Active-DVI

Reference manual

Version 1.10.0

Didier Rémy and Pierre Weis

August 5, 2011

Active-DVI is a viewer for DVI files that also recognizes a new class of `\special`'s targeted to presentations via laptop computers: various visual effects can easily be incorporated to the presentation, via a companion `advi.sty` LaTeX package.

Acknowledgments and contributors

Active-DVI is based on Mldvi, written and distributed by Alexandre Miquel under the LGPL license[1]. This constitutes the core rendering engine. Active-DVI has then been developed by

Jun Furuse Didier Rémy, and Pierre Weis, with contributions by Didier Le Botlan, Roberto Di Cosmo, Xavier Leroy, Alexandre Miquel, and Alan Schmitt.

[1] At the url `http://www.pps.jussieu.fr/~miquel/soft.html`.

Contents

1 Installation

After successful installation of Active-DVI, don't forget to have a look at example presentations provided with the distribution, in the directories:

- `test`: a lot of demonstration presentations. From the main Active-DVI directory, just type `cd test; make all`, and launch `advi` on files `*.dvi`.
- `examples`: examples to get inspiration from to easily write your own talks. A full range of examples, from the simplest one to the most involved.
  - `basics`: very simple presentations (slightly longer presentation in sub-directory `cash`).
  - `slitex`: presentations using the `advi-slides` package. Each presentation has its own directory with a `Makefile` to automate the process of keeping the presentation up-to-date with the sources. Nothing difficult here.
  - `seminar`: one presentation using the popular `seminar` LaTeX package in the `clock` sub-directory.
  - `prosper`: two example presentations using the `prosper` LaTeX package in sub-directories `LL` and `Join`. Quite impressive and involved.

1.1 Automatic installation

In some cases, your Unix system provider will ensure an up-to-date port of the Active-DVI binaries; in such a case, use the recommended procedure to download, compile, and install the software.

In any other case, you have to install Active-DVI manually.

1.2 Manual installation

Manual installation in two steps

To compile Active-DVI, you must install the `camlimages` library first, including its native code version.

Hence, download and install `camlimages` first. After successful installation of this library, download and extract the Active-DVI source files, move to the new source directory of Active-DVI, and compile Active-DVI itself.

Manual installation in one step

This is the "Manual installation in two steps" that has been packaged for you in one single step by the Active-DVI development kit.

Download and install the entire Active-DVI development kit (a.k.a. `adk`) source tree provided by the distribution: it contains Active-DVI sources *and* the `camlimages` library sources, plus a set of `Makefile`s to automate the installation procedure. Then, just type `make` in the directory where the Active-DVI Development Kit has been extracted from the archive. If everything goes well, it automatically compiles and installs the necessary libraries and Active-DVI for you. Some fine tuning may be necessary, if you are not running a standard Linux distribution, such as RedHat, Mandrake, or Debian.

If you have to install some other software to solve installation problems, you should type `make reconfigure` to re-perform the configuration process before launching `make` again.

2 Active-DVI for the impatient

- As a previewer, Active-DVI can preview any correct DVI file.

- As a presenter, Active-DVI provides some LaTeX packages to facilitate animations and interaction with the presenter from within your LaTeX source text. The `advi-slides.sty` package is designed to be a simple way to build a presentation for Active-DVI.

See `examples/slitex/simplistic/` for a simplistic talk example. There, the command `\pause` is used to make the presenter to stop while displaying the document. More involved examples can be found in the directory `examples` of the distribution.

3 Safety concerns when using the Active-DVI previewer

Warning! Active-DVI may execute programs and commands embedded into the DVI file. Hence, when playing a DVI file from an untrusted source, you should run `advi` with the `-safer` option that inhibits the execution of embedded applications. This warning applies in particular if you choose Active-DVI as your default `meta-mail` previewer for the `application/x-dvi` mime-type.

The default safety option is the `-ask` option: it tells Active-DVI to ask the user each time it must launch an application. (Note that in such a case Active-DVI asks only once to launch a given application: it remembers your previous decisions concerning the command and acts accordingly for the rest of the presentation.)

The second safety option is the above mentioned `-safer` option: it completely inhibits the execution of embedded applications.

The last safety option is `-exec`: if you call `advi -exec`, `advi` automatically and silently launches all embedded applications (this is useful to play your own presentations without the burden of answering yes to Active-DVI's questions).

As mentioned, the safe `-ask` option is the default, automatically set when nothing has been explicitly specified by the user. If desired, the default safety option can be set via *initialization files*, either on a system large scale by the machine administrator (in the file `/etc/advirc`), on a local scale by individual users (setting the default policy for that user), or even on a per directory basis (setting the default policy to show DVI files in this directory)!

(This last option is convenient to gracefully run your own talks, while still being cautious when running talks from others.)

4 Initialization files for Active-DVI

4.1 Syntax of initialization files

An initialization file for Active-DVI is simply a text file that contains options exactly similar to those you can give on the command line (with the exception of comments, made of a sharp sign (`#`) followed by some text that is ignored until the end of line). For instance:

```
-exec # I know what I mean!
-bgcolor grey16
-fgcolor grey95
```

is a valid initialization file that sets the safety policy to `-exec`, then sets the background and foreground colors to obtain a nice reverse video effect.

4.2 Loading initialization files

Before parsing options on the command line, Active-DVI loads, in the order listed below, the following initialization files (nothing happens if any of them does not exist):

- system wide initialization file: `/etc/advirc`,
- user specific initialization files: `~/.advirc` then `~/.advi/advirc`,
- directory specific initialization file: `./.advirc`.

4.3 Automatic setting of options

In addition, the user may load an arbitrary file containing options by specifying the file path via the command line argument `-options-file`. Hence, `-options-file filename` loads `filename` when parsing this option to set up the options contained in `filename` (thus overriding the options set before by the default `~/.advirc`, `~/.advi/advirc`, or `./.advirc`, initialization files).

5 Using the Active-DVI presenter

5.1 Launching Active-DVI

Active-DVI is invoked with the following command syntax

```
advi [options] dvifile [dvifile]
```

Once Active-DVI is launched, just press `?` to get help on the keys you can use to control the presenter (type `^f` (Control-F) to get full screen, `<` or `>` to change the magnification of the text).

