# FiXme – Collaborative annotation tool for LaTeX*

Didier Verna
mailto:didier@didierverna.net
http://www.lrde.epita.fr/~didier/

v4.4 (2017/03/05)

**Abstract**

FiXme is a collaborative annotation tool for LaTeX documents. Annotating a document here refers to inserting meta-notes, that is, notes that do not belong to the document itself, but rather to its development or reviewing process. Such notes may involve things of different importance levels, ranging from simple "fix the spelling" flags to critical "this paragraph is a lie" mentions. Annotations like this should be visible during the development or reviewing phase, but should normally disapear in the final version of the document.

FiXme is designed to ease and automate the process of managing collaborative annotations, by offering a set of predefined note levels and layouts, the possibility to register multiple authors, to reference annotations by listing and indexing *etc.* FiXme is extensible, giving you the possibility to create new layouts or even complete "themes", and also comes with support for AUC-TeX.



## Contents



*FiXme homepage: `http://www.lrde.epita.fr/~didier/software/latex.php#fixme`

# 1 Installation

## 1.1 Extraction

If you are building FiXme from the tarball you need to execute the following steps in order to extract the necessary files. FiXme also requires the DoX package (version 2.0, release date 2009/09/21 or later), to build. It is not required to use the package.

```
[pdf]latex fixme.ins
[pdf]latex fixme.dtx
[pdf]latex fixme.dtx
makeindex -s gind fixme.idx
[pdf]latex fixme.dtx
[pdf]latex fixme.dtx
```

After that, you need to install the generated documentation and style files to a location where LaTeX can find them.

### 1.2 TDS-compliant layout

For a TDS-compliant layout, the following locations are suggested:

```
[TEXMF]/tex/latex/fixme/fixme.sty
[TEXMF]/tex/latex/fixme/layouts/fxlayout*.sty
[TEXMF]/tex/latex/fixme/layouts/env/fxenvlayout*.sty
[TEXMF]/tex/latex/fixme/layouts/target/fxtargetlayout*.sty
[TEXMF]/tex/latex/fixme/themes/fxtheme*.sty
[TEXMF]/doc/latex/fixme/fixme.[pdf|dvi]
```

### 1.3 AUC-TEX support

AUC-TEX is a powerful major mode for editing TEX documents in Emacs. In particular, it provides automatic completion of command names once they are known. FiXme supports AUC-TEX by providing a style file named `fixme.el` which contains AUC-TEX definitions for the relevant commands. This file should be installed in a place where AUC-TEX can find it (usually in a subdirectory of your LATEX styles directory). Please refer to the AUC-TEX documentation for more information on this.

## 2 Features summary

If you're new to FiXme, you might be interested in a brief summary of the features it provides. Otherwise, you may only take a look at the History section (section 5 on page 25) to see what's new.

**Annotation levels** FiXme annotations may be of four different importance levels, ranging from simple not-so-important notices to critical things that must absolutely be fixed in the final version.

**Layouts and themes** FiXme gives you full and extensible control on the layout of these annotations: they can be displayed inline, as marginal paragraphs, as footnotes and also in any kind of user-defined way. All these "layouts" may be combined together. FiXme also comes with support for "themes", globally modifying existing layouts, or providing new ones.

**Annotation targets** Annotations may be "targeted" to a specific portion of text that will be highlighted, and on the contrary "floating" around, in which case they may even appear in the document's preamble.

**Listing and indexing** Annotations may be indexed and summarized in a "list of fixmes".

**Logging** Annotations are recorded in the log file, and (depending on their importance level) some of them are displayed on the terminal during compilation. A final summary is also created at the end of the compilation process.

**Modes** All these features are actually available when you're working in `draft` mode. In `final` mode, the behavior is slightly different: any remaining critical note generates an error (the compilation aborts), while non critical ones are just removed from the document's body (they're still recorded in the log file though).

**Authoring** FiXme provides support for collaborative annotating by allowing you to "register" several authors.

**Internationalization** FiXme currently supports 7 different languages and features automatic language tracking for multilingual documents.

# 3 Using FiXme

## 3.1 Initialization

### 3.1.1 Requirements

In order to work properly, FiXme requires the presence of some LaTeX packages. You don't have to load them explicitly though. As long as LaTeX can locate them, they will be used automatically. FiXme currently depends on `xspace`, `ifthen`, `verbatim` and `xkeyval` (version 2.5f, release date 2006/11/18 or later).

### 3.1.2 Loading the package

In order to load FiXme, simply say `\usepackage[`⟨*options*⟩`]{fixme}` in the preamble of your document. There is an important number of options that you can use in order to customize FiXme's default or global behavior. These options will be discussed when appropriate.

There might be times where you would like to use LaTeX commands in package options (for example, see section 3.9 on page 16). In such a case, you should know that LaTeX normally can't handle this. In order to make it work, you need to use the `xkvltxp` package first, like this:

```
\usepackage{xkvltxp}
\usepackage[myoption=\mymacro]{fixme}
```

### 3.1.3 Global setup modification

`\fxsetup` `{`⟨*options*⟩`}`

Another way of customizing FiXme's global behavior is to use the `\fxsetup` command. `\fxsetup` understands the same options as the package itself and can be used in the preamble as well as in the document's body.

### 3.1.4 Local setup modification

Finally, note that unless specified otherwise, all package options are also understood by the annotation commands or environments described in section 3.2 on page 7. The effect is then local to that particular command.

## 3.2 Inserting FiXme notes

### 3.2.1 Commands

`\fxnote` `\fxwarning` `\fxerror` `\fxfatal`

[⟨*options*⟩]{⟨*note*⟩}
FiXme provides four annotation commands corresponding to different levels of importance (notes, warnings, errors and fatal errors). `\fxfatal` is a bit different from the other ones, as will be explained in section 3.4 on page 8.

`\fixme`

> ***Warning:*** *as of version 4, the `\fixme` command is a synonym for `\fxfatal` and is considered deprecated.*

### 3.2.2 Targeted commands

`\fxnote*` `\fxwarning*` `\fxerror*` `\fxfatal*`

[⟨*options*⟩]{⟨*note*⟩}{⟨*text*⟩}
Sometimes, you might not only want to issue a FiXme note, but also highlight the relevant part of the text to which it applies. This is what I call "targeting" the annotation. As of version 4, FiXme provides starred versions of its annotation commands to do that. In star form, these commands expect an additional mandatory argument containing the text to be highlighted.

