

L e c t u r e r

Author: Paul Isambert
zappathustra@free.fr
Date: 07/23/2010
Version: 1.0

Lecturer creates presentations mainly based on PDF features. It doesn't manipulate T_EX's typesetting process in a complex way to give the impression that slides are being layered, but use PDF's so-called optional content to produce exactly this: layers. It was originally meant to work with plain T_EX, but since writing for plain is one step away from writing for any format, it is format-independent, though the heavy hand it lays on the output routine might make it conflict with other code. In plain T_EX, things are innocuous; in L_AT_EX, though not thoroughly tested, *Lecturer* works good, provided it is not in conflict with other packages; since it is pretty self-contained for what it's designed, things should work nicely. Finally, *Lecturer* seems to be usable with ConT_EXt MkII, but not MkIV (except perhaps to make a handout). I welcome comments on use of *Lecturer* in those formats, so I can improve it.

As for engines, *Lecturer* currently works with pdfT_EX and LuaT_EX only, because the code is dependent on pdfT_EX's primitives. My apologies to XeT_EX users. I welcome drivers.

The main drawback in using *Lecturer* is it requires a 'conforming reader,' as they say at Adobe, to work properly, that is a reader that understands advanced PDF code and a little bit of JavaScript. Which basically means Adobe's reader, available mostly everywhere, and free, but 'as in free beer,' to make a quote.

The main advantage is that typesetting (T_EX) and presentation flow (PDF) are neatly divided: each slide is a page *typeset* by T_EX and *manipulated on the screen* by PDF code. Beside the intellectual satisfaction, there are more direct benefits: for instance, the handout is already in the slide, even if it's not directly prepared as such. Better still, there's no need to prepare the manuscript in any way to adapt to the requirements of a presentation. A list or a table, for instance, can be displayed stepwise without changing the usual way you type them in: interspersed `\step` commands will do.

And the best feature in *Lecturer*, as far as I'm concerned, is that you can put anything anywhere with a few commands.

If you don't feel like reading 30 pages of documentation, switch to the demonstrations distributed with Lecturer. For an overview, go to the [Demonstrations](#) section.

Basic things

Loading

Lecturer should be loaded as is customary in the format at work, i.e. in plain TeX:

```
\input Lecturer
```

in LaTeX:

```
\usepackage{Lecturer}
```

and in ConTeXt:

```
\usemodule[Lecturer]
```

First things first

In *Lecturer*, the content of a document is organized in slides, with steps to make things appear progressively on the screen when you're in fullscreen mode. A (default) slide is anything that appears between the following two commands:

```
\slide [<attributes>]
\endslide
```

In LaTeX you can use `\begin{slide}` and `\end{slide}`, and in ConTeXt `\startslide` and `\stopslide`. The `\slide/\endslide` pair can be used in all formats, though, which is why I'll use `\slide` and `\endslide` for the examples in this documentation.

```
\step [<attributes>]
```

To make the content of a slide appear progressively on the screen, this command is used where necessary. By default, it does nothing else, i.e. it doesn't disturb typesetting and simply creates the flow of the presentation with PDF code. Here's an example:

```
\slide
Something visible.
\step My first proposition.

\step And then... \step this.
\endslide
```

which on the screen produces a paragraph with *'Something visible. My first proposition.'* and another one with *'And then... this'* as is usual with TeX. Except these paragraphs don't appear at once.

A `\step` means you'll have to click to move on, and what follows it will appear on screen, whereas it is otherwise invisible (by default at least). In the example, *'Something visible'* is

To advance in the presentation, use either a left click or a right arrow key; to move backward, use a right click or a left arrow, and you can also use `\nextstep` and `\prevstep` with buttons

already on the screen, since it is not in the scope of a `\step` command, but the rest appears piecewise, and it takes three clicks to reach the end of the slide. Conversely, when moving backward, either with a right click or a left arrowkey, steps disappear.

The `\step` command affects the content of what follows only, up to the next `\step` or to the slide's end. Something between the beginning of the slide and the first step is always visible.

Everything between `\slide` and `\endslide` is only one page. Steps appear gradually because they are instructed to do so, but they are on the same page, as physically as is possible for a collection of bytes (which can be printed, though).

`\slide`

`\slide` number The first of those is a count register holding the slide's number (i.e. `\count0`). The second is a macro that returns its value.

Now you know the best part of how things work. The rest, i.e. the optional `<attributes>` and a bunch of other things, is only decoration, although it might be of interest too to enliven your slides.

Presentation and handout

`Lecturer` allows you to easily produce a presentation for screen and a handout out of the same file. The output depends on the value of the `mode` attribute in the `job` parameter. If it is set to `presentation`, then steps behave as defined above. If it is set to `handout`, then only those steps that have the value `true` for the `handout` parameter appear on the page. Other steps are still present, but hidden; they can be turned on in the reader's layers panel, though. By default, all steps are visible in a handout. The following three commands take a more radical way.

See *Attributes for the job*

`\presentationonly` `<material>`

In presentation mode, `<material>` is simply typeset. In handout mode, however, it is gobbled. Thus, it takes no space on the page, and can't be retrieved.

`\handoutonly` `<material>`

The same thing the other way around.

`\presentationorhandout` `<presentation><handout>`

In `presentation` mode, the first argument is executed, and the second one in `handout` mode. Note the difference between

```
\step A\step[handout=false] B\step C
```

and

```
\step A\presentationonly{\step B}\step C
```

In presentation mode, both will produce 'ABC'. In handout mode, though, the first example will produce 'A C', with 'B' still retrievable from the layers panel, and the second example will produce 'AC', with no 'B' anywhere.

Both `\presentat iononly` and `\handoutonly` ignore any space following their argument when they gobble it.

In Adobe Reader, the layers panel is in the navigation panel on the left, represented by two squares overlapping, usually below thumbnails and bookmarks

Settings things globally and locally

Slides and steps can have attributes that modify their behavior. These attributes can be set locally or globally, meaning they either affect the current slide/step or all slides/steps or all slides/steps belonging to a certain type defined with `\setslide`/`\setstep`.

The way attributes are used is a matter of inheritance. At the top of the hierarchy, there are the `slide` and `step` parameters. Attributes to those parameters apply to all slides and steps, unless they are given other values lower in the hierarchy. At the bottom of the hierarchy, each individual slide or step can be followed by attribute settings between brackets. The values given there override all inherited values, but they hold for the current slide or step only. In between, new types of slides and steps can be defined; they inherit attributes from the corresponding parameters, and transmit their own attributes to slides and steps of their kind.

Global settings

`\setParameter <parameter>`:

This is used to set the values to attributes for `<parameter>`. The syntax for the attributes, which is used in many other places in *Lecturer*, is as follows:

```
\setParameter parameter:  
  attribute1 = valueA  
  attribute2 attribute3 = valueB  
  ...  
\par
```

There are only three parameters in Lecturer: job, slide, and step. They're used to build a presentation, and don't need to be created beforehand. Nonetheless, Yax's parameter syntax is used everywhere

You might find this syntax somewhat bizarre, but it is handy. It is the Yax syntax, and if you want to know more about it, see the Yax documentation. Here's how it works: after `\setParameter` you type either `slide` or `step` (or `job`, as we'll see later). Then a colon, which might be preceded by a space. Then you type the name of an attribute, or the names of several attributes separated by a space, and then a '=' sign. Finally you give your value, which is ended by the following space. If a value must contain a space, or if it's empty, you must give it between braces or double quotes, as in:

```
\setParameter step:
  attribute1 = "my value"
  attribute2 = {}
  ...
```

Or, if a value begins or ends with a control sequence, it should be enclosed between braces or double quotes again, unless it is made of a single control sequence. Thus:

If you have any doubt, you can always rely on braces and quotes

```
\setParameter step:
  attribute3 = "\command{argument}"
  attribute4 = \control sequence
  ...
```

Finally, and most importantly, a `\setParameter` declaration is ended by `\par` which, and that's the cool thing, can be a blank line, since \TeX inserts a `\par` at each blank line.