When two file names are provided Active-DVI displays them both: see section 5.12 for details on the use of a secondary DVI file.

5.2 Command line options

The `advi` commands recognized the following options:

Help and info options

`-v`	Prints the `advi` current version and exits
`--version`	Prints the full `advi` current version and exits
`-help`	Short command line options help

Window and display specifications

`-geometry geom` Geometry of Active-DVI's window specification Geometry `geom` is specified in pixels, using the standard format for X-Window geometry specifications (i.e: `WIDTHxHEIGHT[+XOFFSET+YOFFSET]`).

`-fullwidth` Adjust the size of the window to the width of the screen

`-nomargins`	Cancel horizontal and vertical margins
`-hmargin dimen`	Horizontal margin specification (default: 1cm)
`-vmargin dimen`	Vertical margin specification (default: 1cm)

Dimensions are specified as numbers optionally followed by two letters representing units. When no units are given, dimensions are treated as numbers of pixels. Currently supported units are the standard TeX units as specified in the TeX book (D. Knuth, Addison-Wesley, (C) 1986): ‘pt’ (point), ‘pc’ (pica), ‘in’ (inch), ‘bp’ (big point), ‘cm’ (centimeter), ‘mm’ (millimeter), ‘dd’ (didot point), ‘cc’ (cicero) and ‘sp’ (scaled point). Note that dimensions are specified w.r.t the original TeX document, and do not correspond to what is actually shown on the screen, which can be displayed at a different resolution than specified in the original TeX source.

`-crop`	Crop the window to the best size (default)
`-nocrop`	Disable cropping

Color specifications

`-fgcolor <color>`	Specify the color of the foreground color
`-bgcolor <color>`	Specify the color of the background color
`-rv`	Specify that reverse video should be simulated by exchanging the background and foreground colors
`-gamma <float>`	Specify gamma correction (> 0.0) of glyphs

Helpers specification

`-pager`	Specify the name of the pager to launch on a `txt` link
`-browser`	Specify the name of the browser to a `html` link

Debugging options

`--debug`	General debug option
`--debug_pages`	Debug page motion
`--show_ps`	Print a copy of Postscript sent to gs to stdout
`--verbose_image_access`	Change the cursor while loading images

Rendering options

`-A`	Toggle Postscript anti-aliasing
`-passive`	Inhibit effects that are visible when redrawing the page (Transitions, delays, embedded applications)

Safety options

`-exec`	Set safety policy to "always execute embedded applications"
`-ask`	Set safety policy to "ask user before execution of embedded applications"
`-safer`	Set safety policy to "never execute embedded applications"

Option files option

`-option-file <filename>`	Load `filename` as a file containing a list of options as given on the command line to `advi`.

Miscelleanous options

`-autoswitch`	Set the `autoswitch` flag, which allows implicit switch to master on `usr1` signal (default is off).

5.3 Cut and paste

Text can be copied from the Active-DVI previewer to another application. However, this uses the `XBuffer` and not the `XSelection` mechanism.

- Shift middle-click copies the current word.
- Shift right-click and drag copies the specified region.

Moreover, Shift left-click dump an ASCII representation of the region under the mouse pointer in the source file. This expects the DVI to have been instrumented with line numbers of the form

`line:` ⟨*line*⟩ ⟨*file*⟩

where ⟨*line*⟩ and ⟨*file*⟩ are the current source line and current source file.

The position is exported in ASCII, in the form

`#line` ⟨*before*⟩, ⟨*after*⟩ `<<`⟨*prefix*⟩`>><<`⟨*suffix*⟩`>>` ⟨*file*⟩

where ⟨*before*⟩ and ⟨*after*⟩ are the enclosing line numbers, ⟨*prefix*⟩⟨*suffix*⟩ the word constituent surrounding the mouse position, and file is the name of the current file.

Line numbers default to 0 when not found. Note that line numbers may be inconsistent if there `\special`-line commands have not been inserted close enough to the position.

5.4 Hyper-text references

Active-DVI supports the LaTeX `hyperref` package with both internal and cross-file references. For cross-file references, it launches a new `advi` process to view the target.

Active-DVI improves the treatment of hyper-refs over conventional previewers, by emphasizing the hyper-target text of an hyper-link. Thus, an hyper-target definition:

`\hypertarget{`⟨*tag*⟩`}{`⟨*text*⟩`}`
should make the activation of the link ⟨*text*⟩ not only move to the page where ⟨*tag*⟩ occurs, but also emphasize the destination target ⟨*tag*⟩. However, since `\hypertarget` does include its second argument within the target, we use the following command instead:

```
\edef\hyper@quote{\string"}
\edef\hyper@sharp{\string#}

\newcommand{\softtarget}[2]%
  {\special{html:<a name=\hyper@quote#1\hyper@quote>}#2%
   \special{html:</a>}}
```

(If you are viewing this document with Active-DVI, you may move over this area or click on this one to see the effect.)

Similarly, to define a link target we use:

```
\newcommand{\softlink}[2]%
  {\special{html:<a href=\hyper@quote\hyper@sharp#1\hyper@quote>}#2%
   \special{html:</a>}}
```

5.5 Floating table of contents and thumbnails

There are two ways to include a floating table of contents while previewing.

- Active-DVI recognizes the reserved hyper-targets `advitoc.first` and `advitoc.last` as markers for the first and last pages of the table of contents. These pages then become floating, *i.e.* accessible from anywhere in the document with the default key binding `t`. The first stroke on `t` shows the first page of the table of contents. Successive stokes will show the following pages. (As usual, prefix integer argument may be used to directly access a specific page of the table of contents.)

  The package `advi` described below redefines the macro `\tableofcontents` so that it automatically inserts the reserved hyper-targets markers around the table of contents. It also provides two new macros, `\advitoc` and `\endadvitoc`, that serve to insert these markers when the table of contents is hand-made.

- If no table of contents markers are found, then Active-DVI will compute thumbnails, *i.e.* will show the whole set of pages of the presentation, each page drawn at a smaller scale and packed with the others on a single page. Active-DVI computes the scale so that all the thumbnails fit on one page only, provided that the scale is less or equal to a maximal value; otherwise, the maximal value scale is selected and the thumbnail pages spread on several pages. The default maximal scale value is 5, so that 25 thumbnails can fit on the same page. This value can be changed using the option `-thumbnail_scale`.