### 3.2.3 Environments

> ***Warning:*** *as of version 4.0, the environment interface has changed and is* not *backward-compatible.*

`anfxnote` `anfxwarning` `anfxerror` `anfxfatal`

[⟨*options*⟩]{⟨*summary*⟩}
FiXme annotations are normally meant to be short: consider that they are likely to go in the list of fixmes and in the index for instance. If you feel the need for writing longer comments, the environments described below might come in handy. FiXme provides four annotation environments; one for every note level. These environments take one mandatory argument (meant to be a short summary of the long note) and behave in exactly the same way as their command counterpart. The layout policy is a bit different though (see section 3.5 on page 8): the environment's contents will always appear inline, and the ⟨*summary*⟩ will obey all active annotation layouts except for the `inline` one, just as if it had been passed to one of the FiXme annotation commands described in the previous section.

`afixme`

> ***Warning:*** *as of version 4, the `afixme` environement is a synonym for `anfxfatal`, and is considered deprecated.*

### 3.2.4 Targeted environments

`anfxnote*` `anfxwarning*` `anfxerror*` `anfxfatal*`

[⟨*options*⟩]{⟨*summary*⟩}{⟨*text*⟩}
FiXme environments can also be targeted to a specific portion of text. When using the starred version, the environments expect one additional mandatory argument: the text in question that will be highlighted.

## 3.3 List of FiXme's

`\listoffixmes`

FiXme remembers where you put your annotations in a toc-like file whose extension is `lox`. The `\listoffixmes` command generates the annotations lists in a manner similar to that of the "list of figures". A standard layout is automatically selected for the `article`, `report` and `book` classes and the AMS ones. If loaded, FiXme will also use the `tocbasic` package which makes it compliant with the KOMA-Script classes and any other document using it. If another class is used, the `article` layout is selected. Also, note that if there isn't any annotation left in the document, this command doesn't generate an empty list, but rather stays silent. It also stays silent in `final` mode, regardless of the presence of remaining annotations (see section 3.4 on page 8).

## 3.4 Controlling the behavior of FiXme

`final` `draft` The behavior of FiXme is controlled by the two standard options `final` and `draft`. These options are usually given to `\documentclass` which in turn passes them to all packages. In addition, you can also use them as options to `\usepackage`, in the call to `\fxsetup`, and even to the annotation commands and environments.

In `draft` mode, annotations are recorded in the log file and appear in the document as specified by the layout settings (see section 3.5 on page 8). Additionally, warnings, errors and fatal errors are also displayed on the terminal.

In final mode, non fatal annotations (those generated by `\fxnote`, `\fxwarning`, `\fxerror` and their corresponding environments) are still logged, but they're not typeset. On the other hand, fatal ones (those generated by the `\fxfatal` command and the `anfxfatal` environment) will throw a LaTeX error and thus interrupt or abort compilation with an informative message. This will help you track down forgotten important caveats in your document.

Let me rephrase: final documents can only have FiXme notes, warnings, and (non fatal) errors left. Of course, this is not completely true: remember that these options are understood locally by all the annotation commands and environments, so even in `final` mode, you can use something like this:

```
\fxfatal[draft]{bla bla}
```

`status` By default, FiXme is in `final` mode (LaTeX itself behaves that way). If you're manipulating the document status at the level of FiXme itself (as opposed to the `\documentclass` level), then the preferred way to do this is to use the `status` option, and give it the value `final` or `draft`.

## 3.5 Controlling the layout of annotations

Annotations can appear in several forms in your document. Each of these forms can be individually selected, or they can be combined together to some extend.

### 3.5.1 Selecting a layout

#### 3.5.1.1 Individual control

For each annotation layout, there is a corresponding boolean option (for instance, the "inline" layout is controlled by the `inline` option). These options are understood by the package itself, the `\fxsetup` command and also locally by every annotation command or environment. There are some restrictions on their usage however, as discussed in the next section.

To activate a layout, use the option alone or give it a value of `true`. For instance, these two forms are equivalent:

```
\fxnote[inline]{note...}
\fxnote[inline=true]{note...}
```

For convenience, each layout option has a counterpart that deactivates the corresponding layout. The counterpart option has the same name, prefixed with `no` (for instance, `noinline`). Again, these options are understood by the package itself, the `\fxsetup` command and also locally by every annotation command or environment (with the same usage restrictions, discussed in the next section). For instance, these two forms are equivalent:

```
\fxsetup{inline=false}
\fxsetup{noinline}
```

#### 3.5.1.2 Global control

`layout` `morelayout` An even more convenient way to specify the required layout is to use the `layout` and `morelayout` options. In fact, the use of individual control is considered more or less deprecated. Both of these options take a comma-separated list of the individual options described above (this includes the `no`⟨*option*⟩ form as well).

While the `morelayout` option *adds* to the current layout configuration, the `layout` one completely overrides it. For instance, knowing that by default, only the `margin` layout is active, the following forms are all equivalent:

```
\usepackage[nomargin,inline,index]{fixme}
\usepackage[margin=false,inline=true,index=true]{fixme}
\usepackage[morelayout={nomargin,inline,index}]{fixme}
\usepackage[layout={inline,index}]{fixme}
```

Again, these two options are understood by the package itself, the `\fxsetup` command and also locally by every annotation command or environment (with the same usage restrictions, discussed in the next section).

`\fxuselayouts` `{`⟨*name,...*⟩`}`
Finally, an alternative way of selecting (or deselecting) several layouts simultaneously is to use the `\fxuselayouts` command, giving it a comma-separated list of layout options as its only, mandatory, argument.

### 3.5.2 Built-in *vs.* external layouts

Annotation layouts are provided either in the core of FiXme, or in separate files loaded dynamically on demand. Simple layouts are typically built-in, whereas those requiring additional packages are external, so that they don't consume TEX resources if not used. As a consequence, selecting an external layout might involve loading the relevant file first.

`\fxloadlayouts` `{`⟨*name,...*⟩`}`
For technical reasons, it is not possible to do such a thing outside the preamble, neither in the middle of processing `\usepackage` options. As a result, layout options are restricted and you have three possibilities for using an external layout:

| **Name** | **External** | **Description** |
|---|---|---|
| `inline` | | Display note inline |
| `margin` | | Display note in the margin |
| `footnote` | | Display note in a footnote |
| `index` | | Display note in the index |
| `marginclue` | | Display a marginal clue |
| `marginnote` | * | Display non-floating note in the margin |
| `pdfnote` | * | Display note as inline PDF comment |
| `pdfmargin` | * | Display note as marginal PDF comment |
| `pdfsignote` | * | Display signed note ala `pdfnote` |
| `pdfsigmargin` | * | Display signed note ala `pdfmargin` |
| `pdfcnote` | * | Display colored note ala `pdfnote` |
| `pdfcmargin` | * | Display colored note ala `pdfmargin` |
| `pdfcsignote` | * | Display colored note ala `pdfcsignote` |
| `pdfcsigmargin` | * | Display colored note ala `pdfsigmargin` |

Table 1: Available annotation layouts

1. Use its corresponding option in a call to `\fxsetup` in the preamble, like this: `\fxsetup{`⟨*option*⟩`}`. This will load it *and* select it immediately.