The `\setParameter` declaration can be used as often as you wish; there's no need to respecify each attribute on each declaration: they retain their last declared value, which *Lecturer* uses when it needs it.

Local settings

Attributes are set locally when they are given between brackets after the `\slide` or `\step` commands. They will be used for this particular `\slide` or `\step`. The syntax is the custom `key=value` fashion: first the attribute, then the value, then a comma. Here you can forget what's just been said about values: no need for quotes or braces. On the other hand, you can't set several attributes at once, each one must take its own value.

Quotes and braces are removed anyway. So you can use them too, although they're completely useless: the delimiter is the comma.

Thus the following step

```
\step[ myatt = myval ] And now something terrific...
```

will have the value `myval` for the attribute `myatt` (note how unwanted space is ignored), instead of the value given in the `\setParameter` declaration... or in its type, which we turn to right now (and then I promise we'll see what those attributes are). Any space after the right bracket is ignored, so that you don't have to put `%` signs here and there.

Type settings

You can create new types of slides or steps and set their parameters with the `\setslide` and `\setstep` commands respectively. Note that there is no difference between creating and setting the attributes of a new slide/step, and simply setting the attributes to an already created slide/step: the same commands are used in both cases.

```
\setslide <list of slides>
```

The `\setslide` command sets parameters for all slides in the space-separated `<list of slides>` (and creates them if they don't exist yet), as in:

```

\setslide{myslide myotherslide}
  attribute1 = valueA
  attribute2 = valueB
  ...
\par

```

To use the `slide` command, in plain TeX you call `\myslide` and `\endmyslide`, in LaTeX you do `\begin{myslide}` and `\end{myslide}`, and in ConTeXt you use `\startmyslide` and `\stopmyslide`. Actually `\myslide` and `\endmyslide` work in all three formats, but they're not customary.

Slides thus created inherit values from the `slide` parameter, except when new values are specified. In our example, `\myslide` and `\myotherslide` will have `valueA` and `valueB` for `attribute1` and `attribute2` respectively, instead of the values defined for those attributes in the `\setparameter` declaration. Both commands can still be followed by local settings between brackets, and they'll override the ones given here. So in

```

\myslide[attribute2=myval]
...
\endmyslide

```

`\myslide` will have the values `myval` for `attribute2`, `valueA` for `attribute1`, and the values defined in the `\setparameter` declaration for all the other attributes.

`\setstep <list of steps>`

`T` takes a list of commands, i.e. control sequences, as its first argument, and then attributes and values as in `\setparameter`. The commands are just concatenated, with nothing to separate them. Thus

```

\setstep{\mystep \myotherstep}
  attribute1 = valueA
  attribute2 = valueB
  ...
\par

```

creates or modifies `\mystep` and `\myotherstep`, which in a slide will create a step with `valueA` and `valueB` for `attribute1` and `attribute2` respectively, overriding the value of the `step` parameter. The commands can still take values between brackets to set attributes locally.

Attributes for slides

The attributes that follow are to be used in the `\setparameter` declaration for the `slide` parameter, in the `\setslide` declaration, and as optional arguments to the `slide` command and equivalent commands created with `\setslide`, for which I'll use `slide` generically.

PDF and navigation

First of all, if a `slide` command takes anything between brackets, and if among it there is something that doesn't contain a `'` sign, i.e. something that is obviously no attribute setting, then this thing, let's call it a string, becomes the slide's title, and it is recorded in a macro called `slide\title`, which you can use in the slide itself. It is also used in bookmarks (if the slide is to be bookmarked), which leads us to our first attribute.

`pdf\title` *<string>* (Default: slide's title)

Suppose you want a slide with title `Something about \TeX`. Then if that string is used in a bookmark, the bookmark won't look very good, because of the `\TeX` operations contained in the `\TeX` command, which bookmarks don't understand, since bookmarks are pretty limited. This attribute sets the title of the slide for use in bookmarks (and in the layers panel too). It doesn't make much sense except as a local attribute, unless you define it with a command whose definition changes from slide to slide, involving for instance a counter.

If a slide has no title, pdf or not, `Page n` is used in bookmarks and layers, but the `slide\title` command remains empty (not undefined).

Layers, i.e. steps, are grouped in the layers panel under the heading of the slide where they belong.

`bookmark` *<true|false>* (Default: true)

If this is set to `true`, a bookmark is created with the slide's title.

`bookmark\level` *<number>* (Default: 1)

The level of the slide's bookmark. The higher the `number`, the lower the slide in the hierarchy. The `number` needs not be an integer.

See [Creating bookmarks](#)

`bookmark\style` *<bookmark options>* (Default: none)

The style of the bookmark, i.e. the optional argument in the `\createbookmark` command.

`anchor` *<name>* (Default: none)

An anchor to navigate to the slide with a `\goto` command. This is the optional `<anchor>` in `\createbookmark`. Since two anchors can't have the same name, this attribute can be used with `slide` only, not in the `\setparameter` and `\setslide` declarations.

You can actually specify the anchor for several slides at once, provided the value is a command that holds a variable changing with slides.

Dimensions

`width` *<dimension>* (Default: 15cm)

The width of the slide.

`height` <di mensi on> (Default: 12cm)
The height of the slide.

`hsize` <di mensi on> (Default: slide's width minus twice its left)
The width of lines.

`left` <di mensi on> (Default: 2cm)
The left margin, i.e. the distance between the slide's left border and the textblock.

`right` <di mensi on> (Default: unspecified)
The right margin, i.e. the distance between the textblock and the slide's left border. If present, this parameter is always obeyed, in the following way: if neither `left` nor `hsize` is specified, `left` is set to the value of `right` and `hsize` is computed accordingly; if one is specified but not the other, the unspecified one is computed according to the specified one and `right`; and if both are present, `hsize` is ignored and the previous rule applies.

What is inherited from global to type to local settings is the result of this calculation, i.e. values for `left` and `hsize`. If these are changed `right` won't be taken into account. Conversely, if `right` is specified, then `left` and `hsize` aren't inherited. If things were otherwise, `left` would always be specified (since inherited) and it'd be impossible to compute it according to `right` and `hsize`, since it always wins against the latter. Thus, assuming a slide width of 10cm, with the following:

```
\setparameter slide:  
  left = 2cm  
  right = 1cm  
  
\setslide{slideA}  
  left = 1cm  
  
\setslide{slideB}  
  right = 2cm  
  hsize = 5cm
```

we have a default slide with a left margin 2cm wide, right margin 1cm, and a line length of 7cm. For `slideA` nothing is recomputed, thus the left margin is 1cm but the line length is still 7cm, which gives a right margin of 2cm. And `slideB` has a `right` attribute, which triggers a computation; `left` isn't inherited, thus only `hsize` is specified and `left` is computed accordingly (to 3cm, as you might imagine).

`vsize` <di mensi on> (Default: slide's height minus twice its top)
The height of the textblock. This isn't important for page breaks, since no page break occurs in *Lecturer*, but for the `vpos` attribute, and the scaling of the page if any.

Thus, to center the textblock on the slide, specify the right attribute only. The same holds for bottom to center vertically, and for areas it is also true of right, bottom, hshift and vshift*.*

top `<dimension>` (Default: 1cm)

The upper margin, i.e. the distance between the slide's top border and the textblock.

bottom `<dimension>` (Default: unspecified)

The bottom margin, i.e. the distance between the textblock and the slide's bottom border. Works exactly as right in the vertical dimension, i.e. it is always obeyed, but not inherited, etc.

Note that in full screen mode, readers set the page so that its smallest dimension goes from one side of the screen to the other. Thus, with

```
\setparameter slide:
  height = 12cm
\par
\setlide{myslide}
  height = 14cm
\par
```

`\slide` and `\myslide` will actually look like they have the same height but not the same width. And the type will appear smaller in `\myslide`. By the way, the dimensions 12cm and 15cm are default simply because they make T_EX's default 10pt Computer Modern look readable.

baselineskip `<dimension or glue>` (Default: 12pt)

The baseline distance for the slide. Glue means something like 2pt plus fill, which can be useful (see the `vpos` attribute below).

topskip `<dimension or glue>` (Default: 12pt)

The distance between the first line's baseline and the top of the textblock.

parskip `<dimension or glue>` (Default: 0pt)

The distance between paragraphs.

parindent `<dimension>` (Default: 0pt)

The width of the paragraph's indentation for this slide.

