The TEXPower bundle
(Preliminary) Documentation[*]

Stephan Lehmke
mailto:Stephan.Lehmke@cs.uni-dortmund.de

July 3, 2000

Contents

[*]Documentation v0.0.6b of June 23, 2000 for TEXPower v0.0.8f of June 27, 2000 (pre-alpha).

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 this *pre-alpha* release, the documentation does not yet provide a full manual. For finding out how to achieve special effects (as shown in the Examples), please look at the comments inside the `.tex` files in the `doc` directory and inside the file `texpower.sty` to find out what's going on.

For your own first steps with TEXPower, the simple demo file `doc/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.

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

The other, more sophisticated examples in the `doc` directory are to 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.

For the first *alpha* release, this documentation will be completed. For the first *beta* release, when the code is a little more stable, the `texpower` package will be made into a properly documented `.dtx` file.

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.

There are no specific restrictions as to which way the `pdf` format is produced. All documents from the `doc` directory have been tested with pdfLaTeX and standard LaTeX, using `dvips` and **Adobe Acrobat® Distiller** 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.

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.

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 two options for setting the base font of the document to one that is 'bolder' than the default computer modern roman (cmr). This might be neccessary if readability is reduced by using e. g. colored backgrounds.

Note that the support offered by `texpower` is rather primitive. If you're using a document class or package which offers more sophisticated support for this kind of thing, use that by all means.

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

option: `sans` Make the sans serif font the basic text font. By default, this is computer modern sans serif (cmss). If you are using the package `pslatex`, this is Helvetica. For other packages changing the complete set of text fonts, this may be a different font.

option: `slifonts` Change the the text and math fonts to use the "slifonts" collection by L. Lamport. The main text font is then lcmss.

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, or if the `color` package was loaded before `texpower`.

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 two 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 of e. g. standard colors (see section 5). See the comments inside the file `tpsettings.cfg` which is part of the TEXPower bundle for instructions.

1.6 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 3.5).

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

1.7 What else is part of the TEXPower bundle?

Besides the package `texpower` (which is described here), there is one more package and one document class in the TEXPower bundle which so far have no documentation of their own. They are quite trivial at the moment and will be described in this section until they are turned into `dtx` files producing their own documentation.

There is a `doc` directory in the TEXPower bundle which contains (besides this documentation) some examples and demos for TEXPower. 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: `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: `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.

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.

2 The `\pause` command

$\boxed{\texttt{\textbackslash pause}}$ 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`.

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

3 The `\stepwise` command

$\boxed{\texttt{\textbackslash stepwise\{⟨contents⟩\}}}$ 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 $\boxed{\texttt{\textbackslash step\{⟨stepcontents⟩\}}}$, 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 Examples).
4. As ⟨`contents`⟩ is read as a macro argument, constructs involving **catcode** changes (like `\verb` or language switches) won't work in ⟨`contents`⟩. Using a suggestion by Ross Moore, I hope to remedy this until the *alpha* release.
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).

   Labels and hyperlinks work sort of (giving a lot of warnings though).

   I will try to remedy remaining problems until the first *alpha* release.

3.1 `\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 3.5 for more sophisticated (albeit more fragile) variants.

$\boxed{\texttt{\textbackslash boxedsteps}}$ makes `\step` create a blank box the size of ⟨`stepcontents`⟩ when inactive and put ⟨`stepcontents`⟩ into a box when active.

$\boxed{\texttt{\textbackslash nonboxedsteps}}$ 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:

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

3.2 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 caused by interactions between different ways vertical spacing is added to the page. Hopefully, problems caused this way can be reduced until the first *alpha* release.

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.

3.3 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 1.7), 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.

3.4 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.

3.5 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 Examples, 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 5.1). 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 5.2). 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 5.2).

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

3.6 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 3.1). `\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 5.2) 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 5.1) 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.

`\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 1.4).

`\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`.

