raise an error.

`\mkfactor{⟨expr⟩}{⟨macroname⟩}` is a helper command for automatically generating the fixed point numbers between 0 and 1 which are employed by the color calculation commands. ⟨expr⟩ can be any expression which can stand behind `*` in expressions allowed by the `calc` package (for instance: `\value{counter}/\value{maxcounter}` or `\ratio` or whatever). ⟨macroname⟩

should be a valid macro name. ⟨expr⟩ is converted into a fixed-point representation which is then assigned to ⟨macroname⟩.

`\vanishcolors[⟨color⟩]` is similar to the color variant command `\dimcolors`, but instead of dimming colors, all standard colors are replaced by a single color given by the new command `\vanishcolor` (default: `pagecolor`). Hence, the result of calling `\vanishcolors` should be that all text vanishes, as it is written in the background color (this doesn't work with structured backgrounds, of course).

For getting a color different from the default `pagecolor`, you can either redefinine `\vanishcolor` or give an alternative ⟨color⟩ as an optional argument to `\vanishcolors`.

There is no dedicated command for making a single color vanish. To achieve this, use `\replacecolor{⟨color⟩}{\vanishcolor}`.

5.6 Color Emphasis and Highlighting

`texpower` offers some support for text emphasis and highlighting with colors (instead of, say, font changes). These features are enabled by the following options:

option: `coloremph` Make `\em` and `\emph` switch colors instead of fonts.

option: `colormath` Color all mathematical formulae.

option: `colorhighlight` Make new highlighting and emphasis commands defined by `texpower` use colors.

Things to pay attention to

1. You need the `color` package to use any of the color features.
2. To implement the options `coloremph` and `colormath`, it is neccessary to redefine some LaTeX internals. This can lead to problems and incompatibilities with other packages. Use with caution.
3. If the `colorhighlight` option is *not* given, new highlighting and emphasis commands defined by `texpower` are realized otherwise. Sometimes, however, there is no good alternative to colors, so different emphasis commands can become disabled or indistinguishable.
4. Because of font changes, emphasized or highlighted text can have different dimensions whether or not the options `coloremph`, `colormath`, and `colorhighlight` are set. Prepare for different line and page breaks when changing one of these options.
5. Color emphasis and highlighting makes use of the predefined standard colors described in section 5.8. See sections 5.1 to 5.3 for further information on standard colors, color sets, and customization.

5.7 New commands for emphasis and highlighting elements

Some things like setting the page or text color, making emphasised text or math colored are done automatically when the respective options are set. There are some additional new commands for creating emphasis and highlighting elements.

Concerning math:

`\origmath` When the `colormath` option is given, *everything* which appears in math mode is colored accordingly. Sometimes, however, math mode is used for something besides mathematical formulae. Some LaTeX commands which internally use math mode (like `tabular` or `\textsuperscript`) are redefined accordingly when the `colormath` option is given (this is a potential source of trouble; beware of problems...).

If you need to use math mode for something which is not to be colored (like a symbol for `itemize`), you can use the `\origmath` command which works exactly like `\ensuremath` but doesn't color its argument. If a nested use of math mode should occur in the argument of `\origmath`, it will again be colored.

Documenting TeX code:

`\code` Simple command for typesetting `code` (like shell commands).

`\macroname` For `\macro names`. Like `\code`, but with a `\` in front.

`\commandapp[⟨opt arg⟩]{⟨command⟩}{⟨arg⟩}` For TeX commands. ⟨`arg`⟩ stands for the command argument, ⟨`opt arg`⟩ for an optional argument.

`\carg` For ⟨`macro arguments`⟩.

Additional emphasis commands:

`\underl` Additional **emphasis** command. Can be used like `\emph`. Defaults to **bold face** if the `colorhighlight` option is not given.

`\concept` Additional **emphasis** command, especially for new concepts. Can be augmented by things like automatic index entry creation. Also defaults to **bold face** if the `colorhighlight` option is not given.

`\inactive` Additional emphasis command, this time for 'de-emphasising'. There is no sensible default if the `colorhighlight` option is not given, as base LaTeX doesn't offer an appropriate font. In this case, `\inactive` defaults to `\monochromeinactive`, which does nothing.

You can (re-)define `\monochromeinactive` to provide some sensible behaviour in the absence of colors, for instance striking out if you're using the `soul` package.

Color Highlighting:

`\present` Highlighting command which puts its argument into a box with colored background. Defaults to an `\fbox` if the `colorhighlight` option is not given.