You might want the previous dimensions to be connected in some way. For instance, by default, `hsizex` is connected to the slide's `width` and `left`. This is because the latter two attributes set the T_EX dimensions that the `hsizex` attribute refers to. So the order in which those dimensions are set is important. Here are the dimensions and the attributes that set them, in the order of the assignment:

The slide's font, if any (see the `` attribute below), is called before these dimensions are set, so the order and the dimensions works properly.

```
\baselineskip = baselineskip
\topskip = topskip
\parskip = parskip
\pdfpagewidth = width
\pdfpageheight = height
```

```

\pdfhorigin=left
\pdfvorigin=top
\hsizewidth=hsizewidth
\parindent=parindent
vsize

```

Thus, the default value for the `hsizewidth` attribute is `width minus twice left` for each slide, because it is set as:

```
hsizewidth = "\pdfpagewidth-2\pdfhorigin"
```

Which leads to an important point: in *Lecturer*, a `<dimension>` can be an expression, since it is then embedded in a `\dimexpr` primitive.

`hpos` `<ff|fr|rf|rr>` (Default: `fr`)

This controls the slide's horizontal justification. The values mean: `ff` is flushleft/flushright (i.e. text justified on both sides), `fr` is flushleft/raggedright (commonly called raggedright), `rf` is raggedleft/flushright (called raggedleft), and `rr` is raggedleft/raggedright (i.e. text centered). Here second-order infinite glue is used.

`vpos` `<top|center|bottom>` (Default: `center`)

This sets how the lines are set in the textblock. Slides typically doesn't have the same number of lines, so we can't simply treat them as usual pages in typesetting. This is how the text of a slide is vertically positioned in the area defined by `vsize`. The `top` value means the text is flushed at the top, the `bottom` value is the same thing at the bottom, and `center` means the text will be centered in the area. This positioning is done with first-order infinite stretch, so you can use some too, or higher-order stretch, in the slide's content, to achieve some effects.

`scale` `<true|false>` (Default: `false`)

If the main content of a slide is higher than `vsize`, it is scaled to the desired dimension if this attribute is set to `true`. The bad news is it is also scaled horizontally, to keep proportions.

Content

`everyslide` `<code>` (Default: `nothing`)

Material inserted at the beginning of each slide (in vertical mode). For instance after

See [Adding material to areas](#)

```

\setslide{myslide}
everyslide = "\position{title}{\slide{title}}"

```

the title's slide will be inserted in the `title` area for each slide called with the `\myslide` command.

areas <list of areas> (Default: all)

This denotes the areas that may appear on a given slide. The keyword `all`, which is default, means that all areas will be painted (provided they are visible or contain material). Otherwise, `list of areas` enumerates allowed areas, separating them with space. If an area isn't painted on a slide but nonetheless contains material, this material is lost (and not even processed in the first place).

areas* <list of areas> (Default: none)

The list of excluded areas, i.e. the reverse of the previous attribute. Areas belonging to that list aren't painted on the slide. If an area appears in both `areas` and `areas*`, it is painted unless it is simply implicitly present in `areas` with `all`.

Style

background <named color or shade or color expression> (Default: white)

The background color of the slide.

See [Colors, shades and images](#)

foreground <named color or color expression> (Default: black)

The slide's foreground color.

image <image> (Default: none)

An image inserted in the slide's background, above the background color. The image is inserted at the slide's upper left corner, and has the width and height it was declared with. Since this parameter is inherited, to cancel it give as `<image>` any name that is not a declared image; *Lecturer* is in the middle of a shipout when it reads this attribute, so don't worry, you'll get no error message.

See [Images](#)

font (Default: job's font)

The font used to typeset the slide. This is actually just a placeholder for font commands, e.g. `\it` or `\bfseries`. If this attribute is used, it should be specified for all slide types, or the `job` parameter should have a value to its own `font` attribute. Otherwise, if a slide with a font information is followed by a slide without such information, the latter will use the font of the former.

transition <named transition> (Default: none)

This is the way the PDF reader will display, animate, enliven, and perhaps discredit your presentation. A transition is a little animation conforming readers play when they go from one slide to the next. They work in full screen only. A `named transition` should be recorded beforehand with `\newtransition` unless you use the default values, in which case you can use at once `split`, `blinds`, `box`, `wipe`, `dissolve`, `glitter`, `fly` (no less), `push`, `cover`, `uncover` and `fade`.

See [Transitions](#)

Attributes for steps

The attributes that follow are to be used in the `\setParameter` declaration for the `step` parameter, in the `\setstep` declaration, and as optional arguments to the `\step` command and equivalent commands created with `\setstep`, for which I'll use `\step` generically.

PDF and navigation

As with slides, an entry without a ``` sign in the optional argument to the `\step` command becomes the step's name (which should be unique on a given slide). This name is mainly useful for referencing with the `on` and `off` attributes and the `\showorhide` command, but it is also used as the step's name in the layers panel of the reader (if it has one). Hence:

`pdftitle <string>` (Default: step's default name)

Set the name used in the reader's layers panel. Thus you can have a nice title here and keep an uninformative-but-less-painful-to-type name to refer to a step in the attributes that follow. (If a step has no name, it is referred to in the layers panel with ``step n`` with `n` set back to zero at each slide.)

`on <list of steps>` (Default: none)

This sets the steps that make the current step appear. If there is no `on` attribute, a step appears at its position in the source. If there's an `on` attribute, then it appears when the steps referred to in the list are reached. If the list of steps contains the keyword `here`, then the step also appears by itself. Steps in the list are separated by space.

When moving backward, just like the controlling step will toggle its visibility, steps tied with `on` and `o` will too.

`off <list of steps>` (Default: none)

This is the same thing as `on` in reverse: the step disappears when the steps in the list are reached. The `here` keyword can't be used.

Note that a step with an `on` and/or `off` list doesn't follow the steps in those lists, i.e. it doesn't appear/disappear when they do, but simply when they are reached in the linear sequence of steps in the source (which coincide more often than not).

In the following example, the first step will appear by itself. Then the second step appears, along with the fourth, and the first one disappears. Finally, when the third step appears the first reappears. Since the fourth step has no autonomy, after the third step a click will lead to the next slide (provided it's the end of the slide, of course).

This means there's no transitivity in the `on` and `o` attributes: if step A is tied to step B and step B to step C, this doesn't mean A is tied to C (it can, too, of course).

```
\step[on=here B,off=A] First.
```

```
\step[A] Second.
```

```
\step[B] Third.
```

```
\step[on=A] Fourth.
```

`handout <true|false>` (Default: true)

Steps whose value is `true` for this attribute are visible on the page when the document is processed as a handout (i.e. `handout` is the value of the `presentation` attribute for the job

See [Presentation and handout](#)

parameter), no matter how they appear during a presentation. Thus the document can be printed without manipulating layers beforehand.

`visible` **<true|false>** (Default: false)

When this attribute is **true**, then the step is visible on the page when it is opened. For instance, in a slide with:

```
\step One.  
\step[visible=true] Two.  
\step[handout=true] Three.
```

This kind of step of course doesn't take a click during the presentation. In the example on the left, you need only two clicks to display Three.

the following happens: if we're in normal presentation mode, then the second step will be visible on the slide when it is displayed, whereas the other two will appear when clicking. On the other hand, in a handout, only the third step will be visible. With:

```
\setParameter step:  
  visible handout = true
```

all steps will be visible by default, in both modes.

Content

`left` **<dimension>** (Default: 0pt)

The distance between the textblock's left border (defined by the slide's `left` attribute and the left) and the border of the step's text.

You can create list items and sub-items by simply creating steps with positive left and/or right.

`right` **<dimension>** (Default: 0pt)

Same as `left`, on the right.

`vskip` **<skip>** (Default: 0pt)

The value of the vertical glue inserted each time a `\step` occurs in vertical mode.