  Normally, thumbnails are drawn for all the pages. However, thumbnail pages can also be defined manually, with an hyper-target whose anchor is of the form `/page.`⟨*suffix*⟩. In this case, all the desired thumbnails must be explicitly marked.

By default, the binding `T` processes thumbnails and the binding `t` displays thumbnails if already processed, or shows the table of contents if available. Otherwise pressing `t` has no action. Thumbnails computation is explicit, so that incidentally hitting the `t` key does not lead to an unexpected computation, hence an unexpected delay.

5.6 Moving around

See the key bindings in the appendix.

5.7 Scratching on slides

During the show you can annotate your slides, entering the *scratching* mode. There are two modes, one for writing characters (the *writing mode*, entered by pressing `s` during the show) and the other to draw lines or figures on the slides (the *drawing mode*, entered by pressing `S`). In each of this modes, you can enter the *scratch setting mode* to set various properties of the scratching process. See the relevant key bindings for writing mode and drawing mode in the appendix.

5.8 Using the laser pointer

If you press `^X-l` (Control-X then l) the laser pointer appears on the slide; the pointer sticks to the mouse pointer and allows easy pointing to parts of the presentation. The laser pointer size and color can be set on the command line (options `-laser-pointer-color` and `laser-pointer-size`).

5.9 Saving slides

You can save a snapshot of the current slide at any time by pressing `^X-^S` (Control-X then Control-S). An image file is written (by default a `png` file). The name of the file produced can be set via the command line (see `advi -help` for details) or directly from within the LaTeX source file with commands `\advisavepageimage` and `\advisavepageimagefile{filename}`).

5.10 Creating events from the LaTeX source file

Active-DVI provides the command `\advipushkeys` that provides key presses to the presenter as if you had pressed it when viewing the presentation. For instance:

```
\advipushkeys{"q"}
```

ends the talk immediately.

Note that control keys must be encoded inside key strings passed to Active-DVI: we use the Emacs textual convention. For instance, the character "Control-A" (ASCII 1) is denoted by the two charcters `^X` (i.e. a carret character immediately followed by an X). Hence, the command

```
\advipushkeys{"^X^F"}
```

switches to full screen mode.

5.11 Using and making special effects

Presentation examples can be found in the `examples` directory. Don't miss to play them! Then, feel free to read their source code and copy the effects they provide.

Active-DVI can be used as is, but will shine when driven by a user with a bias towards programming: special effects can easily be realized by using the LaTeX packages provided with the distribution.

Creative advanced users may program the presenter at various levels, either using or defining simple LaTeX macros, writing new LaTeX package files, or by implementing extensions to the previewer itself.

5.12 Viewing multiple files simultaneously

Active-DVI can be invoked with several DVI files (currently only two). The first file is always used as the master file and others are client files. The user can switch between files explicitly (see key bindings) or implicitly. There is an implicit switch from the master to the client file *c* when an hyperlink that is not found in the master file can be found in the client file *c*; there is also a switch from the client *c* to the master when using the history stack and the previous event on the stack was an implicit switch from the master to the client *c*.

If `autoswitch` flag is set, there is also an implicit switch to the master, whenever Active-DVI receives signal `usr1` (to mean immediate refresh).

6 The `advi.sty` LaTeX package

Active-DVI provides some LaTeX packages to facilitate animations and interaction with the presenter from within your LaTeX source text.

The `advi.sty` package is the main package to include when writing a presentation for Active-DVI. It defines the main set of interactive commands for Active-DVI to animate the show. However, there is no need to load the package if no Active-DVI special effects are required for the presentation.

Warning! Most commands of `advi.sty` use the TeX `\special` command to insert into the DVI output file the Active-DVI specific commands that implement their semantics. Those commands are interpreted by Active-DVI afterwhile during the DVI file previewing. Note that a `\special{bla bla}` command is equivalent to a `\hbox{}` for TeX's mouth, hence it may alter the document layout accordingly. Thus, be aware that most commands of the `advi.sty` package are equivalent to a `\hbox{}` command as far as the document layout is concerned.

6.1 Printing the presentation

The `advi.sty` package recognizes the special option `ignore`, which helps the production of a printable version of the presentation: the `ignore` option makes the package not to produce Active-DVI specials, so that the show can be previewed by other DVI previewers or turned into Postscript using `dvips`. Of course, this option disables most effects that cannot be printed, although some of them are still approximated.

If the `ignore` option is not set globally, it can be set locally with the commands `\adviignore`. However, this will not prevent all effects, since some decisions are taken when the package is loaded.

The package also defines the conditional `\ifadvi` which evaluates its first argument if advi is not in ignore mode and its second argument otherwise.

6.2 Pauses

Active-DVI provides partial display of pages (slide "strip-tease"): the Active-DVI's rendering engine stops before the display of the current page is complete. The corresponding state is named a *pause*. Upon reaching a pause, Active-DVI may wait for a specified delay, or for user input.

`\adviwait[`⟨*seconds*⟩`]`

> Wait for ⟨*seconds*⟩. If no argument is provided, waits until the user requests to continue (hitting a key to move to next pause or to change page).

6.3 Active-DVI records

Active-DVI allows (almost) any piece of LaTeX code to be recorded, and the corresponding DVI code to be rendered later upon request. To be able to render the code in any order we choose, we must bind the recorded LaTeX code to a name (called a *tag*): this LaTeX code together with its tag defines the notion of an *Active-DVI record*.

Warning! The entire DVI image of an Active-DVI record *must* fit on a single DVI page. The corresponding check is left to the writer of the document.

The command defining an Active-DVI record is as follows:

`\advirecord[play]{`⟨*tag*⟩`}{`⟨*latex code*⟩`}`
`\begin{advirecording}[play]{`⟨*tag*⟩`}{`⟨*latex code*⟩`}`⟨*text*⟩`\end{advirecording}`

> This command processes ⟨*latex code*⟩ and records the corresponding DVI output, then binds it to the tag ⟨*tag*⟩. While recording, the DVI output is not displayed, unless the option `play` is set.
>
> Active-DVI records may be nested. In this case, the inner record is bound to its own tag as usual; in addition, if the inner record is defined with the `play` option, it is also recorded as a part of the outer tag record.
>
> If the environment syntax form of Active-DVI record definition is used, the ⟨*latex code*⟩ may contain fragile commands.