2. Use the `\fxuselayouts` command in the preamble like this: `\fxuselayouts{`⟨*name*⟩`}`. This is strictly equivalent to the previous solution.

3. If on the other hand you want to load one or several external layouts *without* using them immediately (perhaps in order to use them locally in some specific annotation), use the `\fxloadlayouts` command in the preamble like this: `\fxloadlayouts{`⟨*name*⟩`,...}`. After that, you can select any of those layouts anywhere you wish.

#### 3.5.3 Available layouts

`[no]inline` `[no]margin` `[no]footnote` `[no]index`

Table 1 lists the annotation layouts currently distributed with FiXme. By default, only the `margin` layout is active. Most of these layouts should be self-explanatory, but some precisions are given below.

##### 3.5.3.1 `marginclue`

`[no]marginclue`

If your preferred layout is `inline` or say, `footnote`,it might be somewhat difficult to localize the annotation on the page, especially its vertical position. That's where marginal clues come into play. A marginal clue does not display the annotation's contents, but only an indication that there is one at that (vertical) position. So you need to use another layout as well (again, typically `inline` or `footnote`) in order to get the actual annotation.

Obviously, the margin and margin clue layouts are mutually exclusive, so if you try to activate both, only the most recently activated one will be enabled (and you'll get a notice in the log file and on the terminal).

#### 3.5.3.2 marginnote

[no]marginnote

The `marginnote` layout is an alternate (external) way to display annotations in the margin, using the eponymous package. Contrary to LaTeX's standard marginal paragraphs, the ones issued by `marginnote` are constructed in a non-floating way. This might be an advantage in some situations but `marginnote` also comes with some disadvantages of its own. For more information, please refer to `marginnote`'s documentation, and also read the next section. Also, note that it is not currently possible to pass options to the `\marginnote` command through this layout.

For a reasonably robust marginal layout accross all annotations, including those issued in floats, consider using `marginnote` in conjunction with `innerlayout=noinline` (see section 3.5.4 on page 11).

#### 3.5.3.3 PDF comments

[no]pdfnote
[no]pdfmargin
[no]pdfsignote
[no]pdfsigmargin
[no]pdfcnote
[no]pdfcmargin
[no]pdfcsignote
[no]pdfcsigmargin

The PDF format comes with a concept of *comment*, which FiXme can use to display its own annotations. Support for PDF comments varies across PDF viewers. Acrobat Reader is usually considered a reference, and MacOS X's Preview supports them reasonably well. The `pdfnote` and `pdfmargin` layouts use the `pdfcomment` package to display annotations as PDF inline or marginal comments.

The `sig` versions additionally display the author's tag (see 3.12 on page 18) as a signature instead of as a prefix.

The versions with a `c` in their name (as in **c**olor) use one of four different colors named `fx`⟨*level*⟩ (according to the annotation's importance level). They also avoid printing the annotation's level since this information is already conveyed by the color.

### 3.5.4 Inner layout

There might be various reasons for you to change the layout locally for one particular annotation: creating a floating one is an example, see also section 3.5.5 on page 12 for some others. One frequent reason (described below) can be handled automatically by FiXme.

Remember that the default layout is to use margin paragraphs. Unfortunately, margin paragraphs are forbidden by TeX in several situations, like a figure's caption for instance. If you try that, you will get a cryptic "Not in outer par mode" error message.

innerlayout

The good news is that this situation can be detected automatically. FiXme provides an option named `innerlayout` that allows you to specify an alternative layout setting to use when TeX is in *inner* mode. In addition to that, FiXme automatically disables the `margin` and `marginclue` layouts. If you really want to use marginal paragraphs in inner mode, a good idea is then to set your inner layout to `marginnote` (see section 3.5.3.2).

Using `innerlayout` is not as trivial as it may seem: it *really* is an alternative layout configuration, and as such, you can use any combination you like of individual layout options, or you can even use the `layout` and `morelayout` options. This means that your alternative layout can either *add* to the existing one, or *override* it. Here are some examples to clarify things a little. You should try to understand them.

- By default, the FiXme inner layout is set to just `inline`. This can be simulated by the following call:

```
\usepackage[layout=margin,innerlayout={layout=inline}]{fixme}
```

- The following happens to give the same result in our particular case, while having a different semantics:

```
\usepackage[layout=margin,innerlayout=inline]{fixme}
```

- If you have set FiXme to use a safe layout globally (for instance, `inline` and `index`), and you want to use the same layout in inner mode, then you should provide an *empty* inner layout, like this:

```
\fxsetup{layout={inline,index},innerlayout=}
```

  What would happen if you didn't provide the `innerlayout` option?

One final remark on the `innerlayout` option: this option is not processed immediately when you specify it, but instead, its value is stored and used only when needed. As a result, if you plan to use an external layout in inner mode (typically, `marginnote`) , you need to load it explicitely in the preamble first. Use `\fxloadlayouts` for that.

#### 3.5.5 Other common layout problems

This section describes some other common problems that people have encountered using FiXme. Although FiXme might not be directly responsible for them, it is still good to keep them in mind.

**Annotations in captions being counted twice** You are most likely using `\listofsomething` (figure, table, or any other kind of float). Note that a caption will be used twice here: once in the float itself, and once in the list of floats. Any FiXme annotation in the caption will consequently be generated twice as well. The solution to this problem is to use the optional argument to `\caption`, for example:

```
\caption[caption text]{caption text\fxnote{yuck!}}
```

**Footnotes and margin paragraphs in floats** Using footnotes in figures (and *a fortiori* in a figure's caption) does not work in general. Although there are some workarounds out there (for instance, using `\footnotemark` and `\footnotetext` directly), there is no completely reliable solution and it is not possible to detect that situation automatically. Similarly, marginal paragraphs will cause problems in a figure (even when not in its caption) because floats can't be nested in LaTeX. Usual symptoms of these situations are: a footnote not being typeset, compilation breakage with the "Floats lost" message *etc.* If you're facing this problem, you need to change your layout locally.

**Marginal paragraphs showing up on the wrong margin** You want to look at the mparhack package.

**ACM classes compatibility** The ACM Sig classes (acm_proc_article-sp and sig-alternate) forbid the use of \marginpar, so if you use these classes, don't forget to choose another layout for FiXme, and also avoid using marginal clues.