See section 3.6 for some further highlighting commands.

5.8 Predefined standard colors

In previous subsections, it has been mentioned that TEXPower predefines some standard colors which have appropriate values in the predefined color sets `whitebg`, `lightbg`, `darkbg`, and `blackbg` (see sections 5.1 to 5.3 for further information on standard colors, color sets, and customization).

color: `pagecolor` Background color of the page. Is set automatically at the beginning of the document if color management is active.

color: `textcolor` Color of normal text. Is set automatically at the beginning of the document if color management is active.

color: `emcolor` Color used for *emphasis* if the `coloremph` option is set.

color: `altemcolor` Color used *for* double *emphasis* if the `coloremph` option is set.

color: `mathcolor` Color used for math $a^2 + b^2 = c^2$ if the `colormath` option is set.

color: `codecolor` Color used by the `\code` command if the `colorhighlight` option is set.

color: `underlcolor` Color used by the `\underl` **command** if the `colorhighlight` option is set.

color: `conceptcolor` Color used by the `\concept` **command** if the `colorhighlight` option is set.

color: `inactivecolor` Color used by the `\inactive` command if the `colorhighlight` option is set.

color: `presentcolor` Color used as background color by the `\present` command if the `colorhighlight` option is set.

color: `highlightcolor` Color used as background color by the `\highlightboxed` and `\highlighttext` commands (see section 3.6) if the `colorhighlight` option is set.

6 Structured page backgrounds and panels

6.1 Structured page backgrounds

`\backgroundstyle[⟨options⟩]{⟨style⟩}` is the central command for structured page backgrounds. It works like `\pagestyle` and other commands of this type. This means ⟨`style`⟩ is a symbolic name specifying the general method by which the page background is constructed.

The detailed construction is influenced by parameters which can be set in ⟨`options`⟩. If given, the optional parameter ⟨`options`⟩ should contain a list of settings in "keyval" manner. The keyval method is based on associating a symbolic name with every parameter. ⟨`options`⟩ is then a comma-separated list of parameter settings of the form ⟨`name`⟩=⟨`value`⟩, where ⟨`name`⟩ is the symbolic name of the parameter to be set and ⟨`value`⟩ is the value it is to be set to.

Not every ⟨`style`⟩ evaluates every parameter. In the following, a description of all styles, together with lists of the parameters employed, is given. It is followed by a list of all parameters. Note that some parameter names internally access the same parameter. For instance, parameters `startcolor` and `startcolordef` both set the start color of a color gradient. In case of conflict, the last setting in the list ⟨`options`⟩ will prevail. It is noted in the list of parameters which other parameters are overwritten.

⟨`style`⟩ may have one of the following values:

Style: `none` No background. This means the page background is whatever it would be if `\backgroundstyle` wasn't used at all (for instance, a plain area of color pagecolor if one of the color options has been given).

Parameters used: none.

Style: `plain` Plain background. This means the page background is whatever it would be if `\backgroundstyle` wasn't used at all (as for no background). In addition to background style `none`, the background style `plain` does produce panel backgrounds. The colors and dimensions of a `top panel`, `bottom panel`, `left panel`, and `right panel` can be specified.

Parameters used: `hpanels`, `autopanels`, `toppanelcolor`, `bottompanelcolor`, `leftpanelcolor`, `rightpanelcolor`, `toppanelcolordef`, `bottompanelcolordef`, `leftpanelcolordef`, `rightpanelcolordef`, `toppanelheight`, `bottompanelheight`, `leftpanelwidth`, `rightpanelwidth`.

Style: `vgradient` Vertical gradient. The page background is constructed using the `\vgradrule` command. In addition to the usual parameters of gradient rules, the vgradient background style allows to leave space for headers, footers, or panels. The colors and dimensions of a `top panel`, `bottom panel`, `left panel`, and `right panel` can be specified. The gradient rule fills the rectangular space left between the specified panels.

Parameters used: `stripes`, `firstgradprogression`, `startcolor`, `startcolordef`, `endcolor`, `endcolordef` in addition to the parameters used for style `plain`.

Style: `hgradient` Horizontal gradient. The page background is constructed using the `\hgradrule` command. See the description of `\vgradient` concerning panels.

Parameters used: See list for style `vgradient`.

Style: `doublevgradient` Double vertical gradient. The page background is constructed using the `\dblvgradrule` command. See the description of `\vgradient` concerning panels.