As with slides, the step's font, if any (see below), is called before those dimensions are used, so the em and ex dimensions have the proper values. Note that there's already the `\parskip` glue between vertical steps.

`everyvstep` **<code>** (Default: nothing)

Material inserted when a `\step` occurs in vertical mode. The insertion takes place after `vskip` (and doesn't switch to horizontal mode by itself).

`hskip` **<skip>** (Default: 0pt)

The value of the horizontal glue inserted each time a `\step` occurs in horizontal mode.

`everyhstep` **<code>** (Default: nothing)

Material inserted when a `\step` occurs in horizontal mode, after `hskip`

`group` `<true|false>` (Default: false)

If this is set to `true`, the content of the step (including the value of every(v/h) step) happens inside a group.

Style

`font` `` (Default: slide's font)

The font used to typeset the step. This is actually just a placeholder for font commands, e.g. `\it` or `\bfseries`. If this attribute is used, it should be specified for all step types, or the slide or job should have its own `font` attribute, or the step should have `group` set to `true`. Otherwise, font will bleed to the rest of the slide.

`color` `<Named color or color expression>` (Default: slide's foreground)

The color of the step, overriding the slide's foreground color.

See [Colors](#)

`transition` `<named transition>` (Default: none)

The animation used to display the step, as with `\slide`. Note that the `push`, `cover` and `uncover` transitions make more sense with slides, since they affect the entire slide and not the step only.

See [Transitions](#)

Attributes for the job

This section describes attributes pertaining to the entire document. They are set with the `\setparameter` command with `job` as the parameter's name.

`author` `<string>` (Default: nothing)

The author's name. It is stored in a command called `\Author`, and used in the document's properties, unless `pdfauthor` is defined.

`pdfauthor` `<string>` (Default: nothing)

This is used in the document's properties instead of `author`.

`title` `<string>` (Default: \jobname)

The document's title. It is stored in a command called `\Title`, and used in the document's properties, unless `pdftitle` is defined.

`pdftitle` `<string>` (Default: nothing)

This is used in the document's properties instead of `title`.

date **<string>** (*Default:* <month>/<day>/<year>)
The document's date, recorded in the macro \Date.

background **<named color or color expression>** (*Default:* reader's default)
This sets the color of the reader's background (if the reader allows such a thing to be done). You can set it to the color of the slide's background, so that slides appear to have the screen's dimensions. *See Colors*

font **** (*Default:* nothing)
The default font for the job. This is actually just a placeholder for font commands, e.g. \it or \bfseries. It is wise to give a value to this attribute, so that fonts can be safely used in slides and steps.

mode **<presentation|handout>** (*Default:* presentation)
Selects how the document should be displayed. In **handout**, only those steps whose value for the **handout** attribute is **true** are displayed. The other ones nonetheless take space on the slide and can be turned on in the reader's layers panel. The **handoutonly**, **presentationonly**, and **presentationorhandout** commands are more radical yet, since they gobble the discarded material, which isn't typeset. *See Presentation and handout*

fullscreen **<true|false>** (*Default:* false)
If this is turned to **true**, the document is opened in full screen, which will probably make the reader send a message for confirmation.

outoffullscreen **<true|false>** (*Default:* false)
If this is turned to **true**, the document is displayed in full screen before navigating to a destination (when a bookmark is clicked, for instance), so that the destination is properly reached.

normal **<none|outlines|thumbs|layers>** (*Default:* outlines)
Selects what must be displayed in the reader's navigation panel when not in full screen: **outlines** is, well, outlines (or bookmarks), **thumbs** is thumbnails, and **layers** shows the layers panel, where the steps can be turned on and off; **none** folds the navigation panel.

menutext **<text>** (*Default:* ***)
The text to be displayed at the top of a submenu in the navigation pop-up menu, to represent the current bookmark. *See Creating bookmarks*

Areas

Material normally declared is typeset following TeX's usual rules, building paragraphs in the slide's textblock. However, in *Lecturer* you can put material anywhere, thanks to areas. Areas can also be used simply as decorations, since they're basically colored squares.

Using a grid

`\showgrid` [`<left>`,`<top>`][`<increment>`][`<named color or color expression>`][`<line width>`]

This displays a grid on top of the slides (following the command) to ease the creation of a display. Lines are drawn at a distance of `<increment>` from each other. The grid starts from the upper left corner by default, but the optional `<left>` and `<top>` set horizontal and vertical shifts respectively (`<top>` being a vertical distance à la TeX, i.e. going downward). If no optional color is given, the grid is painted grey, and the default line width is .2pt. There can be as many such declarations as wanted, each grid being painted on top of the previous one. For instance:

Grids can also be used as decorations with a little imagination.

```
\showgrid[2cm 2cm]{1mm}
\showgrid{1cm}[red][.4pt]
```

displays a grid in millimeters in grey, starting at 2cm from both the upper and left border, and a grid in centimeters on top of it in red, with a line width twice the previous one, starting from the upper left corner. Note that there should be no space before the last two optional arguments (if it was allowed there'd be risks of unwanted space gobbling).

`\hidegrids` Hides all grids on following slides, as if `\showgrid` had never been called.

Adding material to areas

`\position` `<area>`[`<left>`,`<top>`]`<material>`

This puts `<material>` inside `<area>`. However, things differ greatly whether the optional argument is present or not. In case it isn't, `<material>` is put below previous material in `<area>`, as if continuing a page. On the other hand, if the optional argument is present, then `<material>` is typeset so that its left border is at a distance of `<left>` from the left border of the `<area>`'s textblock and its baseline at a distance of (`<top>` + `<area>`'s `Stopskip`) from its upper border. Thus, if both `<left>` and `<top>` are `Opt`, then `<material>` is typeset normally. Material already in the area has no effect on this process, and `<material>` will have no effect either on incoming material, which is why `\position{area}{material}` and `\position{area}[Opt, Opt]{material}` aren't the same thing at all.

If `<left>` or `<top>` are `Opt`, they can be omitted. Not the brackets and comma, though.

The calculation includes `<topskip>` so the connection between freely positioned material and normally positioned material is preserved. When using it, this formula proves far more intuitive than it seems.

The `\position` obeys the flow of the presentation, and appears only when the `\step` where it belongs appears. In this example:

```

\step \position{myarea}{One}
\step \position{myarea}{Two}
\step \position{myarea}{Three}

```

'One', 'Two' and 'Three' will appear one after the other, and one above the other. On the other hand with:

```

\step[off=B] \position{myarea}[12pt, 12pt]{One}
\step[B, off=C] \position{myarea}[12pt, 12pt]{Two}
\step[C] \position{myarea}[12pt, 12pt]{Three}

```

'One', 'Two' and 'Three' will admittedly appear one after the other, but also disappear when one appears and take the place of the previous one, since they share the same position. All areas are emptied at each new slide. Remember that to be painted on a given slide, an area must be in the list of areas that is the value of the `areas` attribute for this slide, or shouldn't appear in the slide's `areas*` attribute. By default, slides accept all areas.

`\setarea <list of areas>`

To create an area, or to modify an existing one, this command is used with `<list of areas>` being area names separated by space; thus attributes similar for several areas can be declared at once. The command then takes attributes and values like `\setslide` above. For instance the following code creates or modifies two areas, `badidea` and `ugly`:

```

\setarea{badidea ugly}
  background = red
  foreground = green
  ...
\par

```

Dimensions

`width <dimension>` (Default: slide's width)

The area's width, i.e. the width of the painted zone.

`hshift <dimension>` (Default: opt)

The distance between the area's left border and the slide's left border.

`hshift* <dimension>` (Default: unspecified)

The distance between the area's right border and the slide's right border. The `hshift*` attribute is always obeyed if present. If neither `width` nor `hshift` are specified, then `hshift*` is set to `hshift*` and `width` is computed accordingly, which is convenient to center an area on a slide. If one of `width` or `hshift` is present but not the other, then the latter is computed with the given value and `hshift*`. If both are present, `width` is ignored and the previous rule applies.