**Annotation indexing** Remember that some characters are special in an index entry (the ! for instance). FiXme currently does nothing to escape those characters, so avoid using them in your annotations.

### 3.6 Corollary: floating annotations

At some point, people suggested that it would be nice to have global annotations, not related to any portion of the text in particular. Such annotations could be general comments about the whole document, and could even be issued in the preamble. This is what I call "floating" annotations.

I know you don't care, but originally, I started writing a new set of commands to do just that. However, with the flexibility that FiXme 4.0 provides, I quickly realized that such commands were an unnecessary addition.

Since floating anotations are not supposed to relate to any part of the text, they should not be typeset anywhere in it. This is especially true if you want to put some of them in the document's preamble. However, even a preamble annotation could be recorded and displayed in the index or in the list of fixmes. And it turns out that you can specify all that with the layout options described in section 3.5 on page 8.

target

The only remaining problem is the page number, which normally appears in the list of fixmes and in the index: if you choose to reference a floating annotation that way, the page number is likely to be completely meaningless. To compensate, a new option named target is provided. When used, the given value will replace the page number in both the index and the list of fixmes. The target can be anything you like, but should remain rather short. By default, target is set the special value thepage, which as you guessed means to use the page number.

The name "target" bears an intentional resemblance to FiXme's targeted commands and environments, because we are indeed targetting the annotation to something. The only difference is that in the case of floating annotations, the target is non-textual.

Here is an example of a floating annotation that would typically appear in the document's preamble:

```
\usepackage{hyperref}
\fxfatal[layout=index,target=hyperref]{Fill in PDF fields (title etc.)}
```

### 3.7 Controlling the layout of environments

As discussed in section 3.2 on page 7, the contents of a FiXme environment (a longer annotation) always appears inline. However, the exact way this contents is typeset (in draft mode only) is subject to a layout of its own, called the "environment layout".

### 3.7.1 Selecting a layout

`envlayout` The desired environment layout can be selected with the `envlayout` option. Contrary to the annotation layouts, only one environment layout can be active at a time. The `envlayout` option is understood by the package itself, the `\fxsetup` command and all the annotation environments (not the commands!). There are some restrictions on its usage however, as discussed in the next section.

`\fxuseenvlayout` `{`⟨*name*⟩`}`
An alternative way of selecting an environment layout is to use the `\fxuseenvlayout` command, giving it the layout's name as its only, mandatory, argument.

### 3.7.2 Built-in *vs.* external layouts

Environments layouts are provided either in the core of FiXme, or in separate files loaded dynamically on demand. Simple layouts are typically built-in, whereas those requiring additional packages are external, so that they don't consume TEX resources if not used. As a consequence, selecting an external layout with the `envlayout` option might involve loading the relevant file first.

`\fxloadenvlayouts` `{`⟨*name,...*⟩`}`
For technical reasons, it is not possible to do such a thing outside the preamble, neither in the middle of processing `\usepackage` options. As a result, the `envlayout` option is restricted and you have three possibilities for using an external layout:

1. Use the `envlayout` option in a call to `\fxsetup` in the preamble, like this: `\fxsetup{envlayout=name}`. This will load it *and* select it immediately.

2. Use the `\fxuseenvlayout` command in the preamble like this: `\fxuseenvlayout{name}`. This is strictly equivalent to the previous solution.

3. If on the other hand you want to load one or several environment layouts *without* using them immediately (perhaps in order to use them locally in some specific annotation), use the `\fxloadenvlayouts` command in the preamble like this: `\fxloadenvlayouts{`⟨*name*⟩`,...}`. After that, you can select any of those layouts anywhere you wish.

### 3.7.3 Available layouts

Table 2 lists the environment layouts currently distributed with FiXme.

`plain`
- The `plain` environment layout prints its contents as-is, only in bold font (by default) in order to distinguish it from the surrounding text.

`signature`
- The `signature` environment layout prints the author's tag (see 3.12 on page 18) as a signature instead of as a prefix. This layout is used by the `signature` theme (see section 3.13 on page 21).

`color` `fxnote` `fxwarning` `fxerror` `fxfatal`
- The `color` environment layout uses one of four colors named `fx`⟨*level*⟩ (according to the annotation's importance level) to display its contents. It also avoids printing the annotation level, since that information is already conveyed by the color. This layout is used by the `color` theme (see section 3.13 on page 21).

| Name | External | Description |
|---|---|---|
| `plain` | | Display contents as-is |
| `signature` | | Display signed contents |
| `color` | * | Display contents in color |
| `colorsig` | * | Display signed contents in color |

Table 2: Available environment layouts

`colorsig`

- The `colorsig` environment layout combines the features of the `signature` and `color` ones. This layout is used by the `colorsig` theme (see section 3.13 on page 21).

## 3.8 Controlling the layout of targets

As discussed in section 3.2 on page 7, the starred versions of the FiXme annotation commands and environments let you highlight a portion of text which is relevant to the current annotation. The exact way this textual target is typeset (in draft mode only; otherwise it is typeset as-is) is subject to a layout of its own, called the "target layout".

### 3.8.1 Selecting a layout

`targetlayout` The desired layout can be selected with the `targetlayout` option. Contrary to the annotation layouts, only one target layout can be active at a time. The `targetlayout` option is understood by the package itself, the `\fxsetup` command and all the starred versions of the annotation commands and environments. There are some restrictions on its usage however, as discussed in the next section.

`\fxusetargetlayout` `{`⟨*name*⟩`}`

An alternative way of selecting a target layout is to use the `\fxusetargetlayout` command, giving it the layout's name as its only, mandatory, argument.

### 3.8.2 Built-in *vs.* external layouts

Target layouts are provided either in the core of FiXme, or in separate files loaded dynamically on demand. Simple layouts are typically built-in, whereas those requiring additional packages are external, so that they don't consume TEX resources if not used. As a consequence, selecting an external layout with the `targetlayout` option might involve loading the relevant file first.

`\fxloadtargetlayouts` `{`⟨*name,...*⟩`}`

For technical reasons, it is not possible to do such a thing outside the preamble, neither in the middle of processing `\usepackage` options. As a result, the `targetlayout` option is restricted and you have two possibilities for using an external layout:

1. Use the `targetlayout` option in a call to `\fxsetup` in the preamble, like this: `\fxsetup{targetlayout=name}`. This will load it *and* select it immediately.