To play an Active-DVI record, the corresponding DVI must have been recorded on the current DVI page and before issuing the play command. With the proviso that these requirements are satisfied, the syntax of the command to display an Active-DVI record is as follows:

`\adviplay[`⟨*color*⟩`]{`⟨*tag*⟩`}`

> This command plays the DVI code previously recorded and bound to ⟨*tag*⟩.
>
> The optional argument changes the text color to ⟨*color*⟩ during replay.

6.4 Active-DVI anchors

Active-DVI gives the ability to define *Active-DVI anchors*: an anchor is specified by (1) an Active-DVI record, (2) a LaTeX piece of code that defines the area of the page where the anchor is active, and (3) an activation method.

The anchor is *activated* when some mouse events specified by the activation method occur in the area. The anchor is *de-activated* when the event that triggered the activation does not hold any more.

The Active-DVI record associated with the anchor is automatically rendered *anew* each time the anchor is activated. The page is reset to its original appearance when the anchor is de-activated.

The activation method of an anchor may be either `over` or `click`. If the activation method is `over`, the anchor is activated whenever the mouse pointer is inside the anchor area; conversely, the anchor is de-activated when the mouse leaves the anchor area. If the activation method is `click`, the anchor is activated whenever the button is pressed inside the anchor area; conversely, the anchor is de-activated when the button is released.

`\advianchor[`⟨*activation*⟩`]{`⟨*tag*⟩`}{`⟨*text*⟩`}`
`\begin{advianchoring}[`⟨*activation*⟩`]{`⟨*tag*⟩`}`⟨*text*⟩`\end{advianchoring}`

> This command defines the DVI rendering of {⟨*text*⟩} as an Active-DVI anchor area that plays the Active-DVI record bound to ⟨*tag*⟩ when the anchor is activated.
>
> The argument ⟨*activation*⟩ specifies the activation method of the anchor.
>
> If the environment syntax form is used, ⟨*text*⟩ may contain fragile commands.

6.5 Images

Images can be encapsulated into the presentation using the Caml library CamlImages provided with the distribution of Active-DVI (see section 8.4).

Images can be *lighten* by specifying an alpha value (a floating point number between 0 and 1) that measures the mixing between the background and the image.

Images can also be *blended,* meaning that you can choose the algorithm that superimposes the image to the background. Blending modes are reminiscent of the Ghostscript blending options: the blend mode must be one of the following: `normal`, `multiply`, `screen`, `overlay`, `dodge`, `burn`, `darken`, `lighten`, `difference`, `exclusion`, (`none` means unset).

```
{\setblend{burn}
  {\setalpha{0.5}
    {\includegraphics[width = 0.7\textwidth]{bar.eps}}}}
```

6.6 Colors

The `color` LaTeX package

In Active-DVI, colors can be specified with the conventions of the LaTeX package `color.sty`, that is, it should either be a previously defined color or a specification of the form `[`⟨*model*⟩`]` `{`⟨*model color specification*⟩`}`.

For example, the following specifications are all correct:

```
\color{blue}
\color[named]{Yellow}
\color[rgb]{0.7,0.3,0.8}
```

Named colors

Colors can be named using the keyword ⟨*named*⟩. If you use named colors, the color names are case sensitive and should generally be capitalized; for instance: `\color[named]{White}` specifies the white color. Hence, `\color[named]{Red}`{*some text*} writes *some text* in red.

The names of available colors can be found in the `dvipsnam.def` file, generally at location `/usr/share/texmf/tex/latex/graphics/dvipsnam.def`.

To give an idea, the names and colors available on a standard installation of LaTeX are:

GreenYellow Yellow Goldenrod Dandelion Apricot
Peach Melon YellowOrange Orange BurntOrange
Bittersweet RedOrange Mahogany Maroon BrickRed
Red OrangeRed RubineRed WildStrawberry Salmon
CarnationPink Magenta VioletRed Rhodamine Mulberry
RedViolet Fuchsia Lavender Thistle Orchid DarkOrchid
Purple Plum Violet RoyalPurple BlueViolet
Periwinkle CadetBlue CornflowerBlue MidnightBlue
NavyBlue RoyalBlue Blue Cerulean Cyan ProcessBlue
SkyBlue Turquoise TealBlue Aquamarine BlueGreen
Emerald JungleGreen SeaGreen Green ForestGreen
PineGreen LimeGreen YellowGreen SpringGreen
OliveGreen RawSienna Sepia Brown Tan
Gray Black

The CMYK specifications of colors

You may also explicitly use a CMYK (Cyan, Magenta, Yellow, Black) specification. In this case the cyan, magenta, yellow and black values follow the ⟨*cmyk*⟩ keyword, and are given as a list of four integers in the range 0.0 .. 1.0. For instance, `\color[cmyk]{0,1,0,0}` is a valid specification for magenta.

The RGB specifications of colors

RGB (Red, Green, Blue) specifications are similar to the CMYK specifications: following the ⟨*rbg*⟩ keyword, the red, green, or blue color values, are given as floating point numbers in the range 0.0 .. 1.0. Hence, `\color[rgb]{1.0,0.0,0.0}` is a valid specification for red.

The X -Window System colors

Active-DVI provides the package `xwindows-colors`, an extension to the `color` package, that defines a large set of the X Window System color names, as found in the file `rgb.txt` of a typical X installation (this file is generally located on `/usr/X11R6/lib/X11/rgb.txt`). To know which colors are available look at the source file of the package `xwindows-colors.sty` in the directory `tex` of the distribution.

6.7 Background

You can modify the background of your presentation in the LaTeX source of the pages. Background can be defined either as a plain color, as an image, or as a gradient (or as a combination of these!).

You can specify a global option to the background settings, so that these settings are used for the remaining pages of the presentation (otherwise the presenter resets the background options at each new page).