As far as dimensions are concerned, areas are very similar to slides. See the Visual Doc for an illustration of this claim.

- `height` *<dimensi on>* (Default: slide's height)
The area's height. This defines the vertical dimension of the painted zone.
- `vshift` *<dimensi on>* (Default: opt)
The distance between the area's top and the slide's top.
- `vshift*` *<dimensi on>* (Default: unspecified)
The distance between the area's bottom and the slide's bottom. It works like `hshift*` in the vertical dimension, and is consequently always obeyed.
- `hsize` *<dimensi on>* (Default: area's width)
The line's length in the area.
- `left` *<dimensi on>* (Default: opt)
The area's left margin, i.e. the distance between its left border and the left border of its textblock.
- `right` *<dimensi on>* (Default: unspecified)
The area's right margin, i.e. the distance between its right border and the right border of its textblock. Like the same attribute for slides, it is always obeyed, i.e. if specified `hsize` and/or `left` are set accordingly. For the rule governing those values are computed, see the discussion on the `right` attribute for slides, or on the `hshift*` attribute for areas just above, which works similarly.
- `hpos` *<ff|fr|rf|rr>* (Default: fr)
This controls the area's horizontal justification. The values mean: `ff` is flushleft/flushright (i.e. text justified on both sides), `fr` is flushleft/raggedright (commonly called raggedright), `rf` is raggedleft/flushright (called raggedleft), and `rr` is raggedleft/raggedright (i.e. text centered).
- `vsize` *<dimensi on>* (Default: area's height)
The height of the area's textblock, not for page breaks, obviously, but for the `vpos` attribute below.
- `top` *<dimensi on>* (Default: opt)
The area's top margin, i.e. the distance between its upper border and the top of its textblock.
- `bottom` *<dimensi on>* (Default: unspecified)
The area bottom margin, i.e. the distance between its textblock's bottom and its bottom border. It works like `right` in the vertical dimension.
- `vpos` *<top|center|bottom>* (Default: top)
The vertical justification of the area's content in the textblock (defined as `vsize`).

Freely positioned material is unfolded (including its vertical displacement) at the top of normally positioned material. Hence its vertical justification might seem somewhat unpredictable unless `<vpos>` is set to `top`, though it isn't by TeX's standard. As a rule of thumb, take the top of the justified textblock as the vertical reference, and consider that freely positioned material is insensitive to vertical justification.

`baselinekip` *<dimension or glue>* (Default: `current \baselinekip`)
The baseline distance for the area.

`topskip` *<dimension or glue>* (Default: `current \topskip`)
The distance between the baseline of the area's first box and the area's top. Unlike T_EX's `\topskip`, this distance is always respected, no matter the height of the box.

`parskip` *<dimension or glue>* (Default: `opt`)
The distance between paragraphs in the areas, including between two normal `\position`.

`parindent` *<dimension>* (Default: `opt`)
The width of the paragraph indent for the area.

Content

`visible` *<true|false|step>* (Default: `true`)
If this attribute is set to `true`, then the area is visible on the slide even if it contains no material. With `false`, it is painted only on the slides where it is filled. In the latter case, using `step` means the area will follow the first step where the first `\position` command is issued. Empty material freely positioned may be useful to control the area, for instance:

```
\step[on=start, off=stop] \position{mayarea}[Opt, Opt]{}
```

This will make the area appear with `start`, probably its first real content, and disappear with `stop`, which probably also controls the disappearance of the last material in the area.

Two remarks: the first `\position` means the first occurrence of the command in the source, not necessarily the first to be shown on screen; thus, to work properly, the previous example should be put at the very beginning of a slide. Second, in this example, it is important that the material be freely positioned, i.e. with `[Opt, Opt]`, so as to make the subsequent `\position` think (?) the area is still empty.

`everyposition` *<code>* (Default: `nothing`)
Code to be added before `<material>` each time `\position{<area>}{<material>}` is called. This material is added in vertical mode.

All material is actually added in vertical mode with the `\position` command.

`everyfreeposition` *<code>* (Default: `nothing`)
Same as before, except it is used when `\position` is called with the optional argument for free position.

Style

`background` *<named color or shade or color expression>* (Default: none)
The color of the area's background. If none is given, the area is transparent.

See *Colors, shades and images*

`foreground` *<named color or color expression>* (Default: black)
The area's foreground color.

`image` *<image>* (Default: none)
An image inserted in the area's background, above the background color. The image is inserted at the area's upper left corner, and has the width and height it was declared with. Since this parameter is inherited, to cancel it give as `<image>` any name that is not a declared image; no error message will ensue.

`font` ** (Default: nothing)
The font used to typeset material in the area. This is actually just a placeholder for font commands, e.g. `\it` or `\bfseries`.

`frame` *<attribute-value pairs>* (Default: none)
This sets the area's frame, if you want any. The value itself is a setting of attributes, each one being accessible in the main declaration as `frame_<attribute>`. I.e.

```
frame = "width = 1pt, color = blue"
```

and

```
frame_width = 1pt  
frame_color = blue
```

are equivalent.

`frame_width` *<dimension>* (Default: Opt)
The frame's width. If positive, the frame is painted around the area. If negative, it is painted inside the area, with the the `<dimension>`'s absolute value as its width. An area with no frame simply has a `frame_width` of Opt.

`frame_color` *<named color or shade or color expression>* (Default: area's background)
The frame's color.

See *Colors, shades and images*

`frame_corner` *<miter|round|bevel>* (Default: mitre)
This is a **miter** corner: , this is a **round** one: , and this is a **bevel** one: .

`frame_dash` *<numbers>* (Default: none)
The frame's dash pattern, meaning the frame is on (visible) for n_1 points, then off (invisible) for n_2 points, then on for n_3 points, etc., where the n 's are the **numbers**. This is cyclic, so that

it starts again when **numbers** are exhausted. For instance a value of 3 5 2 makes the frame visible for 3pt, then invisible for 5pt, then visible for 2pt, then invisible for 3pt, then...

Navigation

Lecturer is made for fullscreen mode and navigation is no exception. Thus, bookmarks (and anchors in the next subsection) work properly in this mode only; when not in fullscreen, using a link leads to the page where the target appears, but not to the step in which it is embedded. Turning `out of fullscreen` to `true` in the `job` parameters makes the reader go into fullscreen when clicking a link and thus navigation hits its target.

Creating bookmarks

A presentation can be structured thanks to bookmarks (also called outlines), and these are accessible in the reader's outlines panel, or in the presentation via a pop-up menu. To create a bookmark, one uses the following command.

```
\createbookmark [<options>] <level>[<anchor>] <text>
```

This creates a bookmark with `<text>` as the text displayed in the outline panel and the pop-up menu. Bookmark hierarchy is managed with `<level>`, according to the following principle: bookmarks with larger `<level>` are children to bookmarks with smaller one. Thus, `<level>` must be a number, but it need not be an integer nor a positive number. For instance:

```
\createbookmark{0}{Part}
\createbookmark{.5}{Section}
\slide[Slide]
...
\createbookmark{1.5}{Step1}
...
\createbookmark{1.5}{Step2}
...
\endslide
```

This means you don't need to redefine all your sections if you want to add intermediate ones just give the latter any value in the interval where you want them to occur in the hierarchy.

will create bookmarks with the following hierarchy:

```
> Part
  > Section
    > Slide
      > Step1
      > Step2
```

because by default slides are bookmarked and have bookmark level 1. If sections of widest scope are eventually needed in this example, one can create them using bookmarks with a negative `<level>`.

The `<options>` are any combination of the following keywords, separated by commas: `bold`, `italic` and `bolditalic`, which specify how the bookmark is to be typeset in the outlines panel, and `open` and `closed`, which specify whether the bookmark displays its children by default or not. Finally, `nosubmenutext` applies to the pop-up menu displayed with `\showbookmarks`, as explained below.

Any other material in `<options>` is supposed to be a triplet of numbers ranging between 0 and 1, to denote the bookmark's color in an RGB model. For instance:

```
\createbookmark[italic, open, 1 0 0]{0}{Bookmark}
```

creates a red bookmark in italic that displays its children when the document is opened in the reader.

