The standalone Package

Martin Scharrer
martin@scharrer.me

CTAN: http://www.ctan.org/pkg/standalone

VC: https://bitbucket.org/martin_scharrer/standalone

Version v1.1a – 2012/05/20

Abstract

The `standalone` bundle allows users to easily place picture environments or other material in own source files and compile these on their own or as part of a main document. A special `standalone` class is provided for use with such files, which by default crops the resulting output file to the content. The `standalone` package enables the user to simply load the standalone files using `\input` inside a main document.

Contents

1 Introduction

Larger LaTeX documents can be split into multiple TeX files which are then included in a main document with `\include` for e.g. chapter files or `\input` for e.g. TeX-coded pictures. Keeping pictures in their own sub-files improves readability of the main file and simplifies the sharing of them between different documents. However, during the, sometimes lengthly, drawing/coding process it has benefits to be able to compile the pictures on their own. The compile process is much quicker and the resulting document only holds the picture which avoids constant page turning and zooming.

While it is possible to write a small 'main' file for each picture file, this method is a little cumbersome and clutters the directories with a lot of extra files. A second method is to place the 'main' components, i.e. a preamble, directly into the picture files and make the main document ignore this code sections.

The package `standalone` can be used in the main document to skip all extra preambles in included files. The main file must load all packages and settings required by the sub-files. Several package options are provided to collect the preambles of the sub-files automatically and execute them from the main file.

A `standalone` class is also provided to minimise the extra preamble code needed in this files. It's usage is optional, but simplifies and standardises how picture files are compiled standalone. The class uses by default the `preview` package to create an output file which only contains the picture with no extra margins, page numbers or anything else. A configuration file `standalone.cfg` read by the class allows the user to adjust settings and macros easily on a per directory base.

1.1 Quick instructions

Load the `standalone` *package* very early in the main document. Also all packages needed by all the sub-files must be loaded by the main document. Include your picture or other sub-files using `\input` or a similar macro as normal. In the sub-files use the `standalone` *class* with a normal `\documentclass` and load all packages needed for the particular file. Finally wrap the actual content of the sub-file in a `document` environment. Avoid empty lines at the begin or end of the document body.

When the sub-file is compiled on its own the `\documentclass` and `document` environment will be active as normal. The main file, however, will skip everything from the `\documentclass` till the `\begin{document}`. The (now fake) `document` environment is redefined to be a simple TeX-group. Any code lines after the `\end{document}` will be ignored. The real `document` environment of the main file will be unaffected and will work as normal.

The packages required by each sub-file can be transfered automatically to

the main document preamble using the options listed in section 3.2.

1.2 Dependencies

The `standalone` class and package require the `xkeyval` package. The packages `ifpdf`, `ifluatex` and `ifxetex` are loaded if available, otherwise some fall-back code is used. If enabled the class options varwidth, preview and beamer require the package or class of the same name.

The `standalone` package requires the `currfile` package (which in turn uses `filehook`) to track the correct file names of sub-files included using `\input`. For the compilation support for included standalone files the `gincltex` and `filemod` packages are also required.

To compile the documentation of `standalone` the `ydoc` bundle is required.

All of these packages are included in recent versions of the TeXLive or MikTeX distributions and are freely available on CTAN.

1.3 Bug reports, feature requests and other feedback

Bug reports, feature requests and other feedback about the `standalone` bundle can be sent to the author either by email to martin@scharrer-online.de or using the issue tracker for the bundle under . Bug reports should include the used version of `standalone` as well as the used LaTeX format (`pdflatex`, `latex`, `xelatex`, etc.) and distribution including its version. Usually a minimal example which recreate the issue is immensely helpful in analysing and solving any bug. Please look for existing related issue tickets first and check the FAQ/troubleshooting in section 2.8 first. Issues related to the preview class option should be compared with a direct use of the underlying `preview` package.

1.4 Version update and backwards compatibility

The default behaviour of v1.x of the `standalone` class is slightly different as the one of v0.x, but should result in the same output for the majority of standalone files. In previous versions the preview option was enabled by default, but since v1.0 the new, similar crop option is now used. This change should improve several use-cases, like avoiding the creation of a paragraph due to a trailing empty line and issues with TikZ patterns under XeLaTeX. However, paragraph breaks are now ignored by default, which should be no issue at all for picture and similar environments which are the main target of the `standalone` class. Additionally, the default border has been changed from the `preview` default of 0.50001bp to no border (0pt). Both of these settings can be changed back to the old default by adding `\standaloneconfig{preview,border=0.50001bp}` in the configuration file or explicitly stating these options as class options.

One true incompatibility between v0.x and v1.x is the load point of the class configuration file. In v0.x the configuration file was loaded after all options where processed in order to have all if-switches their final value. In v1.x the configuration file is now loaded directly before the given class options are processed. This allows to easily set default options for all standalone files. Code which relies on if-switches (like `\ifstandalone` and `\ifstandalonebeamer`) should be placed inside a `\AtEndOfClass{`⟨*code*⟩`}` macro. This change might require an update of personal configuration files.

1.5 Similar packages and classes

The following packages, libraries and/or classes target the same or similar applications as the bundle and are mentioned here for easy comparison, so that the user can decide which suits them best.

The `docmute` package is written for the same basic task as the `standalone` package. However, no sub-preamble processing other than the removal is support. It also doesn't provide a special class or configuration file.

The `subfile` package and class are written for the same application to allow subfiles to be compiled standalone. However, the `class` class will import the preamble from a given main file, while `standalone` is designed more for the opposite direction where the preamble of subfiles can be imported to the main document. Therefore a `standalone` file can be more easily included into several documents, like a paper (scientific publication), a corresponding presentation and then a thesis, while `subfile` is designed for a one-to-one relationship. At the time of the writing `subfile` is not part of TeXLive due to a missing license statement.