To modify the background of your presentation, you can:

- define the background color,
- define a gradient function to be run on the background (or a defined area of the background) using the predefined color gradients,
- add a background image (which can be alpha-blended on top of the background color).

If these options are used *together*, they are applied in this order: first the solid background color is drawn, then the gradient function is applied, finally the image is drawn on the resulting background.

`\advibg[global]{`⟨*decl*⟩`}`

where ⟨*decl*⟩ is a list of settings of the following from:

`color=`⟨*color*⟩ (default value is `none`)

Set the background color to ⟨*color*⟩. If ⟨*color*⟩ is `none` this unsets the background color. Otherwise, ⟨*color*⟩ must follow the notation above to designate colors.

`image=`⟨*file*⟩ (default value is `none`)

Use the image found in ⟨*file*⟩ as background (`none` means unset).

`fit=`⟨*fit style*⟩ (default value is `auto`)

Fit the background image according to ⟨*style*⟩, which may be one of the following keywords:

		`topleft`	`top`	`topright`
`auto`	or	`left`	`center`	`right`
		`bottomleft`	`bottom`	`bottomright`

The `auto` fit style means scaling the image as desired in both directions so that it fits the entire page. Other styles only force the same scaling factor in both directions:

- Corner-styles means set the image in the corresponding corner and scale it to cover the entire page.
- `center` means set the image in the center of the page and scale it to cover the entire page.
- Segment-styles means adjust the image and the page on the segment (in which case, the image may not completely cover the page on the opposite side).

`alpha=`⟨*float*⟩ (default value is `none`)

Set the alpha channel factor for the background image to ⟨*float*⟩ (`none` means unset). An alpha factor of 0 means that the image is not visible at all; conversely, an alpha factor of 1 means that the image covers the background.

`blend=`⟨*blend mode*⟩ (default value is `none`)

Set the blend mode to ⟨*blend mode*⟩, which are reminiscent of Ghostscript blending options. The blend mode should be one of the following: `normal`, `multiply`, `screen`, `overlay`, `dodge`, `burn`, `darken`, `lighten`, `difference`, `exclusion`, (`none` means unset).

`gradient=`⟨*function*⟩ (default value is `none`)

Set the gradient function to ⟨*function*⟩, one of the predefined functions that convert the plain background color into a color gradient from the chosen color `colorstart` to the color `colorstop` (which is white by default). Available gradients are:

- `hgradient` horizontal gradient (the gradient is a serie of vertical lines),
- `vgradient` vertical gradient (the gradient is a serie of horizontal lines),

- `d1gradient` first bissector gradient (the gradient is a serie of lines which are parallel to the first bissector),
- `d2gradient` second bissector gradient (the gradient is a serie of lines which are parallel to the second bissector),
- `cgradient` centered gradient (the gradient is a serie of concentric squares),
- `circgradient` circular gradient (the gradient is a serie of concentric circles).

`colorstart=`⟨*color*⟩ (default value is `white`)

Set the starting color of the gradient. When left unspecified defaults to white.

`colorstop=`⟨*color*⟩ (default value is `background`)

Set the end color of the gradient. When left unspecified defaults to the background color.

`xstart=`⟨*int*⟩ (default value is `0`)

Set the abscissa of the lower left point of the area where the gradient is drawn.

`ystart=`⟨*int*⟩ (default value is `0`)

Set the ordinate of the lower left point of the area where the gradient is drawn.

`width=`⟨*float*⟩ (default value is `1.0`)

Set the width of the area where the gradient is drawn. The width is a number in the range `[0 .. 1]` that gives the ratio of the area width with respect to the page width (hence 0.0 means a null width and 1.0 means the entire page width).

`height=`⟨*float*⟩ (default value is `1.0`)

Set the height of the area where the gradient is drawn. The width is a number in the range `[0 .. 1]` that gives the ratio of the area height with respect to the page height (hence 0.0 means a null height and 1.0 means the entire page height).

`xcenter=`⟨*float*⟩ (default value is `0.5`)

For a centered or circular gradient, set the abscissa of the center point of the gradient into the gradient area. `xcenter` is a ratio of the gradient area's width. It defaults to 0.5, meaning the middle of the gradient area width.

`ycenter=`⟨*float*⟩   (default value is `0.5`)

For a centered or circular gradient, set the ordinate of the center point of the gradient into the gradient area. `ycenter` is a ratio of the gradient area's height. It defaults to 0.5, meaning the middle of the gradient area height.

`none`

Unset all background parameters. This key must appear on its own, no arguments or keys are allowed.

The optional parameter `global` indicates that the definition is global and will affect the following pages, as well as the current page.

By default, the background settings only affect the current page.

6.8 Transitions

`\advitransition[global]{`⟨*decl*⟩`}`

where ⟨*decl*⟩ is a list of settings of the following from:

`none` or `slide` or `block` or `wipe`

Set the transition mode to the corresponding key. One of this key is mandatory (if several are provided the last one is selected).

`from=`⟨*direction*⟩

Make the transition come from ⟨*direction*⟩. Directions should be one of the following:

`topleft`	`top`	`topright`
`left`	`center`	`right`
`bottomleft`	`bottom`	`bottomright`

The default direction, to be used when no local or global direction has been specified, is determined dynamically: `right` when coming from previous page, `left` when coming from next page, and `top` otherwise.

`steps=`⟨*n*⟩

Make the transition in ⟨*n*⟩ steps.

As for `\advibg`, the optional parameter `global` indicates that the definition is global and will affect the following pages, as well as the current page.

By default, the transition definitions affect the current page only.

`\advitransbox`{⟨*key=val list*⟩}{⟨*hbox material*⟩}

where ⟨*key=val list*⟩ is as above and {⟨*hbox material*⟩} is whatever can follow an `\hbox` command. In particular, the material may contain verbatim commands, since as for the `\hbox` it is parsed incrementally.

The optional parameter `global` indicates that the definition is global and will affect the following pages, as well as the current page.

By default, the transition affects the current page only.

6.9 Embedded applications

To animate your show, Active-DVI can launch arbitrary applications you need.