The `<anchor>` is a reference so that one can go to the bookmarked place with `\goto`, and is equivalent to using `\anchor` (next subsection).

Note that `<options>` and `<level>` are specified for a slide (if it is bookmarked) the `bookmarkstyle` and `bookmarklevel` attributes.

See *PDF and navigation for slides*

```
\showbookmarks [] <text>
```

This creates an hbox containing `<text>` (which can be anything, like real text, or a symbol, etc.); when dicked, a pop-up menu appears, which contains the bookmark hierarchy of the document. The optional style is either `flash` or `push`, which sets the little animation used when the link is dicked: it reverses its colors or seems to be pushed in the background. The keyword `none` can also be used to denote no animation.

In the pop-up menu, the children of a bookmark are displayed as a sub-menu. Hence this bookmark isn't dicked anymore, because its only function is to show the submenu. That's why the first item of this sub-menu is dicked and refers to that parent bookmark. The text used by this item is the value of the `menutext` attribute in the `job` parameter, unless the bookmark has the `nosubmenutext` option.

Navigation commands

```
\anchor <name>
```

This creates an anchor called `<name>`, which is a destination for the `\goto` command.

In what follows, all commands create a dicked hbox containing `<text>`, with animation specified by `<optional style>`, as in `\showbookmarks` above. That's why I will only describe the action performed by the commands.

```
\goto [] <name><text>
```

Go to the destination anchored by `<name>` (which can be the value of the `anchor` argument in a bookmark).

`\gotoA` [`<optional style>`] `<name><text>`
This creates the first item of a bidirectional link called `<name>`. Clicking `<text>` goes where `\gotoB` with `<name>` has been issued.

`\gotoB` [`<optional style>`] `<name><text>`
This creates the second item of a bidirectional link called `<name>`. Clicking `<text>` goes where `\gotoA` with `<name>` has been issued. There's no need for `\gotoA` to appear before `\gotoB`. Using these macros is just simpler than a pair of `\anchor`'s with a pair of `\goto`'s.

`\firstslide` [`<optional style>`] `<text>`
Go to the presentation's first slide.

`\lastslide` [`<optional style>`] `<text>`
Go to the presentation's last slide.

`\prevslide` [`<optional style>`] `<text>`
Go to the presentation's previous slide.

`\nextslide` [`<optional style>`] `<text>`
Go to the presentation's next slide.

`\prevstep` [`<optional style>`] `<text>`
Go backward once. Equivalent to a right click.

`\nextstep` [`<optional style>`] `<text>`
Go forward once. Equivalent to a left click.

`\showorhide` [`<optional style>`] `<actions><text>`
This shows or hides the steps referred to in `<actions>`, which is made of `'<action>=<step list>'` pairs separated by commas, where `<action>` is `on`, `off` or `toggle`, and `<step list>` is a list of step names separated by space. The action is to show, hide, or reverse the visibility of a set of steps. For instance:

```
\showorhide{on=A B, off=C, toggle=D E F}{Button}
```

will, when clicked, make the steps named A and B visible, hide step C, and reverse the visibility of D, E and F.

Colors, shades, and images

Some attributes above take values denoted by the phrases *named color* or *shade* or *color expression*, or *named color* or *color expression*. These are explained in this section.

Colors

A color can either be a *named color* or a *color expression*. The latter is the simpler. It is a `<color model>` followed by as many `<number s>` as required, ranging from 0 to 1. A color model is one of the keywords `grey` (or `gray`), which takes one `<number>` (0 is black, 1 is white), `rgb`, which takes three `<number s>` (red, green and blue), and is additive, and `cmyk`, which takes four numbers (cyan, magenta, yellow, black), and is subtractive. Hence:

```
\setparameter slide:
  background = "rgb 1 0 0"
  foreground = "cmyk 1 0 1 0"
  ...
\par
```

sets the slide's background color to red, and its foreground color to green. Which is not recommended.

Color expressions aren't very handy, since you have to type them as many times as you need them. Which is why there exists the following command to declare a *named color*:

```
\newcolor <name><color model>[<opacity>] <values>
```

After this declaration you can use `<name>` to denote a color, which itself is somehow as a color expression, i.e. it has a `<color model>` and as many `<values>` as required. The optional `<opacity>` is a number between 0 and 1, 0 meaning fully transparent and 1 fully opaque. If not present, it is set to 1. The following named colors are already defined (with full opacity): `black`, `white`, `red`, `green`, `blue`, `cyan`, `magenta`, `yellow`. I suppose you can see where they come from. So our example above could be more easily rewritten as:

Unlike `\newslide` or `\newarea`, `\newcolor` and `\newshade` below take only one `<name>` (instead of a list).

```
\setparameter slide:
  background = red
  foreground = green
  ...
\par
```

Finally, colors can be used simply with:

```
\usecolor <Named color or color expression><text>
```

This typesets `<text>` with the specified color, overriding whatever is used as the current color wherever this command is issued (i.e. step, slide, area).

Beware, the operation of the `\usecolor` command isn't executed inside a group, as may be the case with colors in other packages.

Shades

A *shade* is a transition from one color to another. It can be used for backgrounds and frames only. It takes many parameters, so it is declared with the \Yax syntax.

$\backslash\text{newshade}$ { <name>}

The <name> is what is used to refer to this shade where it can be used.

model <grey|gray|rgb|cmyk > (Default: grey)

The shade's model for its colors. You can't create a shade with colors from different models.

from <values > (Default: black)

The shade's starting color. There must be as many numbers as required by the model parameter. If nothing is specified, black is supplied.

to <values > (Default: white)

The shade's ending color. There must be as many numbers as required by the model parameter. If nothing is specified, white is supplied. Neither from nor to can take named colors as arguments.

angle <angle > (Default: 90)

The shade varies along an axis. This axis starts in the upper left corner of the area to be painted and makes an angle of <angle> degrees with the area's upper border. The <angle> must range between 0 and 90. 0 means that the shade progresses horizontally and 90 that it moves vertically. If <angle> is between 0 and -90, its absolute value is taken as its value but the shade starts from the upper *right* corner.

The value of <angle> actually refers to a shade painted in a square; if the area to be painted isn't a square, this imaginary square is scaled and the angle of the shade will follow the scaling. For instance, an angle of 45 makes the shade progress from the top left corner to the bottom right corner; if the area to be painted is a rectangle, the shade will still progress between these two corners, and the angle will be skewed.

Shades' attributes may be more easily understood with the interactive examples in the Visual Doc.

speed <number > (Default: 1)

The speed of the shade's progression: 1 means the shade takes its axis' full length to progress; a value between 0 and 1 makes it take more than this length (and thus it won't be fully painted in the area) whereas a value larger than 1 makes it take less than this length (for instance with $\text{speed} = 2$ it'll take half the axis' length).

width <dimension > (Default: none)

A shade normally adapts to the area it paints. However, if width is given, the shade will take the specified dimension; if width is larger than the area to be painted, the shade won't be seen in its entirety; if it is smaller, part of the area will be painted with the shade's to color.

height <dimension > (Default: none)

Same as width in the vertical dimension.

`fixed <true|false>` (Default: `false`)

By default, a shade is painted from the area's upper left corner (or right corner if `angle` is negative). If `fixed` is set to `true`, then it is painted from the slide's upper left (or right) corner, although only the area displays it. Note that a `fixed` shade whose width and height are unspecified doesn't make much sense.

To sum up: picture two areas painted with the same shade. If the shade has no width nor height and isn't `fixed`, they will both display the entire shade, and if they don't have the same dimensions the shade will have different shapes in them. If the shade has a width and a height, both areas will display exactly the same shade, although what is revealed of it depends on the areas' dimensions. Finally, if the shade is `fixed`, the areas will look like open windows on the same underlying shade.

Images

`\newimage <name>[<width>,<height>]<file>`

This loads image file `<file>`, setting its dimensions to `<width>` and `<height>`; if none are present, the image has its natural width; if only one dimension is given, the other one is scaled accordingly. If both are given, deformations might ensue. The image can be used in the background of a slide or area by setting `<name>` as the value for the `image` attribute, and anywhere else with the following command.