3.7 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 3.8 for a list of internal values). By using `\value{step}=`⟨n⟩, this `\step` can be forced to appear as the nth one. See example 1.5 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 1.8 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 1.6 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.

3.8 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 3.3) of `\stepwise`. You can set this in ⟨`settings`⟩ for special effects (see example 1.6).

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`⟩.

3.9 `\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 1.7.
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 `hyperref`s `\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. Unfortunately, the **scope** of the setting of the page transition achieved by the `\pageTransition...` commands (via `\hypersetup`) seems to be dependent on the driver used...

   This means, depending on the driver employed (i. e. `pdfmark` or `pdftex`), the setting of the page transition will be undone when a group ends, or it will be valid until the end of the document if you don't explicitly set another page transition.

   This means, if you want your documents to work with different drivers, you have to take *both* behaviours into account:

   - 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 1.2).
   - If you want the defalt behaviour (Replace) back, set it explicitly with `\pageTransitionReplace`.

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 1.8 for a demonstration of this effect.

The same restrictions as for **page transitions** apply. In particular, the page duration setting is undone (for some drivers) 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 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.

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.

The effect of the options `whitebackground`, `lightbackground`, `darkbackground`, and `blackbackground` can also be achieved by the commands `\whitebackground`, `\lightbackground`, `\darkbackground`, and `\blackbackground`, respectively.

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 is controlled by a set of defined colors for different emphasis and highlighting elements (described in section 5.2). All these colors are redefined when one of the options `whitebackground`, `lightbackground`, `darkbackground`, and `blackbackground` or one of the commands `\whitebackground`, `\lightbackground`, `\darkbackground`, and `\blackbackground` is given.

   These definitions represent standard settings; the colors can be redefined as convenient. See the documentation of the `color` package for details.

5.1 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.

Helpers:

`\replacecolor{⟨newname⟩}{⟨origname⟩}` For some presentation effects using colors, it is neccessary to redefine standard colors like `mathcolor`. The `color` package only offers the `\definecolor` command for doing this, which needs a 'raw' color definition, like **RGB** values. It can not be used to redefine `mathcolor` to be the same as `inactivecolor`, for instance.

`\replacecolor` will make `⟨newname⟩` have the same definition as `⟨origname⟩`, where `⟨newname⟩` and `⟨origname⟩` are color names as given in the first argument of `\definecolor`.

See example 1.7 for a presentation effect where this is used.

Changing color sets:

In addition to the standard colors for displaying 'normal' text, there exist 'dimmed' and 'enhanced' color sets especially for dynamic effects. Compare section 5.2.

`\dimcolors` Switch to **dimmed** color set.

`\enhancecolors` Switch to **enhanced** color set.

`\vanishcolors` Replace *all* standard colors by *one* color. This color is given by the command `\vanishcolor`, which defaults to `pagecolor` and can be redefined as convenient.

5.2 Defined colors for emphasis and highlighting elements

The following colors are used automatically by the text emphasis and highlighting commands:

color: `pagecolor` Background color of the page. Is set automatically if you give one of the options `colorhighlight`, `whitebackground`, `lightbackground`, `darkbackground`, or `blackbackground`.

color: `textcolor` Color of normal text. Is set automatically if you give one of the options `colorhighlight`, `whitebackground`, `lightbackground`, `darkbackground`, or `blackbackground`.

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 3.5) if the `colorhighlight` option is set.

Dimmed and enhanced color sets

For every one of the colors listed above, there exists a **dimmed** and an *enhanced* variant. The names of these are the same as the standard colors, preceded by `d` for the dimmed set and preceded by `e` for the enhanced set. For instance, the dimmed variant of `textcolor` is called `dtextcolor` and the enhanced variant of `textcolor` is called `etextcolor`.

The standard color set is replaced by the dimmed color set by the command `\dimcolors` and by the enhanced color set by the command `\enhancecolors`. Compare section 5.1.