2. Use the `\fxusetargetlayout` command in the preamble like this: `\fxusetargetlayout{name}`. This is strictly equivalent to the previous solution.

| Name | External | Description |
|---|---|---|
| `plain` | | Display target as-is |
| `changebar` | * | Display a vertical bar aside target |
| `color` | * | Display target in color |
| `colorcb` | * | Display a colored vertical bar aside target |

Table 3: Available target layouts

3. If on the other hand you want to load one or several target layouts *without* using them immediately (perhaps in order to use them locally in some specific annotation), use the `\fxloadtargetlayouts` command in the preamble like this: `\fxloadtargetlayouts{`⟨*name*⟩`,...}`. After that, you can select any of those layouts anywhere you wish.

#### 3.8.3 Available layouts

Table 3 lists the target layouts currently distributed with FiXme.

`plain`

- The `plain` target layout displays its contents as-is, only in italics (by default) in order to distinguish it from the surrounding text.

`changebar`

- The `changebar` target layout displays a vertical bar in the margin, on the side of the target text.

`color`
`fxtarget`

- The `color` target layout uses the color named `fxtarget` to display the target text. This layout is used by the `color` and `colorsig` themes (see section 3.13 on page 21).

`colorcb`
`fxnote`
`fxwarning`
`fxerror`
`fxfatal`

- The `colorcb` target layout uses one of four colors named `fx`⟨*level*⟩ (according to the annotation's importance level) to display a colored vertical bar in the margin, on the side of the target text.

### 3.9 Faces

In the FiXme jargon, a "face" characterizes the visual aspect of some portion of text. If you're familiar with the Emacs editor, this will come as no surprise to you. FiXme provides several faces that allow you to further customize the layout of annotations or their targets.

#### 3.9.1 Setting face values

There are different ways to customize a face. The first one is to use the corresponding face option. For each face ⟨*name*⟩, their is a ⟨*name*⟩`face` option. For instance, the "inline" face is controlled by the `inlineface` option. Face options are understood by the package itself, the `\fxsetup` command and locally by all annotation commands or environments. Here is an example:

```
\fxsetup{inlineface=\bfseries}
```

Since you will probably want to use LaTeX commands in face values, you should know that LaTeX normally can't handle such commands in package options. If you want this to work, you need to use the `xkvltxp` package first, like this:

```
\usepackage{xkvltxp}
\usepackage[inlineface=\bfseries]{fixme}
```

`\fxsetface` `{`⟨*name*⟩`}{`⟨*value*⟩`}`
Another way to customize a face is to use the `\fxsetface` command by providing the face name and the face value as two mandatory arguments. For example:

```
\fxsetface{inline}{\bfseries}
```

### 3.9.2 Available faces

`inline` **The `inline` face** By default, the `inline` annotation layout displays its contents in bold font, to distinguish the note from the surrounding text. This is controlled by the `inline` face whose value is `\bfseries` by default.

`margin` **The `margin` face** By default, the `margin` and `marginclue` layouts display their contents in footnote size. This is controlled by the `margin` face whose value is `\footnotesize` by default.

`env` **The `env` face** By default, the `plain` environment layout displays its contents in bold font, to distinguish it from the surrounding text. This is controlled by the `env` face whose value is `\bfseries` by default. The `color` and `colorsig` environment layouts honor this face as well, but reset it to ⟨*nothing*⟩ first. You should probably keep the same value for the `inline` and `env` faces, since they are both used to display annotations within the document's body.

`signature` **The `signature` face** The `signature` environment layout honors the `env` face, and adds a `signature` face on top of it for the signature part. It is set to `\itshape` by default. The `colorsig` environment layout honors this face as well.

`target` **The `target` face** By default, the `plain` target layout displays its contents in italics, to distinguish it from the surrounding text. This is controlled by the `target` face whose value is `\itshape` by default. The `changebar`, `color` and `colorcb` target layouts honor this face as well, but reset it to ⟨*nothing*⟩ first.

## 3.10 Controlling the logging of annotations

As well as being displayed in the document itself, all annotations are "logged" in different ways: by default, simple notes are recorded in the log file while the others (warnings, errors and fatal errors) are also displayed on the terminal output during compilation.

`[no]silent` You have the ability to suppress logging altogether by using the `silent` option. This option is understood by the package itself, the `\fxsetup` command and all annotation commands and environments. Just as individual layout options, `silent` is a boolean option, so all those forms are possible: `silent`, equivalent to `silent=true`, and `nosilent`, equivalent to `silent=false` (the default).

## 3.11 Controlling the language of FiXme

### 3.11.1 Available languages

english french francais spanish italian german ngerman danish croatian lang

FiXme currently supports English (the default), French, Spanish, Italian, German, Danish and Croatian. You can select your preferred language by using the corresponding language option. These options usually appear in the call to `\documentclass` or `\usepackage`, but they are also understood by `\fxsetup` and all the annotation commands or environments. This allows you to change the selected language either globally or locally, and at any point in the document. The `french` and `francais` options are synonyms. The `german` and `ngerman` options are currently equivalent.

If you're manipulating language settings at the level of FiXme itself (as opposed to the `\documentclass` level), then the preferred way to specify a language is to use the `lang` option, and give it the language name as a value. For instance:

```
\usepackage[lang=french]{fixme}
```

### 3.11.2 Language tracking

langtrack

If the document you're working on has parts written in different languages, it might be the case that the annotations should follow the current language as well (especially if you're in collaborative mode; see section 3.12 on page 18). FiXme provides a boolean option named `langtrack`. When specified, FiXme assumes that you're using `babel` and automatically switches to the current language (as specified by `babel`'s `\languagename` command), without requiring an explicit language option.

defaultlang

In the case where tracking falls on a language unsupported by FiXme, a warning will be issued and FiXme will switch to the language specified by the `defaultlang` option (`english` by default). If you happen to get one of these warnings, please consider sending me a patch with support for this new language (see section 6.12 on page 46).

Finally, note that specifying a language explicitely (by means of a language option) in the annotation commands and environments always takes precedence over the language tracking behavior.

### 3.11.3 Indexing in different languages

If your document contains annotations written in different languages, and you have requested the `index` layout, FiXme will not only classify the notes by their level of importance, but also by language. For example, if you have FiXme warnings in both English and French, you will find two different subcategories for warnings in the index: one called "Warnings" and one called "Avertissements".

## 3.12 Standalone or collaborative mode

FiXme supports collaborative annotations as well as "standalone", single-author documents.

#### 3.12.1 Standalone mode

By default, FiXme is in standalone mode, meaning that it assumes there is only one person annotating the document. This has several implications on the layout. If you've tried it already, you may have noticed the following points.