If only one dimension is used, the comma should nonetheless remain, with nothing left or right, in order to indicate which dimension is referred to. Specifying a `<height>` only is done for instance with `[,5cm]`. Note that background images in slides and areas (as values to the `image` attribute) aren't scaled to their container's dimensions

`\useimage <name>`

This returns an `hbox` containing the image `<name>`. The height and width of the box are the height and width of the image as declared with the previous command.

Drawing symbols

Lecturer provides very basic but still useful drawing facilities aimed at creating symbols to whatever end, e.g. a button for navigation or a bullet before each step.

`\newsymbol <command>[<settings>]<drawing>`

This creates a symbol that is called with `<command>` and designed with `<drawing>`; the symbol is actually an `hbox`, and its reference point (i.e. the intersection of its left border with the baseline) is the origin of the coordinate system for `<drawing>`; in turn, this `hbox`'s width is the largest x-coordinate in `<drawing>`, its height the largest y-coordinate, and its depth the smallest negative y-coordinate. Any part of `<drawing>` on the left of the y-axis, i.e. with a negative x-coordinate, is ignored in the box's width and will overlap material on the box's left.

In `<drawing>` goes a set of simple statements separated by commas, a statement being an operator sometimes followed by arguments (separated by space), which either define

paths or paint them. All coordinates are expressed in units, a unit being 1pt by default, which can be changed in `<settings>`. At the beginning the current point is (0, 0). Here are the operators (in the first two, '[' +]' means an optional + character, without the brackets):

`move [+]` *x* *y*

If + is not present, this moves to (*x*, *y*), which becomes the current point; if there is a +, and the current point is (*x'*, *y'*), then it moves to (*x'* + *x*, *y'* + *y*), which becomes the current point. If there was a path being drawn, it is ended and stroked beforehand.

`line [+]` *x* *y*

If + is not present, this creates a line from the current point to (*x*, *y*), which becomes the current point; if there is a +, and the current point is (*x'*, *y'*), then it creates a line between the current point and (*x'* + *x*, *y'* + *y*), which becomes the current point. Note that the `line` operator is actually optional and any statement of the form '[' +]' *x* *y*' is understood as `line [+]` *x* *y*'.

`circle <direction>` `<radius>`

Despite its name, this actually creates a quarter-circle from the current point to a point depending on `<direction>` and `<radius>`, with the latter being a number and the former one of the keywords `ul`, `ur`, `lu`, `ru`, `dl`, `dr`, `ld` or `rd`, where each letter denotes *left*, *right*, *up* or *down*, these in turn being the quarter-circle's tangents' directions at the starting (for the first letter) and ending (for the second letter) points. For instance, `circle ur 7`, `circle rd 7` creates a quarter-circle of radius 7 going up and then right, followed by a quarter-circle of the same radius going right and then down, which figure mathematicians tend to call a semi-circle, i.e. . The current point after a (quarter-)circle is its ending point, i.e. `radius` units away left or right and below or above from the starting point.

`close`

`T` closes the path, i.e. appends a line from the current point to the path's starting point.

`stroke`

`T` strokes the current path. `T` is operator is appended at the end of `<drawing>` if there remains a path that hasn't been painted; thus it is useless as the very last command.

`fill`

`T` fills the current path, i.e. the path is closed and the area it delimits is colored.

Differences between fill and paint are visible when the line width, set with pen, is large.

`paint`

`T` is similar to `fill`, except the path is also stroked.

`pen <width>`

`T` sets the width of the stroking pen to `width`.

color <named color or color expression>

This changes the current color to the specified value. If several such commands are issued, the painting operators above use the color most recently issued when they are called. If no color is specified, the symbol takes the foreground color of the place where it appears.

See *Colors*

The optional <settings> to \newsymbol are comma-separated 'attribute=value' pairs; there can also be a single <dimension> among them expressing the value of a unit in the coordinate system, which is 1pt by default. The attributes are left, right, top, bottom and padding, and the values are <dimensions>. These attributes set the amount of padding on the left, right, top and bottom borders respectively, with padding setting them all at once. Padding is sometimes necessary because the drawing's box's (and clipping path's) dimensions are defined according to the abstract paths the drawing is made of, not according to the paths as they are painted, and a painted path has the width of the pen used to paint it. For instance, the simple drawing \line 0 10' has no width, even though it might be painted with a 10-unit wide pen. If there's no padding, the symbol won't show anything. By default, padding is set to 1pt on all sides.

As an example, here's how the alien has been drawn in the demonstration file called LecturerDemo-KitschScienceFiction.pdf:

```
                % Default unit.
\newsymbol \alien[1.7em] {%
  pen .05,
  color cmyk .2 1 .4 0,
  % The alien's body
  1 0, circle ul .5, circle ld .5, fill,
  % The left tentacle
  circle dl .2, circle lu .2,
  % The right tentacle
  move 1 0, circle dr .2, circle ru .2,
  % The legs
  move .16 0, + 0 -.3, % stroke is implicit in the next move
  move .49 0, + 0 -.3,
  move .82 0, + 0 -.3, stroke,
  % The eyes
  color cmyk 0 0 1 0,
  move .2 .25, + .2 0,
  move + .2 0, + .2 0 % stroke is implicitly added at the end
}
```

\symbol width <command>

\symbol height <command>

\symbol depth <command>

When a symbol is defined, its dimensions can be queried with those macros, where <command> is the symbol's command.

Transitions

Transitions are basic animations played when a step (dis) appears on the slide or when one advances to the next slide. The `transition` attribute for slides and steps can take a `<named transition>` as its value, and one declares a named transition with the following command:

```
\newtransition <name>
```

This command takes several attributes, which are:

`type` `<split|blinds|box|wipe|dissolve|glitter|fly|push|cover|uncover|fade>`
(Default: none)

(These values can also have an uppercase first letter.) This is the type of the animation being played. The predefined transitions already available in *Lecturer* are those animations with the following parameters set to their default values.

The Visual Doc has a slide illustrating those transitions

`motion` `<inward|outward>` (Default: inward)

(Only for transitions of type `split`, `box`, and `fly`.) The direction of motion, either from or to the center of the page.

`direction` `<lr|bt|rl|tbdx>` (Default: lr)

(Only for moving transitions.) The direction of movement: left to right, bottom to top, right to left, top to bottom, and diagonal (top-left to bottom-right).

`dimension` `<horizontal|vertical>` (Default: horizontal)

(Only for `split` and `blinds`.) Whether the splitting is horizontal or vertical.

`scale` `<number>` (Default: 1)

(Only for `fly`.) The scale at which elements affected by the transition are drawn at the beginning of the transition (if its motion is `inward`) or at the end (if it is `outward`).

`duration` `<number>` (Default: 1)

The length of the transition, in seconds.

Inserting PDF code

Lecturer relies heavily on PDF code and unfortunately pdfTeX's management of some PDF objects is far from optimal. Consequently, if you want to insert some PDF constructs things might go wrong. Hence the following macros, where arguments are always expanded immediately.

`\addtopageobject <code>`
This adds `<code>` to the Page object for the current page.

`\addtoeachpageobject <code>`
This adds `<code>` to the Page object for all pages. This command and the previous one fill the `\pdfpageattr` token list, which after each shipout is emptied of material added with `\addtopageobject`.

`\addtopageresources <code>`
This adds `<code>` to the Resources dictionary for all pages. This uses the `\pdfpageresources` token list. For the Properties, Shading and ExtGState resources, the following command should be used.

`\addproperties <name><object number>`
Maps `<name>` to `<object number>` in the Properties dictionary of the current page's Resources. For instance,

```
\addproperties{foo}{3}
```

produces the following in the page's Resources:

```
...  
/Properties << ... /foo 3 O R ... >>  
...
```

`\addshading <name><object number>`
Same as `\addproperties`, but for the Shading dictionary. And this is added to all pages' Resources, not just the current one.

`\addgstate <name><object number>`
Same as `\addshading`, but for the ExtGState dictionary.