Parameters used: `gradmidpoint`, `secondgradprogression`, `midcolor`, `midcolordef` in addition to the parameters used for style `vgradient` (and `plain`).

Style: `doublehgradient` Double horizontal gradient. The page background is constructed using the `\dblhgradrule` command. See the description of `\vgradient` concerning panels.

Parameters used: See list for `doublevgradient`.

Now, a list of all parameters and their meaning. In the following,

- ⟨`n`⟩ denotes a (calc expression for a) nonnegative integer
- ⟨`i`⟩ denotes a (calc expression for an) integer
- ⟨`r`⟩ denotes a fixed-point number
- ⟨`l`⟩ denotes a (calc expression for a) length
- ⟨`c`⟩ denotes the name of a defined color
- ⟨`cm`⟩ denotes a valid color model name (in the sense of the color package)
- ⟨`cd`⟩ denotes a valid color definition (in the sense of the color package) wrt a given ⟨`cm`⟩ parameter
- ⟨`t`⟩ denotes a 'truth value' in the sense of the ifthen package: either true or false. As usual for keyval, if =⟨`t`⟩ is omitted, the default true is assumed.

Option: `stripes=`⟨`n`⟩ Set the ⟨`stripes`⟩ parameter of gradient rules to ⟨`n`⟩.
Default: `\bgndstripes`.
Used by: `vgradient`, `hgradient`, `doublevgradient`, `doublehgradient`.

Option: `gradmidpoint=`⟨`r`⟩ Set the ⟨`midpoint`⟩ parameter of double gradient rules to ⟨`r`⟩.
Default: `\bgndgradmidpoint`
Used by: doublevgradient, doublehgradient

Option: `firstgradprogression=⟨i⟩` Set the first gradient progression of gradient rules to ⟨i⟩.
Default: `\bgndfirstgradprogression`
Used by: vgradient, hgradient, doublevgradient, doublehgradient

Option: `secondgradprogression=⟨i⟩` Set the second gradient progression of double gradient rules to ⟨i⟩.
Default: `\bgndsecondgradprogression`
Used by: doublevgradient, doublehgradient

Option: `startcolor=⟨c⟩` Set the ⟨`startcolor`⟩ parameter of gradient rules to ⟨c⟩.
Default: If neither startcolor nor startcolordef is given, the color bgndstartcolor is used as startcolor.
Used by: vgradient, hgradient, doublevgradient, doublehgradient
Overwrites: startcolordef

Option: `startcolordef={⟨cm⟩}{⟨cd⟩}` Set the ⟨`startcolor`⟩ parameter of gradient rules to color foo, which is obtained by `\definecolor{foo}{⟨cm⟩}{⟨cd⟩}`. Note that the two pairs of curly braces are mandatory.
Default: If neither startcolor nor startcolordef is given, the color bgndstartcolor is used as startcolor.
Used by: vgradient, hgradient, doublevgradient, doublehgradient
Overwrites: startcolor

Option: `endcolor=⟨c⟩` Set the ⟨`endcolor`⟩ parameter of gradient rules to ⟨c⟩.
Default: If neither endcolor nor endcolordef is given, the color bgndendcolor is used as endcolor.
Used by: vgradient, hgradient, doublevgradient, doublehgradient
Overwrites: endcolordef

Option: `endcolordef={⟨cm⟩}{⟨cd⟩}` Set the ⟨`endcolor`⟩ parameter of gradient rules to color foo, which is obtained by `\definecolor{foo}{⟨cm⟩}{⟨cd⟩}`. Note that the two pairs of curly braces are mandatory.
Default: If neither endcolor nor endcolordef is given, the color bgndendcolor is used as endcolor.
Used by: vgradient, hgradient, doublevgradient, doublehgradient
Overwrites: endcolor

Option: `midcolor=⟨c⟩` Set the ⟨`midcolor`⟩ parameter of double gradient rules to ⟨c⟩.
Default: If neither midcolor nor midcolordef is given, the color bgndmidcolor is used as midcolor.
Used by: doublevgradient, doublehgradient
Overwrites: midcolordef

Option: `midcolordef={⟨cm⟩}{⟨cd⟩}` Set the ⟨`midcolor`⟩ parameter of double gradient rules to color foo, which is obtained by `\definecolor{foo}{⟨cm⟩}{⟨cd⟩}`. Note that the two pairs of curly braces are mandatory.
Default: If neither midcolor nor midcolordef is given, the color bgndmidcolor is used as midcolor.
Used by: doublevgradient, doublehgradient
Overwrites: midcolor