- All the built-in annotation layouts (index excepted) put the FiXme logo in front of every note. This is also true for the environments. The idea is to distinguish FiXme contents from the rest of the document (for instance other marginal notes or footnotes).
- All annotations are indexed under the main FiXme category, and sorted by importance level, but the FiXme logo is not repeated constantly (that would be useless).
- Similarly, the list of fixmes does not clutter itself with the logo, because we already know that its contents is specific to FiXme.

As a matter of fact, when you see the FiXme logo appear somewhere, you're not actually contemplating it, but rather the annotation's *author*. It just happens that by default (meaning in standalone mode), the only author is FiXme itself.

`author` In standalone mode, you might be annoyed by this orgy of FiXme logos. This might happen if for instance you're using the `margin` layout and you *know* there is nothing but FiXme annotations in there. In such a case, you will most likely want to change the author to *nothing*. This can be acomplished by using the `author` option, which is understood by the package itself, the `\fxsetup` command and all the annotation commands or environments. Doing something like the following will get rid of the damn logo for good:

```
\usepackage[author=]{fixme}
```

#### 3.12.2 Collaborative mode

If, on the other hand, you're working in collaboration with other people, every potential "fixer" might want to tag his or her own annotations. So assuming that John Doe is another author, he would most likely do something like this:

```
\fxfatal[author=JD]{rephrase this}
```

And suddenly, John's fatal comment will be prefixed with his initials. This is not a very satisfactory solution however, because it would require you to explicitely provide the author's tag in every single note you create. Fortunately, FiXme offers an easier way to achieve this.

##### 3.12.2.1 Registering new authors

`\FXRegisterAuthor` `{`⟨*cmdprefix*⟩`}{`⟨*envprefix*⟩`}{`⟨*tag*⟩`}`

The command `\FXRegisterAuthor` registers a new author with FiXme. It takes three arguments: the last one (⟨*tag*⟩) is just the same as the value you would pass to the `author` option: it will serve as a prefix (or signature) for John's annotations. In addition to that, a complete new set of user-level commands (prefixed with

⟨*cmdprefix*⟩) and environments (prefixed with ⟨*envprefix*⟩) will be created. To clarify, suppose that we have registered John like this:

```
\FXRegisterAuthor{jd}{ajd}{JD}
```

Now, John can use the commands `\jdnote`, `\jdwarning` *etc.*, along with their starred versions, and he can also use the environments `ajdnote`, `ajdwarning` *etc.*, along with their starred versions as well. If you really want to know the whole story, it turns out that the main FiXme interface described in section 3.2 on page 7 is created with this single line of code:

```
\FXRegisterAuthor{fx}{anfx}{fixme}
```

**Warning!** ⟨*cmdprefix*⟩ and ⟨*envprefix*⟩ need to be different, or you will get very strange errors. The technical reason is that in LaTeX, an environment named `foo` is defined in terms of two commands: `\foo` and `\endfoo` (yes, this is silly; the first one should really be `\beginfoo`). As a consequence, if you use the same prefix, you will get a name clash between the annotation commands and environments.

#### 3.12.2.2 Fun with the `author` option

Some precisions about the author option are in order here. When a new author is registered with FiXme, the generated commands and environments work by *presetting* the author option to the specified ⟨*tag*⟩. This means that it is still possible to override it explicitely like this:

```
\jdfatal[author=Anonymous]{For $500.00, you got your Ph.D.}
```

I don't see any good reason for doing it though, the above example notwithstanding.

The final remark is about the default `fx*` user interface: the `fixme` default user is special in that it is the only registered user to honor a global `author` option (provided in the call to `\usepackage` or `\fxsetup`). The intended use of this is that the *main* author of the document uses the `fx*` interface (preferably with a personal `author` setting, different from the FiXme logo), and all other authors are registered via `\FXRegisterAuthor`.

#### 3.12.2.3 Globally switching to collaborative mode

We're getting close, but we're not quite there yet. Perhaps you would like to see the tags from the different authors in the list of fixmes, or even in the index? Remember that FiXme is in standalone mode by default, so the (only) tag does not appear in those places.

`singleuser` `multiuser` `mode`

If you want this additional information, you've got to ask FiXme to globally switch to collaborative mode. This can be done with either one of the three options `singleuser`, `mutliuser` or `mode`. `singleuser` and `multiuser` are boolean options. The `mode` option takes a value of either `singleuser` or `multiuser`. This is the preferred way to switch the mode. These options are understood globally by `\usepackage` or `\fxsetup`, and also locally by the annotation commands or environments.

When collaborative mode is active, FiXme adjusts the list of fixmes layout to display the authors tags as well. Additionally, the annotations are indexed as before, but additional index entries, sorted by author, are generated as well.

## 3.13 Themes

Themes are orthogonal to layouts: they provide a way to modify the overall appearance of FiXme by overriding the existing layouts and/or by providing new ones. In fact, a theme can be any kind of customization that you would otherwise put in your preamble.

### 3.13.1 Using themes

`theme` The interface for using a theme is quite simple: use the `theme` option and give it the name of the theme you want to use. Themes are always external: there are none in the core of FiXme but instead they are provided as independent files. As a consequence, the `theme` option has the same usage restrictions as all the layout options we've encountered so far. Moreover, it is not possible to "maintain" several themes and switch between them in a single document. Themes can be loaded only in the preamble.

`\fxusetheme` `{`⟨*name*⟩`}`
An alternative to the `theme` option is to use the `\fxusetheme` command, which takes the theme's name as its only mandatory argument.

### 3.13.2 Available themes

FiXme comes with a number of predefined themes listed below.

#### 3.13.2.1 The `signature` theme

`signature` This theme uses the `signature` environment layout (see section 3.7.3 on page 14), and overrides the built-in ones to display the author tags as a signature (*i.e.* at the end of the annotations) instead of as a prefix. All original layout faces are honored.

#### 3.13.2.2 The `color` theme

`color` This theme uses the `color` environment and target layouts (see sections 3.7.3 on page 14 and 3.8.3 on page 16), and overrides the built-in ones to use different colors for the different annotation levels. As a consequence, it also avoids printing the annotation names because this information is already contained in the colors themselves. All original layout faces are honored, but the `inline` one is reset to ⟨*nothing*⟩. Remember that the `env` and `target` faces are reset as well (this is actually done by the `color` environment and target layouts).

#### 3.13.2.3 The `colorsig` theme

`colorsig` This theme combines the features of the `color` and `signature` ones. All original layout faces are honored, but the `inline` one is reset to ⟨*nothing*⟩.

# 4 Extending FiXme

Hear hear, this is where you start spending more time hacking LaTeX than actually writing your document...