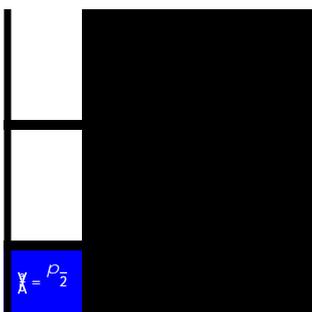
`\addOCG <object number>`
Adds the Optional Content Group `<object number>` to the document's catalog, with base state off. It is also turned on when one arrives on the page where it's been added.

`\addvisibleOCG <object number>`
Same as `\addOCG`, except base state is on.

Demonstrations

The present documentation might appear as a dull list of attributes. That's why it is distributed with a Visual Documentation (`LecturerDemo-Vi s u a l Doc. pdf`), which isn't really a demonstration, in the sense that it is not supposed to be an imaginary presentation but a visual display of many features of *Lecturer*. Thus commands and attributes can make sense at once. One should nevertheless keep in mind that this visual documentation doesn't show everything in *Lecturer*, and that the reference documentation remains the present one. The source isn't terribly user-friendly (and requires LuaTeX and non-free fonts to compile), but still interesting constructions are used and can be copied, in particular with `\showorhide`.

The visual documentation explains how to use *Lecturer*; the following demonstrations show what it can do. Barring `LecturerDemo-SquaresOfAs`, they have been typeset with the default Computer Modern fonts, so the sources can be modified and compiled again to see how things work. To do so, one should use the plain TeX format with either pdfTeX or LuaTeX.



`LecturerDemo-Mondrian.pdf` is a simple presentation based on the work of one famous Dutch painter. It illustrates what you can do with areas, setting the main text in one square (although the text is not `\positioned` but typed in as the slide's content, see below), the maths (synced with the main text) in another one, and footnotes, which are turned on and off with the `\showorhide` command, in a third square. The other squares are just decorations. The alignment of the squares is made easy by grouped attribute declarations with the `\setarea` command. For instance, the three squares on

the left are declared as:

```
\setarea{area1 area2 matharea}
width = 3cm
```

and the first two, which have the same height and color, are specified further again with:

```
\setarea{area1 area2}
height = 4.3cm
background = white
```

after which `area1` doesn't need anything; another declaration sets `area2's` `vshift`, and still another one is used for `matharea` to display its content.

The slide's background is painted black so lines appear between the areas (which is also why the `white` color must be specified as some areas' `background`, otherwise they'd be transparent). The main text is set in the slide's `textblock` (i.e. not `\positioned`), and a red area is painted below, which gives the impression that the text is set in an area, whereas it is simply asymmetrically shifted.

Lecturer Demo- SquaresOf As. pdf was inspired by some ConTeXt presentations, and uses only areas to typeset its material. Barring the last slide, there's nothing in the main `textblock`. On the first slide, the `visible` attribute of the areas is set to `step`, so areas appear with their content. The background of the reader is turned to white, so the slides seem to fill the entire screen. The entire presentation is built on a single length (`\squarewidth`) in which all dimensions are expressed, including font sizes (such cumbersome font sizes as `1.83\squarewidth` are meant to make all letters appear at the same height). Thus the presentation is totally scalable. Like in the Mondrian presentation, attributes are declared mostly for all areas at once, so the design can be easily modified. The frames of the areas have negative width, so they are painted inside the areas and they don't have to be taken into account when computing positions. Actually, their contribution to the design resides in their rounded corner only, since they have the same `background` color as the areas.

The presentation itself is a wandering among several guises of the letter *a*, with several non-free TrueType fonts (requiring LuaTeX), and thus you should change those to fonts you own and can manage if you want to compile the source. (If you think it is inhuman to make the part of the letters below the baseline disappear in the white background, just change the latter. Jovica Veljović's *Esprit* will regain its spirit indeed.)

Lecturer Demo- SimplePresentation. pdf uses a very simple display, built on a millimetric grid, which can be displayed by uncommenting the first lines of the file. Some subtleties are included in the presentation's flow, for instance, the triangles at the left of each vertical step are already painted on the slide before the steps appear. This is done by creating two steps, one for the triangle with `visible` set to `true`, and one for the step's content, a process which is automated by a simple code:

```
\def\Step{%
  \step[visible=true]\quittmode\l\ap{\stepsymbolker n. 2cm}%
  \step}
```

where `\stepsymbol` is the symbol's command. Note the necessary `\quittmode`: those steps are called in vertical mode and plain TeX's `\l\ap` macro creates a box which should be inserted in

horizontal mode to get things right. The `everyvstep` attribute could also have been used, but care should be taken to ensure that no loop occurs, by creating two kinds of step, the `everyvstep` of one executing the other.

The horizontal and vertical lines are actually areas with no width but a frame indeed, created with:

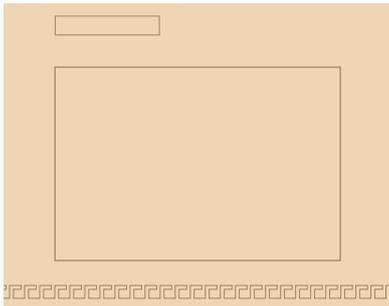
```
\string\setarea{vline hline}
  frame = "width = .15pt, color = black"
```

```
\setarea{vline}
  hshift = 3cm
  width = 0pt
```

```
\setarea{hline}
  vshift = 2cm
  height = 0pt
```

When clicked, the circle (a symbol) with the slide number in the bottom right corner displays the bookmarks in a pop-up menu. And the maths in the blackboard are of course inserted with free `\position's`. The next presentation is an elaboration on this one.

`Lecturer Demo- Beamer CambridgeUS. pdf` is an imitation (not terribly faithful) of the Beamer package's CambridgeUS theme. It uses `everyslide` to typeset recurrent material, like the date or the slide's title. It also mimicks Beamer's display of inactive steps in transparent color. This is no feature in *Lecturer*, though, and must be done by hand with overlapping elements, which means it's basically undoable for the main text. The presentation is also heavy on symbols designed with `\newsymbol` used as buttons, and uses bookmarks and anchors for navigation, including between the main text and appendices, thanks to `\got oA` and `\got oB`. The last slide displays scaled content, and finally, if the `mode` attribute is turned to `handout`, a handout is produced. (Which is true of all presentations, except in this case care has been taken so that the handout isn't just a presentation on a sheet of paper, although the differences are rather slight with the original presentation.)



Lecturer Demo- ThePoodl eLect ures. pdf (don't ask) is made of five slides each displaying a different presentation style, sometimes verging on the PowerPoint-ish. The point of this demonstration is that the code for each slide remains minimal: you don't need to write hundreds of lines of attribute specifications to set up a presentation. Areas are used for decoration and the slide's title, and no complex positioning is involved. The same steps are reused (with silly text, don't ask either) and illustrate the use of the `left` attribute for

steps as an easy way to create list items. Most of the effort, though, is spent on symbol design, from simple square to music note to sea shell. (Note that this demonstration contains such declarations as `\hshi ft hshi ft* = 2cm!` when `hshi ft*` is supposed to automatically set `hshi ft` if the latter isn't specified; but since the same areas are reused for all slides, `hshi ft` is specified after the first slide.)

Lecturer Demo- Ki t schSci enceFi cti on. pdf is a pathetic attempt at creating a video game with T_EX. Obviously there are better ways to do that. Nevertheless, the presentation illustrates many *Lecturer's* features, including the use of `\show or hide`, free positions, transparent colors, shades, and a background image. The drawing of symbols is also used to an extent it wasn't designed for in the first place. The reader can pay a visit to the layers panel (which is displayed by default when not in full screen) to see how elements are organized on the screen. Once again an area (the green square) is painted below the slide's textblock, with similar colors, to give the impression of a screen, even though as far as *Lecturer* is concerned the textblock and the area are totally unrelated, i.e. the text isn't the content of the area. The darker line that appears at the area's perimeter is produced by the area and its frame overlapping, an unwanted side effect with transparent colors, unavoidable but not so bad-looking here.