The `external` library of `tikz` allows to externalize `tikzpictures` from an main document. Its build feature is similar to the one provided by `standalone`. However, both work form different directions: `standalone` allows to include external `tikzpictures` to be included in a main file while ignoring the preamble while `external` writes them from the main file to temporary external files. The user must decide which workflow is better suited for him/her. Also `standalone` is working independently of `tikz` and supports other picture environments like `pstricks` or any other TEX material.

2 Usage of the `standalone` class

2.1 Basic usage

Creating a basic standalone is straight-forward: Create a normal LaTeX document which uses the `standalone` as document class. The preamble should load all required packages and libraries for the content. The content, usually a single picture environment like `tikzpicture`, is placed in the `document` body. Empty lines before and after the picture should be avoided. Also the `\begin{document}` and `\end{document}` should each stand on a source line of their own.

Listing 1: Basic use of the `standalone` class.

```
\documentclass{standalone}
\usepackage{somepackage}
\begin{document}
\begin{somepicture}
    \somedrawingcommands
\end{somepicture}
\end{document}
```

Such a file can be compiled as normal. The `standalone` class will crop the resulting output file (PDF or DVI/PS) to the content size plus a certain border. Page number and other header or footer material will be suppressed.

For pictures drawn with TikZ a dedicated tikz option is provided which loads the `tikz` package and also configures the `tikzpicture` environment to create a single cropped page. For PSTricks pictures an corresponding pstricks option is provided.

Listing 2: Basic use of the `standalone` class.

```
\documentclass[tikz]{standalone}
%\usetikzlibrary{calc}
\begin{document}
\begin{tikzpicture}
  \draw (0,0) rectangle (2,1) node [midway] {Example};
\end{tikzpicture}
% Further 'tikzpicture' environments are possible which will create further pages.
\end{document}
```

2.2 Class options

The `standalone` class provides the following options to adjust the processing and size of the content. These options are removed from the normal list of class options and not passed to any loaded packages or classes like it would usually occur. This is also done to avoid option conflicts with identical named

options of the underlying class.

All boolean options take either '`true`' or '`false`' as optional values. Otherwise, if the option is used without a value, '`true`' is used. If not mentioned otherwise all options set to '`false`' initially. Options might switch other options on or off. For example, mutual exclusive options will disable each other. The order of the option is obeyed and later options will prevail over earlier ones.

By default the crop option with border=0 is enabled. In versions prior to v1.0 the option preview was the default. This chance was deemed required and should not affect most documents. However, in some cases resetting the preview option might be required.

Certain class options can also be set inside the preamble or document body using `\standaloneconfig{`⟨*options*⟩`}`.

`class=`⟨*class name*⟩

Specifies the underlying class which is loaded by the `standalone` class. By default `article` is used, which should be suitable for standalone pictures. In certain cases it may be from benefit to use the same class than in the targeted main document. For the `beamer` class the special beamer option should be used instead.

`crop=true|false`

If enabled this option crops the content to its natural size plus a specified border. This is done by saving the content in a box register and resizing the page size relative to the box dimensions. This option is enabled by default (since v1.0). This option is mutually exclusive with the similar preview option and will therefore disable it. If both options are used the last one will be enabled and the other will be disabled. Also `float=false` will be set by `crop=true` in order to avoid issues with floating environments.

`preview=true|false`

If enabled this option loads the `preview` package with the tightpage option and wraps the content into a `preview` environment. This crops the content to its natural size plus a specified border. Issues with the `preview` options and TikZ shadings under XeLaTeX have been reported. In this cases the crop option should be used instead. Note that this option was enabled by default for versions before v1.0, but since then crop is enabled by default.

This option is mutually exclusive with the similar crop option and will therefore disable it. If both options are used the last one will be enabled and the other will be disabled. Also `float=false` will be set by `preview=true` in order to avoid issues with floating environments.

```
border=⟨length (all sides)⟩
border={⟨length (left/right)⟩ ⟨length (bottom/top)⟩}
border={⟨length (left)⟩ ⟨length (right)⟩ ⟨length (bottom)⟩ ⟨length (top)⟩}
```

This option allows to specify the border used by the preview and crop options. An alternative name of this option is margin. The border can either be given using a single value for all sides, separately for the horizontal and vertical borders or for all sides separately. Multiple values are separated by spaces, which require the whole value to be wrapped in braces. By default a border of `0pt` is set.

This option can be changed during the document using **`\standaloneconfig`** and will affect all following pages.

```
multi=true|false
multi={⟨environment name⟩, ...>}
```

By default the **`standalone`** class assume that the whole content is one block which should be shown on one single page. If this option is activated multiple pages are supported. Each page will be cropped to its content plus the selected border (as long either preview or crop are enabled). A set of environments which hold a single page must either be given as option value or declared using **`\standaloneenv`**{⟨*environment name*⟩, ...}. No typeset material should be used outside such environments. Note that this option is enabled automatically by **`\standaloneenv`** if either crop or preview is enabled. However, it needs to be set explicitly as class option if the ignorerest option is also set. If environment names are provided as option values the option is set to '**`true`**' and the environments are passed to **`\standaloneenv`** which is executed at the begin of the document environment, where all mentioned environments should be already defined.

```
ignorerest=true|false
```

This option is only meaningful when both multi and crop are enabled. Then it determines if all material which does not appear inside environments declared with **`\standalone`** should be ignored or not. This is done by boxing and discarding all outside material. Any code will be placed inside a group and therefore local settings made between environments will not affect later code. Code in the preamble is not affected. It is recommended to keep this option disabled and only use it if really required. It should be noted that which preview such material is always ignored while not affecting local settings. Therefore the ignorrest option can be seen as a compatibility setting to make crop act more like preview, if this is required by the user.