6.9.1 Launching embedded applications

The LaTeX command to launch an application during the presentation is

`\adviembed`[⟨*key=value list*⟩]{⟨*command*⟩}

where ⟨*key=value list*⟩ is a list of bindings of the following kind:

`name=`⟨*name*⟩

Allows to refer to the embedded application as ⟨*name*⟩. Anonymous applications have actually the default name `anonymous`.

`ephemeral=`⟨*name*⟩

This is the default case: the application is specific to a given page. An `ephemeral` application is automatically launched whenever the page is displayed, and automatically killed when the page is turned.

`persistent=`⟨*name*⟩

A `persistent` application is launched only once and keeps running in the background; however, Active-DVI automatically hides and shows the window where the application runs, so that the application is visible only on the page where it has been launched.

`sticky=`⟨*name*⟩

A `stiky` application is launched only once, keeps running, and remains visible when turning pages. It is also resized and moved as necessary to fit the page size.

`raw=`⟨*name*⟩

A `raw` application is launched each time its embedding command is encountered. A `raw` application is not managed *automatically* by Active-DVI, except for the initial launching and the final clean-up that occurs when Active-DVI exits; hence, you can completely monitor the `raw` applications graphical behavior, using the `advikillembed` command and the window mapping facilities for `raw` applications described below.

`width=`⟨*dim*⟩
`height=`⟨*dim*⟩

The application takes ⟨*dim*⟩ width (respectively height) space in LaTeX. Both values default to 0pt.

These dimensions are also substituted for all occurrences of `@g` in the command string.

6.9.2 Monitoring embedded applications

To monitor embedded applications, Active-DVI provides the `advikillembed` primitive to send a signal to any named embedded application. For `raw` applications, there are additional functions to map or un-map the window allocated to a named `raw` application. Mapping or un-mapping windows of non-`raw` applications is unspecified, since it may interfere in a non trivial way with Active-DVI's automatic treatment of those applications.

Monitoring a single application

`\advikillembed`{⟨*name*⟩}

Kill the embedded application named ⟨*name*⟩. An optional signal value or symbolic name can be given to send to the designed process: for instance, `\advikillembed[SIGUSR1]{clock}` will send the `SIGUSR1` signal to the embedded application named `clock`.

Signal value defaults to -9.

`\advimapembed`{⟨*name*⟩}

Map the window of the (`raw`) embedded application named ⟨*name*⟩.

`\adviunmapembed`{⟨*name*⟩}

Un-map the window of the (`raw`) embedded application named ⟨*name*⟩.

Monitoring a group of embedded applications

The primitives `advikillallembed`, `advimapallembed`, and `adviunmapallembed` behave the same as their non-`all` counterparts, except that they operate on all the applications that have been launched with the given name.

`\advikillallembed{`⟨*name*⟩`}`

> Similar to `advikillembed` but kill all the embedded applications named ⟨*name*⟩.

`\advimapallembed{`⟨*name*⟩`}`

> Map the windows of all the (`raw`) embedded applications named ⟨*name*⟩.

`\adviunmapallembed{`⟨*name*⟩`}`

> Un-map the windows of all the (`raw`) embedded applications named ⟨*name*⟩.

6.10 Active anchors

Active anchors are annotated pieces of text that get associated activation records. To define an active anchor, the command is

`\advianchor[`⟨*decl*⟩`]{`⟨*tag*⟩`}{`⟨*text*⟩`}`
`\begin{advianchor}[`⟨*decl*⟩`]{`⟨*tag*⟩`}` ⟨*text*⟩`\end{advianchor}`

> The text is first displayed as usual, then the anchor is drawn according to the style given by ⟨*decl*⟩, and made active. Its activation, which depends on the mode given by ⟨*decl*⟩, will play the record named ⟨*tag*⟩.
>
> The declarations ⟨*decl*⟩ are of the following form:
>
> **`over`, `click`, or `stick`**
> The mode `stick` plays the tag ⟨*tag*⟩ on click. The mode `click` is similar, except that it restores the previous state when leaving the anchor area. The mode `over` is as `click` but display the ⟨*tag*⟩ when the mouse is `over` the anchor instead of waiting for a click.
>
> **`box`, `invisible`, or `underline`**
> this defines the style in which the anchor should be drawn. The default style is `box`.
>
> In the environment form, ⟨*text*⟩ may contain fragile commands.

`\adviemphasize[`⟨*color*⟩`]{`⟨*text*⟩`}`

> This makes an invisible anchor around ⟨*text*⟩, which when activated will redraw text in a box colored with ⟨*color*⟩, which defaults to `yellow`.

6.11 Postscript specials

Active-DVI can deal with most of PStricks by calling `ghostscript` on included Postscripts. Basic change of coordinates are implemented, but this feature remain fragile, as Active-DVI must in turn call `ghostscript` to get the new coordinates. Also, rotations will definitely not work for text, which is rendered by Active-DVI and cannot be rotated.

6.11.1 Overlays

The `overlay` class implements overlays with PStricks. By contrast, Active-DVI implements overlays directly, using records and plays. This is more efficient, and of course more natural. (In fact, Active-DVI chooses the cumulative semantics of overlays, displaying all layers below the current overlay.)

The `xprosper` style, derived from the `prosper` class, uses the `overlay` class and works with Active-DVI in exactly the same way (relaxing the `\overlay@loop` macro inhibits all layers, but the first page).

6.11.2 PStricks known to work

Active-DVI can deal with main PStricks. In particular, the following work

+ Simple drawings, such as `\psframe`, `\ovalnode`, ...

+ Connections between nodes `\ncline`, `\ncarc`, also works.

+ Labels over arrows `\Aput`, `Bput`, etc.

+ `\SpecialCoor`, *i.e.* commands of the form `\rput(A){bla bla}` works where `A` is a node name.

+ Embeddded-Postcript figures, including scaling.

Other PStricks may or may not work.

7 The `advi-slides.sty` LaTeX package

Active-DVI provides this specialized LaTeX package to facilitate writing presentation slides in the spirit of SliTeX. See examples in the directory `examples/slitex`.

8 Auxiliary LaTeX packages

8.1 The `superpose` package

This package allows superposition of horizontal material, creating the smallest horizontal box that fits all of the superpositions.