## 4.1 Modifying existing layouts

FiXme annotations, environment and target layouts are implemented as a (set of) commands conforming to strict prototypes. If you're not happy with the way they perform, you have the possibility to `\renewcommand` them (in fact, you should use `\renewcommand*` for annotation and environment layouts). In such a case, it is probably best to have a look at the code in order to figure out how the original ones are written. However, a description of their prototypes is given below.

### 4.1.1 Modifying existing annotation layouts

`\FXLayout...` {⟨*type*⟩}{⟨*annotation*⟩}{⟨*author*⟩}
Each annotation layout is implemented as a macro taking three mandatory arguments. By convention, this macro is named `\FXLayout`⟨*name*⟩, for instance `\FXLayoutInline`.⟨*type*⟩ is the annotation type. It can be one of `note`, `warning`, `error` and `fatal`. ⟨*annotation*⟩ is the annotation itself, and ⟨*author*⟩ is the author's tag.

### 4.1.2 Modifying existing environment layouts

`\FXEnvLayout...Begin` {⟨*type*⟩}{⟨*author*⟩}
`\FXEnvLayout...End`

Each environment layout is implemented as two macros taking two mandatory arguments. By convention, these macros are named `\FXEnvLayout`⟨*name*⟩`Begin` and `\FXEnvLayout`⟨*name*⟩`End`, for instance `\FXEnvLayoutPlainBegin` and `\FXEnvLayoutPlainEnd`. ⟨*type*⟩ is the annotation type. It can be one of `note`, `warning`, `error` and `fatal`. ⟨*author*⟩ is the author's tag.

### 4.1.3 Modifying existing target layouts

`\FXTargetLayout...` {⟨*type*⟩}{⟨*target*⟩}
Each target layout is implemented as a macro taking two mandatory arguments. By convention, this macro is named `\FXTargetLayout`⟨*name*⟩, for instance `\FXTargetLayoutPlain`. ⟨*type*⟩ is the annotation type. It can be one of `note`, `warning`, `error` and `fatal`. ⟨*target*⟩ is the textual target.

## 4.2 Creating new layouts

Creating a new layout first requires that you write new layout macros as described in the previous section. Once you've done that, the next step is to make FiXme aware of this addition. This is called "registering" a layout.

### 4.2.1 Registering a new annotation layout

#### 4.2.1.1 Early *vs.* late layouts

Normally, FiXme typesets your annotations at the current position in the text, using a sensible order for built-in layouts. For instance, the `footnote` layout, if active, is performed before the `inline` one, so that the footnote mark is sticked to the preceding text and not to the annotation. When using targeted commands or environments, the situation is a bit more complex: some layouts make more sense at the beginning of the textual target, and some others at the end. The former ones are called "early layouts" and the later ones are called "late layouts". A typical example of an early layout is the `margin` one: if you're highlighting a long portion of text, it is more convenient to see the marginal note appear near the top of that text, rather than near the end of it (a nice illustration of this is to combine the `changebar` target layout and `margin` annotation layout). As for built-in layouts, only the `margin` and `marginclue` ones are early. All others are late. When you create a new layout, you need to decide whether it is an early or a late one.

#### 4.2.1.2 Registering late layouts

`\FXRegisterLayout` `[`⟨*mutex*⟩`]{`⟨*name*⟩`}{`⟨*macro*⟩`}`

In order to register a late annotation layout with FiXme, use the command `\FXRegisterLayout`. This macro has two mandatory arguments: the layout ⟨*name*⟩ (at least 3 characters long) and the associated layout ⟨*macro*⟩. For instance, the `inline` layout is registered like this:

```
\FXRegisterLayout{inline}{\FXLayoutInline}
```

Once registered, the new layout gets a boolean option ⟨*name*⟩ and is also recognized by the `layout` and `morelayout` options, as well as by the `\fxuselayouts` command as ⟨*name*⟩.

The first (optional) argument ⟨*mutex*⟩ is a comma-separated list of other layout names that should be in mutual exclusion with the layout we are registering (for example, the `margin` and `marginclue` layouts are in mutual exclusion). Note that mutual exclusion between two layouts need only be registered once. In other words, a previsouly registered layout will automatically be made aware of subsequent mutex declarations.

#### 4.2.1.3 Registering early layouts

`\FXRegisterLayout*` `[`⟨*boolfunc*⟩`]{`⟨*name*⟩`}{`⟨*macro*⟩`}`

In order to register an early annotation layout with FiXme, use the starred form of `\FXRegisterLayout`. Everything else behaves the same.

#### 4.2.1.4 Providing a layout

`\FXProvidesLayout` `{`⟨*name*⟩`}[`⟨*release information*⟩`]`

If you want to save your layout externally, you need to store it in a file named `fxlayout`⟨*name*⟩`.sty` and advertise it by calling `\FXProvidesLayout`. It will then be recognized by the `\fxloadlayouts` command as ⟨*name*⟩.

### 4.2.2 Registering a new environment layout

\FXRegisterEnvLayout {⟨*name*⟩}{⟨*begin*⟩}{⟨*end*⟩}
In order to register a new environment layout with FiXme, use the command `\FXRegisterEnvLayout`. This macro has three mandatory arguments: the layout ⟨*name*⟩ and the associated ⟨*begin*⟩ and ⟨*end*⟩ macros. For instance, the `color` layout is registered like this:

```
\FXRegisterEnvLayout{color}{\FXEnvLayoutColorBegin}{\FXEnvLayoutColorEnd}
```

Once registered, the new layout is recognized by the `envlayout` option and by the `\fxuseenvlayout` command as ⟨*name*⟩.

\FXProvidesEnvLayout {⟨*name*⟩}[⟨*release information*⟩]
If you want to save your layout externally, you need to store it in a file named `fxenvlayout`⟨*name*⟩`.sty` and advertise it by calling `\FXProvidesEnvLayout`. It will then be recognized by the `\fxloadenvlayouts` commands as ⟨*name*⟩.

### 4.2.3 Registering a new target layout

\FXRegisterTargetLayout {⟨*name*⟩}{⟨*macro*⟩}
In order to register a new target layout with FiXme, use the command `\FXRegisterTargetLayout`. This macro has two mandatory arguments: the layout ⟨*name*⟩ and the associated ⟨*macro*⟩. For instance, the `color` layout is registered like this:

```
\FXRegisterTargetLayout{color}{\FXTargetLayoutColor}
```

Once registered, the new layout is recognized by the `targetlayout` option and by the `\fxusetargetlayout` as ⟨*name*⟩.

\FXProvidesTargetLayout {⟨*name*⟩}[⟨*release information*⟩]
If you want to save your layout externally, you need to store it in a file named `fxtargetlayout`⟨*name*⟩`.sty` and advertise it by calling `\FXProvidesTargetLayout`. It will then be recognized by the `\fxloadtargetlayouts` commands as ⟨*name*⟩.