`varwidth=true|false`
`varwidth=`⟨*width*⟩

A trailing empty line between the content and `\end{document}` will normally create a paragraph which is `\linewidth` wide. This paragraph (or any other one) will enlarge the size of smaller pictures and display itself as a large right border. This option uses the `varwidth` package to wrap the content into a `varwidth` environment, which is based on `minipage`, but will always use the natural width of the content if it is smaller than the given maximum width. The resulting effect is that the created paragraph will not cause any additional width and that multiple paragraphs can be included as part of the content. The used maximal width (which is provided to the underlying `minipage` environment) is `\linewidth` by default, but can be set by provided a width as value to the option. Doing so will also switch the option on.

If the crop option is used the content is placed in restricted horizontal mode which ignores paragraph breaks. Using the varwidth option paragraph breaks are enabled again.

A drawback of this option is that the content will be limited to the given width, i.e. wider picture environment will be cropped to the width at the right side. In such cases either a larger width should be selected, the option be switch off, any paragraph breaks should be avoided (no trailing empty lines) or one of the specific picture options like tikz or pstricks should be used instead.

This option can be changed during the document using `\standaloneconfig` and will affect all content of the following pages.

`tikz=true|false`

This option declares that the content contains of one or more `tikzpicture` environments. This sets `multi=tikzpicture,varwidth=false` and loads the `tikz` package.

`pstricks=true|false`

This option declares that the content contains of one or more `pspicture` or `pspicture*` environments. This sets `multi=pspicture,varwidth=false` and loads the `pstricks` package. Because `pspicture*` uses `pspicture` internally it is also supported. Other environments which use it as well should also be supported, but might also declared explicitly using `\standaloneenv{`⟨*environment name*⟩`, ...}`.

`beamer=true|false`

If set to '`true`' this option enables a special `beamer` mode, where the normal cropping is disabled. Instead the content is shown on a blank beamer frame.

`float=true|false`

If this option is that to '`false`' (which is the default) any floats like `figure` and `table` environments are turned into non-floating environment. This is required for the options crop and preview to work, so these will set `float=false` when set to '`true`' itself. In general it is recommended to keep floating environments inside the main document and only place the content of them into standalone files. This also makes it simple to include the same content in different floats of different main documents.

If custom floats are defined using a package like `float` are not supported yet. Dependent on the way they define floats they might still work. For these `float=true` should be set as class options so that the normal definition of floats is preserved. Afterwards `\standaloneconfig{float=false}` can be used to disable floats while taking the changed float definition into account.

`convert={`⟨*conversion options*⟩`}`
`png={`⟨*conversion options*⟩`}`

These options allow to enable and configure the conversion feature. See section 2.6 for the full description.

2.3 Macros and environments

The following macros and environments can be used inside the preamble of `standalone` files. Further macros are listed in section 4 which are defined by both the class and package and can be used in standalone files but also in the main document.

`\standaloneconfig{`⟨*options*⟩`}`

This configuration macro accepts the class options described in section 2.2. It can be used inside the class configuration file to set default settings used by all standalone files, as mention in section 2.5. These settings are set just before the class options of the standalone file are processed.

Certain class options (e.g. border, varwidth) which do not have a global effect can also be changed using this macro later in the preamble or even inside the document body between different content if the multi option is enabled.

`\standaloneenv{`⟨*environment*⟩`,`⟨*environment*⟩`,...}`

If the multi option is in effect this macro should be used to declare all environments which produce content. Common examples of such environments are `tikzpicture`, `pspicture` and other picture environments. This macro must only be used inside the preamble. Every use of such an environment in the document body will produce a new page. An exception are nested appearances of such environments, e.g. a `tikzpicture` inside a node of another `tikzpicture`. The environments must be previously defined and must not be redefined afterwards. Multiple appearances of the same environment name inside one or multiple `\standalonenv` should be avoided.

This macro uses `\PreviewEnvironment` internally if the preview option is active. Own code is used with the alternative crop option. If none of these options are enabled this macro will have not effect and will be silently ignored.

`\standaloneignore`

In rare cases some code must be placed before the `\documentclass` of a sub-file (e.g. `\PassOptionsToPackage`). Because the main document will only skip code between `\documentclass` and `\begin{document}` this code will be executed by it. In order to avoid this the macro `\standaloneignore` can be used at the very beginning of a sub-file to skip over this code. However it must be written as `\csname standaloneignore\endcsname` to avoid a 'Undefined control sequence' error when compiled standalone. After all the class is not loaded at this point, therefore no `standalone` macros are yet defined. The `\csname`...`\endcsname` construct will simple make it equal to `\relax` in this case.

Please note that all code before `\documentclass` is not processed by any of the subpreamble options but always simply removed. This macro was inspired by the similar macro `\docmute` of the `docmute` package.

`\begin{standalone}`
⟨*sub-file content*⟩
`\end{standalone}`

The `standalone` environment is automatically wrapped around the content of standalone files. If the multi option is enabled it is wrapped around every page, i.e. every environment declared with `\standaloneenv`. The definition of this environment depends on options like crop and preview. It is possible to redefine this environment in the configuration file or the document preamble to adjust the processing of the content, but this is not recommended. If done most content related options will stop work and/or cause errors.

The beamer specific macros and environments are described in section 2.4.

2.4 Support for Beamer Presentations

Presentation can be written in LaTeX using the `beamer` class. Each presentation frame is wrapped in a `frame` environment. Overlay effects can be added using special macros. This effects result in multiple pages per frame. Pictures with such overlay effects can not be compiled standalone using the normal settings. Instead the `standalone` class must load the `beamer` class and wrap the content also in a `frame` environment while skipping the `preview` environment. To activate this settings load the `standalone` class with the `beamer` option. Because the `frame` environment is quite special (it normally collects all it's content and calls the `\frame`) and must also support verbatim content it is not easily possible to redefined the `document` environment to include `frame`. Also `frame` accepts options which `document` doesn't. Therefore a second environment called `standaloneframe` is used in the beamer picture files. It will be equal to `frame` in standalone mode, but without effect otherwise.