`\usepackage{superpose}`

The package defines a single environment:

`\begin{superpose}[`⟨*alignment*⟩`]`⟨*list*⟩`\end{superpose}`

> The ⟨*alignment*⟩ can be one the letters `c` (default value), `l`, or `r`.
>
> Items of the ⟨*list*⟩ are separated by `\\` as in tabular environments. Each item should be a horizontal material.

8.2 The `bubble` package

This package draws bubbles over some text.

```
\usepackage{bubble}
\usepackage[ps]{bubble}
```

By default bubbles are produced using the `epic` and `eepic` packages, for portability. However, for better rendering and easier parameterization, bubbles can also be drawn using the `pst-node` package of the PStricks collection. This is what the `ps` option is designed for.

The package defines a single command:

`\bubble[`⟨*key=value list*⟩`]{`⟨*anchor*⟩`}[`⟨*ps options*⟩`](`⟨*pos*⟩`){`⟨*text*⟩`}`

> The ⟨*key=value list*⟩ is a list of bindings of the following kind:
>
> `bg=`⟨*color*⟩ (default value is `yellow`)
>
> > The background color for annotations.
>
> `unit=`⟨*dim*⟩ (default value is `yellow`)
>
> > Set the package unit to ⟨*dim*⟩.
>
> `col=`⟨*colspec*⟩ (default value is `c`)
>
> > Where ⟨*colspec*⟩ is a column specification for the tabular environment. Moreover, the following abbreviations are recognized:

key	expands to
`c`	`col=c`
`l`	`col=l`
`r`	`col=r`
`p=`⟨*w*⟩	`col=p{`⟨*w*⟩`}`

key	expands to
`C`	`col={>{$}c<{$}}`
`L`	`col={>{$}l<{$}}`
`R`	`col={>{$}r<{$}}`
`P`⟨*w*⟩	`col={>{$}p{`⟨*w*⟩`}<{$}}`

> ⟨*pos*⟩ is the optional relative position of the annotation, it defaults to $1, 1$, and is counted in the package units.
>
> ⟨*ps options*⟩ are passed to the command `\psset`) in `ps` mode and ignored otherwise.

Parameters (color and tabular columns specifications) can also be set globally using the command:

`\bubbleset`{⟨*key=value list*⟩}

8.3 The `advi-annot` package

This package uses active anchors and the bubbles package to provide annotations by raising a bubble when the cursor is over the anchor.

The package defines a single command

`\adviannot`[⟨*key=value list*⟩]{⟨*anchor*⟩}[⟨*ps options*⟩](⟨*pos*⟩){⟨*text*⟩}

> whose options are identical to those of the `\bubble` macro; however the bubble appears within an active anchor.

8.4 The `advi-graphicx` LaTeX package

This 3-lines long package loads the `graphicx.sty` package and provides declarations so that JPEG, EPS, TIF, TIFF source images can be embedded: Active-DVI will preview these images directly while other drivers will translate them on demand.

A Limitations

Postscript Fonts

Postscript fonts are not natively handled by Active-DVI. You must use the command `dvicopy` to expand those virtual fonts to base fonts before visualization with Active-DVI. (For instance, `dvicopy talk.dvi talk.expanded; advi talk.expanded` very often does the trick.)

In-lined Postscript and Ghostscript

PS relies on `ghostscript` to display Postscript in-lined specials. However, some earlier releases of `ghostscript` implements the Postscript `flushpage` command as a `XFlush` call which does not force the evaluation of commands, and thus makes the synchronization between `ghostscript` and Active-DVI drawings uncontrollable. In this case, the interleaving of in-lined postscript and other material may be inconsistent.

Fortunately, recent versions of ghostscript (> 6.5) have fixed this problem by using `XSync(false)` instead. If you use those versions of ghostscript, in-lined specials should be correctly rendered.

Unfortunately, some releases of version 6.5x also carry a small but fatal bug for Active-DVI, that will hopefully be fixed in future releases. A workaround is available here `http://cristal.inria.fr/~remy/ghostscript/`.

In-lined Postscript change of coordinates

So far, the implementation of in-lined Postscript does not correctly handle complex change of coordinates. (See PStricks section).

B Reporting bugs

Please, send bug reports to `mailto:advi@inria.fr`.
See `http://gallium.inria.fr/advi` for up to date information.

C Key bindings

Active-DVI recognizes the keystrokes listed below when typed in its window. Some keystrokes may optionally be preceded by a number, called arg below, whose interpretation is keystroke dependant. If arg is unset, its value is 1, unless specified otherwise.

Active-DVI maintains an history of previously visited pages organized as a stack. Additionnally, the history contains marked pages which are stronger than unmarked pages.

Survival command kit

?	info	–	This quick info and key bindings help.
q	quit	–	End of show.
space	continue	–	Move forward (arg pauses forward if any, or do as return otherwise).
^X-^C	quit	–	End of show.

Moving between pages

n	next	–	Move arg physical pages forward, leaving the history unchanged.
p	previous	–	Move arg physical pages backward, leaving the history unchanged.
,	begin	–	Move to the first page.
.	end	–	Move to the last page.
g	go	–	If arg is unset move to the last page. If arg is the current page do nothing. Otherwise, push the current page on the history as marked, and move to physical page arg .

Moving between pauses

N	next pause	–	Move arg pauses forward (equivalent to continue).
P	previous pause	–	Move arg pauses backward.

Adjusting the page size

^X-^F	set fullscreen	–	Adjust the size of the page to fit the entire screen.
^F	toggle fullscreen	–	Adjust the size of the page to fit the entire screen or reset the page to the default size (this is a toggle).
<	smaller	–	Scale down the resolution by scalestep (default $\sqrt{\sqrt{\sqrt{2}}}$).
>	bigger	–	Scale up the resolution by scalestep (default $\sqrt{\sqrt{\sqrt{2}}}$).
#	fullpage	–	Remove margins around the page and change the resolution accordingly.
c	center	–	Center the page in the window, and resets the default resolution.

Moving the page in the window

h	page left	–	Moves one screen width toward the left of the page. Does nothing if the left part of the page is already displayed
l	page right	–	Moves one screen width toward the right of the page. Does nothing if the right part of the page is already displayed
j	page down	–	Moves one screen height toward the bottom of the page. Jumps to the top of next page, if there is one, and if the bottom of the page is already displayed.
k	page up	–	Moves one screen height toward the top of the page. Jumps to the bottom previous page, if there is one, and if the top of the page is already displayed.