Option: `hpanels=⟨t⟩` Specifies the 'direction' of panels produced. hpanels=true means the top and bottom panel span the full width of the screen. In the space left in the middle, the left panel, the background itself, and the right panel are displayed. hpanels=false means the left and right panel span the full height of the screen. In the space left in the middle, the top panel, the background itself, and the bottom panel are displayed.
Default: hpanels=true is the default for plain, hgradient and doublehgradient. hpanels=false is the default for vgradient and doublevgradient.
Used by: plain, vgradient, hgradient, doublevgradient, doublehgradient

Option: `autopanels=⟨t⟩` Specifies whether the default values of the parameters toppanelheight, bottompanelheight, leftpanelwidth, rightpanelwidth should be calculated automatically from the contents of declared panels. The automatism used is analogous to that of `\DeclarePanel*`. Note that for panel arrangement, both the width and the height of all declared panels are overwritten. If you don't want this, calculate the panel parameters yourself and set autopanels=false. In this case, the current panel dimensions of declared panels are used as defaults for toppanelheight, bottompanelheight, leftpanelwidth, rightpanelwidth.
Default: true.
Used by: plain, vgradient, hgradient, doublevgradient, doublehgradient

Option: `⟨pos⟩panelheight=⟨l⟩` Set the height/width of the space left for the top / bottom / left / right panel to ⟨`l`⟩. Note that the remaining dimensions of panels, for instance the width of the top panel, are always calculated automatically, depending on the setting of the hpanels parameter.
Default: If a respective panel has been defined using `\DeclarePanel`, the default used depends on the setting of the autopanels parameter. If autopanels=true, the correct dimension is calculated from the contents of the panel. The respective one of `\toppanelheight`, `\bottompanelheight`, `\leftpanelwidth`, `\rightpanelwidth` is overwritten with the result. If autopanels=false, then the respective setting of `\toppanelheight`, `\bottompanelheight`, `\leftpanelwidth`, `\rightpanelwidth` is taken as the default. If a panel has not been declared, the appropriate one of `\bgndtoppanelheight`, `\bgndbottompanelheight`, `\bgndleftpanelwidth`, `\bgndrightpanelwidth` is used as default.
Used by: plain, vgradient, hgradient, doublevgradient, doublehgradient

Option: ⟨pos⟩panelcolor=⟨c⟩ Set the color of the space left for the top / bottom / left / right panel to ⟨c⟩.

Default: The standard colors toppanelcolor, bottompanelcolor, leftpanelcolor, rightpanelcolor are used as defaults.

Used by: plain, vgradient, hgradient, doublevgradient, doublehgradient

Overwrites: toppanelcolordef bottompanelcolordef leftpanelcolordef rightpanelcolordef

Option: ⟨pos⟩panelcolordef={⟨cm⟩}{⟨cd⟩} Set the color of the space left for the top / bottom / left / right panel to color foo, which is obtained by `\definecolor{foo}{`⟨cm⟩`}{`⟨cd⟩`}`. Note that the two pairs of curly braces are mandatory.

Default: See the description of top/bottom/left/rightpanelcolor.

Used by: plain, vgradient, hgradient, doublevgradient, doublehgradient

Overwrites: toppanelcolor bottompanelcolor leftpanelcolor rightpanelcolor

6.2 Panel-specific user level commands

If you're using a package that has it own panel (as pdfscreen) don't even consider using the following.

`\DeclarePanel[`⟨name⟩`]{`⟨pos⟩`}{`⟨contents⟩`}` declares the contents ⟨contents⟩ of the panel at position ⟨pos⟩. Afterwards, on every page the panel contents are set in a parbox of dimensions and position specified by ⟨pos⟩panelwidth, ⟨pos⟩panelheight, `\panelmargin` and ⟨pos⟩panelshift for top and bottom panels and ⟨pos⟩panelraise for left and right panels. The parbox is constructed anew on every page, so all changes influencing panel contents or parameters (like a `\thepage` in the panel contents) are respected.

The panel contents are set in color ⟨pos⟩paneltextcolor. There is another standard color ⟨pos⟩panelcolor, which is however not activated by `\DeclarePanel` but by selecting an appropriate background style.

Note that `\backgroundstyle` must be called after the panel declaration.