`\ifstandalonebeamer`

Both the class and the package provide the if-switch `\ifstandalonebeamer`, which can be used to only include code if the file is compiled standalone with the `beamer` class option set. The switch is set to `\iftrue` by the class when loaded with the beamer option and always to `\iffalse` by the package. It can be used to place beamer specific options in the configuration files, which should be skipped for non-beamer standalone files. If used inside the configuration file this switch must be placed inside `\AtEndOfClass{...}`, because the beamer option is not yet processed

```
\begin{standaloneframe}<⟨overlay specification⟩>[<⟨default overlay spec⟩>]
                       [⟨options⟩]{⟨optional frame title⟩}{⟨optional frame subtitle⟩}
  ⟨code with beamer overlays⟩
\end{standaloneframe}
```

The `standaloneframe` environment must be used in sub-file holding beamer overlay code. It is only defined when the class is called with the `beamer` option and acts as a replacement of the `frame` environment of beamer when compiled standalone. All optional arguments of `frame` are supported but most might not be useful for normal sub-files. When compiled as part of a main document it does nothing except of gobbling its arguments.

The listings 3–5 shows a beamer standalone example and its effective code in standalone and main document mode.

Listing 3: Use of `standalone` class with `beamer` option.

```
% Use of 'standalone' class with a beamer overlay:
\documentclass[beamer]{standalone}
% Load packages needed for this TeX file:
\usepackage{tikz}
% Surround TeX code with 'document' environment:
\begin{document}
\begin{standaloneframe}[<options>] % e.g. 'fragile'
 % Add your TeX code:
  \only<1>{ One }%
  \only<2>{ Two }%
\end{standaloneframe}
\end{document}
```

Listing 4: Effective beamer code if compiled standalone.

```
\documentclass{beamer}
<beamer code from standalone.cfg file>
\usepackage{tikz}
\begin{document}
\begin{frame}[your options]
  \only<1>{ One }%
  \only<2>{ Two }%
\end{frame}
\end{document}
```

Listing 5: Effective code if included in a beamer presentation.

```
\begingroup
  \only<1>{ One }%
  \only<2>{ Two }%
\endgroup
\endinput
```

2.5 Class configuration file

The `standalone` class loads a configuration file called `standalone.cfg` just before the options are processed, but after all options and if-switches are declared. Any class options can then also be given using `\standaloneconfig{`⟨*options*⟩`}`. Settings which depends on the finally used options should be placed inside `\AtEndOfClass{...}`, so that they are processed after all options. This is particular required for `beamer` specific settings, because at load time of the

configuration file a given beamer option is not yet processed. Please note that this was handled differently before v1.0, so in old configuration files edited by the user the `\AtEndOfClass` must now be added.

A default configuration file is provided together with the bundle and holds some default settings. Because this file will be overwritten every time the bundle is updated, users should create an own configuration file in the local TEXMF tree or the document directory. In order to keep the default behaviour this file should either contain the content of the bundle configuration file or load it. Because it can be assumed that the bundle configuration file resides inside a `standalone` directory, therefore it can be loaded from a user configuration file using `\input{standalone/standalone.cfg}`.

2.6 Conversion to images

Using the convert class option the standalone file can be easily converted to an raster image. This is done by executing an external program to convert the output file (PDF or PS) to an image (recommended is the lossless PNG format, but also others are supported).

2.6.1 Conversion settings

Conversion settings can be given as the value of the `convert={`⟨*settings*⟩`}` option. By default conversion is disabled (`convert=false`). If enabled without providing own settings (`convert`, `convert=true`) the following default settings are used: PNG format, a density of 300dpi, no explicit size and the output file name is given by `\jobname`, i.e. the name of the LaTeX document. Using the convert option with any value other than `false` will enabled it. All normal conversion settings are listed in Table 1, while Table 2 lists the more advanced options which e.g. can be used to modify the conversion command directly.

2.6.2 Conversion software

The conversion requires an external image converter program to be installed. By default the two following tools are supported and either of them must be installed in order to use the conversion feature. In order for an external program to be executed the `-shell-escape` option[1] must be used for the compiler executable, e.g. `pdflatex -shell-escape filename`. Without this option no conversion is possible.

By default the conversion program of Image Magick is used for PDF LaTeX files, which is freely available for Unix/Linux, Mac and MS Windows. Under Ubuntu Linux it can be installed using the shell command '`sudo apt-get install imagemagick`'. The conversion executable is simply called

[1] Maybe named differently depending on the used LaTeX distribution

Table 1: Conversion Options (to be used in the value of convert class option)

Sub-Option	Description	Default value
`(no value)`	Conversion enabled with default settings	./.
`true`	Conversion enabled (with default settings if no other options are given)	(no value)
`false`	Conversion disabled	(no value)
`density`	Sets the density in dots-per-inch (dpi). Can be a single numerical value or '⟨*X*⟩`x`⟨*Y*⟩'.	`300`
`size`	Sets the size of the image. Can be a single numerical value or '⟨*X*⟩`x`⟨*Y*⟩'. If empty the size is determined by the density setting and the size of the PDF.	(empty)
`subjobname`	The jobname used for the internal LaTeX run	`\jobname`
`inext`	Input file extension including the leading dot	`.pdf` or `.ps`
`inname`	Name base of input file (i.e. file name without extension)	`\subjobname`
`infile`	Input file name	`\inname\inext`
`outext`	Output file extension including the leading dot	`.png`
`outname`	Name base of output file	`\inname`
`outfile`	Output file name	`\outname\outext`

Note: the settings (except 'true' and 'false') can also be used as macros in other settings.