^left button	move page	–	A black line draws the page borders; moving the mouse then moves the page in the window.
^C	toggle center on cursor	–	Toggles `center-on-cursor` flag, which when sets moves the screen automatically so that the cursor appears on the screen.

Switching views

w	switch	–	Switch view between master and client (if any).
W	sync	–	Goto page of client view corresponding to page of master view.
^W	autoswitch	–	Toggle autoswitch flag.

Redisplay commands

r	redraw	–	Redraw the current page to the current pause.
R	reload	–	Reload the file and redraw the current page.
^L	redisplay	–	Redisplay the current page to the first pause of the page.
a	active/passive	–	toggle advi effects (so that reloading is silent).

Using the navigation history stack

return	forward	–	Push the current page on the history stack, and move forward n physical pages.
tab	mark and next	–	Push the current page on the history as marked, and move forward n physical pages.
backspace	back	–	Move arg pages backward according to the history. The history stack is poped, accordingly.
escape	find mark	–	Move arg marked pages backward according to the history. Do nothing if the history does no contain any marked page.

Table of contents

T	Thumbnails	–	Process thumbnails.
t	toc	–	Display thumbnails if processed, or floating table of contents if available, or else do nothing.

Writing and drawing on the page

s	write	–	Give a pencil to scratch, typing characters on the page.
S	draw	–	Give a spray can to scratch, drawing on the page.
?	info	–	While in scratch mode, press ? for more info.

Using the laser pointer

ˆX-l	toggle laser	–	Toggle the laser beam to point on the page.
ˆG	laser off	–	When laser is on turn it off.

Saving the current page

ˆX-ˆS	save page	–	Save the current page as an image file.

Dealing with caches

f	load fonts	–	Load all the fonts used in the document. By default, fonts are loaded only when needed.
F	make fonts	–	Does the same as f, and precomputes the glyphs of all the characters used in the document. This takes more time than loading the fonts, but the pages are drawn faster.
C	clear	–	Erase the image cache.

D Key bindings for scratch writings

Entering scratch writing mode

Press s to enter scratch writing; the cursor is modified and you must click somewhere on the page to start writing text there. Before clicking, you can

- press ? to get help,
- press ˆG to quit scratching immediately,
- press Esc to enter the scratch writing settings mode and tune the font and font size.

Survival command kit when scratch writing

Active-DVI recognizes the following keystrokes when scratch writing on the page.

ˆG	quit	–	End of scratch writing.
Esc	settings	–	Enter the scratch writing settings mode.

In the scratch writing settings mode, the cursor is modified and you can set some charateristics of the scratch writing facility. When in doubt, press

- press ? to get help,
- press ˆG to quit scratching immediately,
- press Esc to quit the setting mode.

Scratch writing settings mode keys

When in the scratch writing settings mode, the following keys have the following respective meanings:

>	greater	–	Increments the scratch font size.
<	smaller	–	Decrements the scratch font size.
b	blue	–	Set the color of the font to blue.
c	cyan	–	Set the color of the font to cyan.
g	green	–	Set the color of the font to green.
k	black	–	Set the color of the font to black.
m	magenta	–	Set the color of the font to magenta.
r	red	–	Set the color of the font to red.
w	white	–	Set the color of the font to white.
y	yellow	–	Set the color of the font to yellow.
B	more blue	–	Increment the blue component of the color.
G	more green	–	Increment the green component of the current color.
R	more red	–	Increment the red component of the current color.
+	positive increment	–	Set the color increment to positive.
–	negative increment	–	Set the color increment to negative.
?	help	–	Give the list of settings available.
Esc	quit	–	Quit the sratch writing settings mode.

Setting the scratching font size

Just press Esc to enter the scratch writing settings mode, then > or < to increment or decrement the font size; then press Esc again, to leave the scratch writing settings mode and continue to write on the page with the new font size.

E Key bindings for scratch drawings

Entering scratch drawing mode

Press S to enter scratch drawing; the cursor is modified and you must click somewhere on the page to start drawing there. Before clicking, you can

- press ? to get help,
- press ˆG to quit scratching immediately,
- press Esc to enter the scratch drawing settings mode and tune the color and size of the pen.

Survival command kit when scratch drawing

Active-DVI recognizes the following keystrokes when scratch drawing on the page.

^G	quit	–	End of scratch drawing.
Esc	settings	–	Enter the scratch drawing settings mode.

In the scratch drawing settings mode, the cursor is modified and you can set some charateristics of the scratch drawing facility.

Scratch drawing settings mode keys

When in the scratch drawing settings mode, the following keys have the following respective meanings:

General scratch drawing settings keys

- press ? to get help,
- press Esc to quit the settings mode,
- press ^G to quit scratching immediately.

Setting the drawing line color

b	blue	–	Set the color of the font to blue.
c	cyan	–	Set the color of the font to cyan.
g	green	–	Set the color of the font to green.
k	black	–	Set the color of the font to black.
m	magenta	–	Set the color of the font to magenta.
r	red	–	Set the color of the font to red.
w	white	–	Set the color of the font to white.
y	yellow	–	Set the color of the font to yellow.
B	more blue	–	Increment the blue component of the current color.
G	more green	–	Increment the green component of the current color.
R	more red	–	Increment the red component of the current color.
+	positive increment	–	Set the color increment to positive.
−	negative increment	–	Set the color increment to negative.

Setting the drawing line size

>	increment	–	Increment by one the size of the line.
<	decrement	–	Decrement by one the size of the line.

Setting the kind of figure to draw

In the setting mode, pressing one of the following keys enter the (still experimental) figure drawing mode:

V	vertical line	–	Draw a vertical line.
H	horizontal line	–	Draw a horizontal line.
S	segment	–	Draw a segment.
C	circle	–	Draw a circle.
p	point	–	Draw a point.
P	polygone	–	Draw a polygone.
e	endpoly	–	Close the polygone that is beeing drawn.
F	free hand	–	Draw a line following the pointer.
' '	cancel	–	Cancel the figure setting.

F Index