Pages are constructed as follows: first the page background, then the panels, and then the page contents. *Hence, panels overwrite the background and the page contents overwrite the panels.* The user is supposed to make sure themselves that there is enough space left on the page for the panels (document class specific settings). The panel declaration is global. A panel can be 'undeclared' by using `\DeclarePanel{`⟨pos⟩`}{}`.

If the optional argument ⟨name⟩ is given, the panel contents and (calculated) size will also be stored under the given name, to be restored later with `\restorepanels`. This is nice for switching between different sets of panels.

For an example look at the files `simplepanel.tex` and `panelexample.tex`. A very simple example follows:

```
\DeclarePanel{left}{%
  \textsf{Your Name}
```

```
\vfill

\button{\Acrobatmenu{PrevPage}}{Back}
\button{\Acrobatmenu{NextPage}}{Next} }
```

There is a starred version which will (try to) automatically calculate the 'flexible' dimension of each panel. For top and bottom panels this is the height, for left and right panels this is the width. Make sure the panel contents are 'valid' at the time `\DeclarePanel*` is called so the calculation can be carried out in a meaningful way.

While the automatic calculation of the height of top and bottom panels is trivial (using `\settoheight`), there is a sophisticated procedure for calculating a 'good' width for the parbox containing the panel. Owing to limitations set by TeX, there are certain limits to the sophistication of the procedure.

For instance, any 'whatsits' (specials (like color changes), file accesses (like `\label`), or hyper anchors) or rules which are inserted directly in the vertical list of the parbox 'block' the analysis, so the procedure can't 'see' past them (starting at the bottom of the box) when analysing the contents of the parbox.

The user should make sure such items are set in horizontal mode (by using `\leavevmode` or enclosing stuff in boxes). Furthermore, only overfull and underfull hboxes which occur while setting the parbox are considered when judging which width is 'best'. This will reliably make the width large enough to contain 'wide' objects like tabulars, logos and buttons, but might not give optimal results for justified text. vboxes occurring directly in the vbox are ignored.

Note further that hboxes with fixed width (made by `\hbox` to...) which occur directly in the vbox may disturb the procedure, because the fixed width cannot be recovered. These hboxes will be reformatted with the width of the vbox, generating an extremely large badness, unsettling the calculation of maximum badness. To avoid this such hboxes should be either contained in a vbox or set in horizontal mode with appropriate glue at the end.

6.3 Navigation buttons

The following provides only the very basics for navigation buttons. If you're using a package that has it's own naviagtion buttons (as pdfscreen) don't even consider using the following.

`\button{⟨navcommand⟩}{⟨text⟩}` creates a button labelled ⟨`text`⟩ which executes ⟨`navcommand`⟩ when pressed. The command takes four optional arguments (left out above): ⟨`width`⟩, ⟨`height`⟩, ⟨`depth`⟩ and ⟨`alignment`⟩ in that order. ⟨`navcommand`⟩ can be for instance `\Acrobatmenu{⟨command⟩}` or `\hyperlink{⟨target⟩}` (note that ⟨`navcommand`⟩ should take one (more) argument specifying the sensitive area which is provided by `\button`). If given, the optional parameters ⟨`width`⟩, ⟨`height`⟩, and ⟨`depth`⟩ give the width, height and depth, respectively, of the framed area comprising the button (excluding the shadow, but including the frame). Default are the 'real' width, height

and depth, respectively, of ⟨text⟩, plus allowance for the frame. If given, the optional parameter ⟨alignment⟩ (one of l,c,r) gives the alignment of ⟨text⟩ inside the button box (makes sense only if ⟨width⟩ is given).

The button appearence is defined by some configurable button parameters:

`\buttonsep` Space between button label and border. (Default: `\fboxsep`)

`\buttonrule` Width of button frame. (Default: `0pt`)

`\buttonshadowhshift` Horizontal displacement of button shadow. (Default: 0.3`\fboxsep`)

`\buttonshadowvshift` Vertical displacement of button shadow. (Default: 0.3`\fboxsep`)

A list of predefined buttons follows:

`\backpagebutton[`⟨width⟩`]` Last subpage of previous page.

`\backstepbutton[`⟨width⟩`]` Previous step.

`\gobackbutton[`⟨width⟩`]` 'Undo action' (go back to whatever was before last action).

`\nextstepbutton[`⟨width⟩`]` Next step.

`\nextpagebutton[`⟨width⟩`]` First subpage of next page.

`\nextfullpagebutton[`⟨width⟩`]` Last subpage of next page.

`\fullscreenbutton[`⟨width⟩`]` Toggle fullscreen mode.

Index