Table 2: Advanced Conversion Options

Sub-Option	Description	Default value
`command`	Command line used for conversion.	(see `imagemagick`)
`imagemagick`	Sets the convert command to use Image Magick:	
	`command={\convertexe\space -density \density\space \infile\space \ifx\size\empty\else -resize \size\fi\space -quality 90 \outfile}`	
`convertexe`	Name of the executable of Image Magick.	(see section 2.6.2)
`ghostscript`	Sets the convert command to use Ghostscript:	
	`command={\gsexe\space -dSAFER -dBATCH -dNOPAUSE -sDEVICE=\gsdevice\space -r\density\space -sOutputFile=\outfile\space \infile}`	
`gsexe`	Name of the executable of Ghostscript.	(see section 2.6.2)
`precommand`	Command to be executed before the actual conversion command.	`dvips \jobname` (DVI/PS), empty (PDF)
`gsdevice`	The output device to be used for ghostscript. Already set up for PNG and JPG output.	Uses known device if defined for output format, otherwise the output format itself.
`onfailure`	Sets if an type of 'message' which should be triggered on conversion failure: `error`, `warning`, `info` or `ignore`.	`warning`

'convert'. However, there is another program with the same name provided by MS Windows itself which converts old FAT filesystems to NTFS! It has been suggested to rename the Image Magick executable to 'imgconvert' instead. By default standalone uses 'imgconvert' as executable if MS Windows is detected and 'convert' otherwise. The executable name can be change manually using the 'convertexe' conversion option or by using

```
\standaloneconfig{convert={convertexe={convert}}}
```

in the configuration file 'standalone.cfg'.

Another conversion program is Ghostscript which is a very common PostScript interpreter which also supports PDF. It is used by default for DVI/PS files. Under Ubuntu Linux it is most likely already installed but otherwise can be installed using 'sudo apt-get install ghostscript' or 'sudo apt-get install gs'. It can convert both to various output formats and is freely available for Unix/Linux, Max OS X and MS Windows. It requires to set the correct output device which is not always fully identical to the output format (e.g. 'png16m' for a PNG (with 16 million colors)). The devices for PNG and JPG are already configured. Other devices can be configured using the `defgsdevice={`⟨*.extension*⟩`}{`⟨*device*⟩`}` conversion setting. The Ghostscript executable is usually named 'gs' under Linux/Unix and 'gswin32c' under MS Windows and configured this way by default, but this may be changed using the gsexe setting.

2.6.3 Conversion process

The conversion process is currently implemented in the following way to allow the normal compilation and subsequent conversion using only one (manual) compiler run. Because the document must be fully compiled before the conversion can occur the standalone executes the same LaTeX compiler (e.g. textttpdflatex) again as a sub-process which compiles the current document fully. This is done when the standalone class is loaded, so that the main compiler instance is still at \documentclass and has not yet itself opened the output file for writing. After the document got compiled using the sub-process the external conversion tool will be executed. If required intermediate conversions like dvips are also executed beforehand. Finally the main compiler run is terminated without producing any output, keeping the output file generated by the sub-process intact. A drawback of this implementation is that the log file created by the sub-process is overwritten by the main process and does not hold meaningful information. This can be compensated by setting a different jobname for the sub-process using the subjobname conversion setting.

2.6.4 Conversion examples

PDF/PS is rastered with 600x100dpi and then converted to JPG:

```
\documentclass[convert={density=600x100,outext=.jpg}]{standalone}
```

Produces BMP with 400x400px (one side might be meder if content is not quadratic in shape):

```
\documentclass[convert={outext=.bmp,size=400}]{standalone}
```

Produces TIFF G4 output file using Ghostscript with a density of 72dpi:

```
\documentclass[convert={ghostscript,gsdevice=tiffg4,
                outext=.tiff,density=72}]{standalone}
```

Produces PNG (default) with a size of 640px (suitable to be uploaded on StackExchange sites without the image getting downscaled):

```
\documentclass[convert={size=640}]{standalone}
```

2.7 Simple TeX File

A simple `standalone.tex` file is provided together with the bundle, which may be useful in special occasions. It will set the `\ifstandalone` switch to *true* when compiled standalone but to *false* when loaded after any `\documentclass` macro, as long the switch isn't defined yet. It must be used if this switch is required before the `\documentclass` of a standalone file.

Listing 6: Usage of 'standalone.tex'.

```
\input{standalone} % use before any '\documentclass'
\ifstandalone
  % Used only if compiled standalone
\fi
```

2.8 FAQ / Troubleshooting

This section expands some issues and their solution which can arise with the `standalone` class.

Large white space / border at the right side

A large white space / border on the right side occurs when the content is placed inside a paragraph. This causes the content to be `\linewidth` wide and so smaller pictures will contain now a white space at the right. A common cause for this is that there was is a empty line between the content and `\end{document}` which causes a paragraph break.

This issue can be solved by either removing any trailing lines or other paragraph breaks, or by using the varwidth option which suppresses the extra added

width. It is also possible to use the multi option and `\standaloneenv{`⟨*environment name*⟩`}` to declare certain environments as page content. The tikz option does this for `tikzpictures` and the pstricks option for `pspicture`. See the descriptions of these options for more details.

Some amount of the content on the right side is missing

If the content is cropped to much on the right side, check if the varwidth option is used. In this case the used maximum width (`\linewidth` by default) is too small. A larger width can be set using varwidth=⟨*length*⟩ or the option can be disabled altogether using varwidth=`false`. The largest width possible is given by `\maxdimen`, which however might cause internal overflows.

This can also be caused with `beamer` content (i.e. when the beamer option is used). In this case no cropping or `varwidth` environment is used at all, but the content is simply to large to fit on a `beamer frame`. To avoid this rescale the content to do fit. This can be realised by either using scaling facilities of the used picture environment (like `scale` with `environment`, but this only scales coordinates) or using `\scalebox` or `\resizebox` from `graphicx`. For complicated code which contains verbatim or other catcode changing code either the `\Resizebox` from the `realboxes` package or the `{adjustbox}{scale=<factor>}` environment from the `adjustbox` package should be used.

A multi-page document contains some pages with unwanted content

This is caused while multi=`true` and crop=`true` are set but ignorerest=`false` *and* the document contains typeset material outside of environments declared with `\standaloneenv`. To avoid that this extra material should be removed or ignorerest should be set to `true`. This will also ignore all settings inside the document body which are not inside a declared environment. These can be moved to the preamble instead. See the description of the ignorerest option for more details.

In a multi-page document using DVI/PS mode all pages except the first have a vertical offset

The vertical reference points in PostScript could does not change when the pages are resized to fit the individual content of every page. Therefore an offset is added to compensate for this, which shifts the content to the appropriate vertical position. Should this not work as expected please inform the package author and provide a small example which causes this issue, together with the version number of the used `latex` compiler and tools (like `dvips`, `ps2pdf`) as well as the used `standalone` bundle.

Issues with cropped files in DVI mode

The crop option uses PostScript commans in DVI mode, i.e. when `latex` not `pdflatex` (or others) is used as a compiler. This PostScript commands will only work once the DVI is converted to PS or EPS. Currently this cropping code is experimental and might not produce a full (E)PS standard compatible file. This can lead to wrong bounding boxes and wrong orientations or, dependent on the used PostScript tool, even to PostScript compiler errors. Some issues can be overcome by converting the the (E)PS file to a (more) standard compatible version using tools like `eps2eps` or Ghostscript.

Errors "Float(s) lost" or "not in outer par mode"

Floating environments like `figure` or `table` can not be used while `float=true` and either `crop=true` or `preview=true` is set. The last two options will try to store the float into a box which is not allowed (because it can't the float any longer). Usually `float=false` will solve this error, because it turns these environments into non-floating alternatives. Because both the `crop` and `preview` option will set `float=false` themselves, this issue can only arise when the `float` option is manually set afterwards.

Image conversion does not work

In order for the image conversion to work an external conversion software must be installed. By default either Image Magkick or GhostScript is used. Please insure that either or both of these softwares are installed. Installation guide for your operating system should be easily available on the Internet. The LaTeX compiler option `-shell-escape` must be used to allow this external software to be executed from within the LaTeX code. If this two points are fulfilled but the conversion does still not work, please check the log file. The lines in question start with '`runsystem`' (at least with TeX Live 2011).

3 Usage of the standalone package

3.1 Basic usage

The standalone package needs simply be loaded using \usepackage in a main document. It redefines the \documentclass macro, which can occur in sub-files, so that it ignores anything till the next \begin{document} and then takes the document environment as a simple group. The real document environment in the main file is not affected. Sub-files can then be included in the main document body using \input{⟨*filename*⟩}.

The standalone package must not be loaded before the document class using \RequirePackage, because this will cause issues. Also it is not possible to \input standalone files inside the preamble, e.g. as part of a \savebox assignment.

It is possible to cascade standalone files, i.e. \input a standalone file from within a standalone file. Then both the standalone class and the standalone package must be loaded by the any parent standalone file. These parent files can still be used inside other LaTeX documents if these load the standalone package themselves.

See section 3.2 for a list of package options which enable further features.

3.2 Package options

The following options are supported by the standalone package. Most of them are boolean options which take either 'true' or 'false' as optional values. If such an option is used without a value, 'true' is used. If not mentioned otherwise all options set to 'false' initially. Options might switch other options on or off. The order of the option is obeyed and later options will prevail over earlier ones. Note that some older versions of the standalone package only take the option without any value.

`subpreambles=true|false`

The standalone package removes all sub-file preambles ("sub-preambles") by default when loaded. However, if the package is loaded with the subpreambles options, all sub-preambles are stored in an auxiliary file with the name '⟨*main tex file name*⟩.sta' (for *sta*ndalone). This file is then loaded or processed at the beginning of the next LaTeX run (i.e. at the place in the preamble where the standalone package is loaded). The way how the subpreambles option works can be controlled by the options sort, print and comments/nocomments. Please note that the sort and print options require of course the subpreambles option and will enable it if not already done so.

`sort=true|false`

With only the subpreambles option set, the sub-preambles are simple read and executed unchanged. This includes the risk of option clashes if one package is loaded with different options inside the sub-preambles and/or the main preamble. This is avoided by the sort option, which accumulates all packages loaded by all sub-files together with their options. The options are then marked to be loaded by the package using LaTeXs `\PassOptionsToPackage` macro. The packages are loaded at the end of the preamble using the `\AtBeginDocument` hook. This allows the user to load the same packages with own options in the main file, after the `subversion` package is loaded, without any option clashes.

`print=true|false`

While the sort option is giving already good results, problems with the order of packages can still occur. Some packages provide, redefine or patch the same macros, so that they must be loaded in the correct order to give the desired result. Potential additional code in the sub-preambles, required for some sub-figures but maybe incompatible with others, complicates the situation further. If such issues occur they can hardly be handled in an automatic way. Instead the sub-preambles must be carefully merged into the main preamble. The option print was created to simplify this otherwise cumbersome task. It concatenated all sub-preambles into a single file named '⟨*main tex file name*⟩`.stp`' (for *st*andalone, *p*rint). Each preamble is commented with its original file name. Please note that `.sta` file mentioned above, while quite similar, holds additional macros and might not be easily user readable or editable. After the file was generated it can be easily pasted into the main file preamble using a text editor.

When the print option is enabled the normal `.sta` file is not generated or loaded. Because this will cause most likely some errors related to packages not loaded, all sub-file bodies will be skipped. A warning is printed for each sub-file to remind the user about this fact. The print option is only indented to by used when required to get a list of sub-preambles. After including this list in the main file the option must be removed to compile the main file normally.

`print,sort`

Finally if both the print and sort options are enabled, a 'sorted' list of sub-preambles is printed into the `.stp` file. In this 'sorted print' mode all `\usepackage` macros and other similar macros like `\usepgflibrary`, `\usetikzlibrary` as well as `\usetikztiminglibrary` from the `pgf`, `tikz` and `tikz-timing` packages, respectively) are removed from the rest of the sub-

preamble code. A list of packages (and libraries) without duplicates is printed at the begin of the `.stp` file (using `\usepackage`, of course). Every option provided by any sub-file for a package is added, again without duplicates. If specific package date was requested in a sub-file it is also added. If multiple dates are requested for one package, the most recent (i.e. the "highest one", not the last processed) is used. After this list(s) the rest of the sub-preamble code is printed with the above macros removed. This mode frees the user from the need to remove duplicates and collect package options manually.

Please note that all `\usepackage` and similar macros inside braces `{}` will not be seen by `standalone`s sort macro and therefore are not extracted or handled in any special form mentioned above. This can be exploited to load certain packages only in `standalone` mode but not in the main document. Unfortunately, macros inside `\ifstandalone...\fi` are seen and extracted while not wanted inside the main file. The macro `\onlyifstandalone{`⟨*code*⟩`}(see below)` was created because of this two reasons. Its argument braces hide the content from the scanner. It is then also completely removed from the printed sub-preamble code.

`comments=true|false`
`nocomments`

The comment option selects if the `.stp` file should also include the comments of the sub-preambles. For backwards compatibility nocomments exists which is identical to `comments=false`. Comments are included by default in the non-sorting print mode (print without sort option), but can cause 'wrong' results during the 'sorting' process and are therefore removed by default in this mode. The reason for this can be explained as follows. In order to transfer the comments from the sub-files to the `.stp` file TEX must be instructed to handle them as normal input and not discard them. However, in this case the scanning algorithm which removes `\usepackage` and friends can not distinguish between 'active' macros and macros which are commented out. All above mentioned macro inside comments will then be processed as when there where 'active'. The user might favour the information provided by the comments over this small risk and enable them using the comments option.

`group=true|false`

This option is set the '`true`' by default and controlled whether or not a group is added around the content of standalone files. Normally ('`true`') the `document` environment of the sub-files is turned into an environment which does nothing, besides adding the usual group. If set to '`false`' this environment made transparent, so that no group is added. Any definition inside the document body of sub-files will still be accessible after the `\input`

macro. Note that this does not effects the **\includestandalone** macro which always will add a group.

mode=⟨*mode*⟩

Sets the mode for **\includestandalone**. Valid values are '**tex**' (use source file, default), '**image**' (use existing image file produced by the source file), '**image|tex**' (use image if available, source otherwise), '**build**' (build image from source, then use it), '**buildmissing**' (only build image if it does not exist) and '**buildnew**' (only build image if source file is newer). See section 3.3 for more details. See also section 3.4 for further details.

obeyclassoptions=true|false

If this option is enabled the **\includestandalone** will try to obey the class options used in the standalone files while in '**tex**' mode. This only works if the standalone file uses the **standalone** class and only with certain options. The class configuration file will also be loaded (in a local scope, for every standalone file) in order to load the default settings.

This feature is intended to ensure (nearly) identical results independent if the standalone files are included as source code or as image, in order to permit an easy switch between this two modes. In particular, the standard size options **10pt**, **11pt** and **12pt** are applied to the standalone file (supported for the standard and KOMA Script classes) as well as the border class option. The multi'='<environment>, ... option is supported and will make the **page=**⟨*number*⟩ option of **\includegraphics** work with **\includestandalone**. This means, that one particular page can be selected, while all other environments are skipped. By default the first page is taken (if multi was used). The special value of **-1** will include all pages from the source file (but not from the image). Because multi option will assume that either crop or preview is enabled and will always ignore other content like with **ignorerest=true**. These three class option will be ignored by the package, which might lead to different behaviour between standalone and main-document mode, but only for uncommon cases where multi is used without declaring environments and with disabled cropping (crop/preview). In order to support a potential varwidth option the **varwidth** is loaded if it is available.

This is an extended feature, which requires substantial amount of extra code and some advanced techniques to switch the font size. It might not work correctly under all circumstances. Because of this it is disabled by default. At the moment it does not take the class configuration file into account and does not work for **beamer** standalone files.

`extension=`⟨*.extension*⟩

The image file extension (with leading dot) used for `mode=image` can be selected using this option. By default the target output file extension of the used LaTeX compiler is used, i.e. '`.pdf`' for `pdflatex`, `lualatex` and `xelatex` and '`.eps`' (converted from DVI) for `latex`.

`build={`⟨*build options*⟩`}`

This option allows to set the options used for building images from standalone files. See section 3.4, especially Table 3 for further details.

3.3 Macros

The following user macros are provided by the `standalone` package. Further macros are listed in section 4 which are defined by both the class and package and can be used in standalone files but also in the main document.

`\standaloneconfig{`⟨*options*⟩`}`

This configuration macro accepts some of the package options described in section 3.2. These options are group, mode, extension and build, which can be changed for different included standalone files.

If both the `standalone` class and package is used together this macro can also be used to set the class options as described in section 2.3.

`\includestandalone[`⟨*options*⟩`]{`⟨*file*⟩`}`

This sophisticated macro can be used instead of `\input` to include standalone files. Its behaviour is controlled by the mode package option. This macro can either include the source code in the same way as `\input` (`mode=tex`), include the output file (PDF, EPS) using `\includegraphics` (`mode=image`), try first the output file and use the source file if it is available (`mode=image|tex`), build the output file from the source file either always (`mode=build`), only if the image files does not exist (`mode=buildmissing`) or only if the source file is newer (`mode=buildnew`). See also the section 3.4 for further details.

The ⟨*file*⟩ argument must be the file name of the standalone source file *without* the extension. The macro accepts the same ⟨*options*⟩ as `\includegraphics` as well as any options suitable for `\standaloneconfig`. This means that the source file can also be resized and rotated in '`tex`' mode like an image. TODO: In this mode the package also tries to extract and apply the class options from the standalone file and apply these to the included source. Unfortunately, it can not be fully guaranteed that the standalone content will be displayed identical in source code and image mode. Some settings might not be applied in the same way and rounding differences may occur.

3.4 Building images from standalone files

Using the `\includestandalone` macro standalone files can be either included directly as source files or as vector graphic images which are build from these. The `standalone` package provides the feature to automatically build image files from given standalone source files. This is controlled by the mode options. This was already described in section 3.2 and 3.3.

This enables the user to switch easily between including source code or images, either globally or only for selected standalone files. Using images has the benefit that the included material, often complicated pictures, does not have to be recompiled every time with the main document. This leads to significant speed improvements. The drawback is a slight increase in file size, because the material will have its own file headers. Also any settings done in the main document which would affects the source code will not have an effect on the image. This can be positive or negative dependent on the case.

An extended feature is the automatic building of images from the standalone files, either always or only if the source files are newer than the existing image files. In this cases the `\includestandlone` macro will call the LaTeX compiler on the standalone files in question to produce the images, then include these using `\includegraphics`. This requires the '`-shell-escape`' compiler option to be set, otherwise the execution of shell commands is disabled for security reasons.

The image files will normally be created in the current directory of the main document, which is not necessarily the same directory where the source files are located. Dependent on the used compiler settings, files in the current directory will be found first before other directories are searched. Using `mode=buildnew` newly build image files placed in the current directory will therefore taken before older images files potentially located in the directory of the standalone files. Because the exact directory of source files is not accessible within LaTeX documents, it is not possible to create the images files always in the same directories as the source files. Compiler options like '`-output-directory`' can be useful to influence the output directory of the build images. However, these options must be used with the internal compiler run, i.e. by setting `build={latexoptions=`⟨...⟩`}` appropriately, not (only) on the main LaTeX compiler run.

If the build process fails a warning is issued and the source code will be included instead. It should be noted that failure detection is not perfect and might lead to false positives or negatives.

4 Common macros

The following conditional macros are defined by both the `standalone` class and package, but react differently when the code is compiled standalone or as part of a main document.

Table 3: Build settings

Build setting	Description	Default value
`latex`	LaTeX compiler to be used	Same as main compiler
`latexoptions`	Command line options for compiler	`-interaction=batchmode -shell-escape -jobname '\buildjobname ' \file`
`jobname`	Jobname for build compiler run	`\latex \space \latexoptions \space \file`
`command`	Full build shell command	`dvips -o '\file.eps' '\file.dvi'` (DVI mode only)
`postcommand`	Command executed after main command, to produce final output file	

Note: the settings (except '`command`' and '`postcommand`') can also be used as macros in other settings. The given file name is available (without extension) as `\file`.

`\ifstandalone`

Both the class and the package provide the if-switch `\ifstandalone`, which can be used to only include code if the file is compiled standalone. The switch is set to `\iftrue` by the class and to `\iffalse` by the package.

The additional file `standalone.tex` also defines this switch by checking if `\documentclass` was already used. It can be included with `\input{standalone}` and is intended for specialised files which do not use the `standalone` class.

`\IfStandalone{`⟨*code for standalone mode*⟩`}{`⟨*code for main document*⟩`}`

This is the macro version of the `\ifstandalone` if-switch. It executes the first argument only in `standalone` mode, i.e. when the file is compiled on its own. When included in the main document the second argument is executed instead. As mentioned in section 3.2 it can also be used to hide `\usepackage` and similar macros from the extraction scanner of the `sort` option. The macro and its arguments is not printed into the `.stp` file.

`\onlyifstandalone{`⟨*code*⟩`}`

This macro is similar to `\IfStandalone` but only has takes one argument which is executed only in standalone mode, but ignored when compiled as part of a main document. As mentioned in section 3.2 it can also be used to hide `\usepackage` and similar macros from the extraction scanner of the `sort` option. The macro and its argument is not printed into the `.stp` file.

5 Usage Examples

Example 1: Use of *standalone* package.

```
% Main file
% Real document class:
\documentclass{article}

% Use the 'standalone' package:
\usepackage{standalone}

% Load all packages needed for all sub-files:
\usepackage{tikz}

% Inside the real 'document' environment
% read the sub-file with '\input'
\begin{document}
% ...
\begin{figure}
  \input{subfile}
  \caption{A subfile}
\end{figure}
% ...
\end{document}
```

Example 2: Use of *standalone* class.

```
% A sub-file (e.g. picture) using the 'standalone' class:
% Use 'standalone' as document class:
\documentclass{standalone}

% Load packages needed for this TeX file:
\usepackage{tikz}

% Surround TeX code with 'document' environment as usually:
\begin{document}
% Add your TeX code, e.g. a picture:
\begin{tikzpicture}
  \draw (0,0) rectangle (2,1) node [midway] {Example};
\end{tikzpicture}
\end{document}
```

Example 3: Effective code if compiled standalone.

```
\documentclass{article}

\newenvironment{standalone}{\begin{preview}}{\end{preview}}
\input{standalone.cfg}
% which by defaults loads:
% \PassOptionsToPackage{active,tightpage}{preview}
\usepackage{preview}

\usepackage{tikz}

\begin{document}
\begin{standalone}
\begin{tikzpicture}
  \draw (0,0) rectangle (2,1) node [midway] {Example};
\end{tikzpicture}
\end{standalone}
\end{document}
```

Example 4: Effective code if included in a main document.

```
\begingroup
\begin{tikzpicture}
  \draw (0,0) rectangle (2,1) node [midway] {Example};
\end{tikzpicture}
\endgroup
\endinput
```