## 4.3 Creating a new theme

Creating a new theme may involve anything from using (by way of `\fxsetup`) or modifying existing layouts, to providing new ones. If your new theme has specific layouts, you may consider writing them in seperate files as described before, in order to make them more generally available.

\FXRequireLayout
\FXRequireEnvLayout
\FXRequireTargetLayout
{⟨*name*⟩}
In order to use an external layout in a theme, use the commands `\FXRequire*Layout` and give them the layout's name as argument.

\FXProvidesTheme {⟨*name*⟩}[⟨*release information*⟩]
A theme should be saved in a file named `fxtheme`⟨*name*⟩`.sty` and advertised by calling `\FXProvidesTheme`. It will then be recognized by the `theme` option and the `\fxusetheme` command.

### 4.4 Internationalization

\fx...name
\fx...sname

FiXme's language control has been described in section 3.11 on page 18. For every supported language ⟨*lang*⟩, a number of macros define the language-dependent part of FiXme. The commands `\fx`⟨*lang*⟩`notename`, `\fx`⟨*lang*⟩`notesname`, and their equivalent for the other annotation levels define the singular and plural forms of the note names.

\...listfixmename

The title for the list of fixmes is defined by the command `\`⟨*lang*⟩`listfixmename`.

All of these commands may be renewed, and their values will be honored by FiXme in all situations, including potential language changes across the document.

## 5 History

v4.4 Handle existing yet empty lox file properly, meaning don't actually typeset an empty list of corrections.
Don't update the `lox` file in final mode, avoiding potential typesetting artifacts, reported by Lars Madsen.
Various internals and documentation improvements.

v4.3 Add a paragraph about the duplication of notes in captions, upon exchange with Kreuvf.
Update support for the KOMA-Script classes by using the `tocbasic` interface when available, reported by Dirk Surmann.
Separate inline notes from the text they follow, suggested by Victor Porton.
Fix potential inline layouts color leakage, reported by Victor Porton.
Fix several parsing problems when passing optional arguments containing brackets, thanks to Joseph Wright and Lars Madsen.

v4.2 Improve Danish translation, thanks to Lars Madsen.
Fix buglet in `\@wrindex` redefinition, reported by Norman Gray.

v4.1 8 new PDF-specific annotation layouts.
New annotation layout: `marginnote`, suggested by Sébastien Mengin.
Better mechanism for handling layout mutual exclusion.
Fix bug in inner layout processing.

v4.0 Support for collaborative annotations, suggested by Michael Kubovy.
Support for "targeted" notes and environments (highlighting a portion of text), suggested by Mark Edgington.
Support for "floating notes" (not specific to any portion of text), suggested by Rasmus Villemoes.
Support for alternative layout autoswitch in TEX's inner mode, suggested by Will Robertson.
Support for automatic language tracking in multilingual documents.
Support for themes.
Extended support for user-provided layouts.
Support for `key=value` argument syntax in the whole user interface.
New command `\fxsetup`.
Homogenize the log and console messages.
Heavy internals refactoring.

v3.4 `\fixme`, `\fxerror`, `\fxwarning` and `\fxnote` are now robust, thanks to Will Robertson.
Fix incompatibility with KOMA-Script classes version of `\@starttoc` when the lox file is inexistent, reported by Philipp Stephani.

v3.3 Document incompatibility between marginal layout and the ACM SIG classes, reported by Jochen Wuttke.
Honor `twoside` option in marginal layout, suggested by Jens Remus.
Support for KOMA-Script classes version 2006/07/30 v2.95b, suggested by Jens Remus.
Documentation improvements suggested by Brian van den Broek.
Fix incompatibility with `amsart` reported by Lars Madsen: `\@starttoc` takes two arguments.
Fix bug reported by Stefan Mann: a typo in the `\fixme@footnotetrue` macro name.

v3.2 Added the marginclue layout option which only signals a fixme in the margin, withtout the actual contents.
Support for Croatian thanks to Marcel Maretic.
Fix incompatibility with `amsbook` reported by Claude Lacoursière: `\@starttoc` takes two arguments.
Fix incompatibility with Beamer reported by Akim Demaille: protect contents of lox file.

v3.1 Fix bug reported by Arnold Beckmann: the environments were visible in final mode.

v3.0 Added environments corresponding to the annotation commands.
Added an optional first argument to the annotation commands to change the layout locally.
Fix bug reported by Akim Demaille: marginal notes could mess up the document's layout by flushing it right.

v2.2 New option `silent` to suppress notes logging.
Support for Danish thanks to Kim Rud Bille.

v2.1 Use `\nobreakspace` instead of the tilda character. This avoids conflicts with Babel in Spanish environments.
Fix bug reported by Knut Lickert: index entries were unconditionally built.

v2.0 New feature: note levels.
New feature: FiXme note counters and usage summary.
Suggestions from Kasper B. Graversen.
Support for Spanish thanks to Agustín Martín.

v1.5 New appearance option: `inline`.

v1.4 Support for the KOMA-Script classes.
Fix bug reported by Ulf Jaenicke-Roessler: the `\listoffixmes` command didn't work when called before the first FiXme note.

v1.3 Support for Italian thanks to Riccardo Murri.

v1.2 Support for German thanks to Harald Harders.

# 6 Implementation

## 6.1 Preamble

```
⟨fixme⟩
\NeedsTeXFormat{LaTeX2e}
⟨*header⟩
\ProvidesPackage{fixme}[2017/03/05 v4.4 Collaborative annotations for LaTeX2e]

⟨/header⟩
```

Some required packages:

```
⟨*fixme⟩
\RequirePackage{ifthen}
\RequirePackage{verbatim}
\RequirePackage{xkeyval}[2006/11/18]

⟨/fixme⟩
```

`\fixmelogo` The FiXme logo:

```
⟨*header⟩
\newcommand\fixmelogo{\textsf{FiXme}}

⟨/header⟩
```

## 6.2 Utilities

### 6.2.1 Miscellaneous

`\@fxpkginfo` `\@fxpkgwarning` {⟨*msg*⟩}

Issue a FiXme package info or warning:

```
⟨*fixme⟩
\newcommand\@fxpkginfo{\PackageInfo{FiXme}}
\newcommand\@fxpkgwarning{\PackageWarning{FiXme}}
```

`\@fxpkgerror` {⟨*shortmsg*⟩}{⟨*longmsg*⟩}

Issue a FiXme package error:

```
\newcommand\@fxpkgerror{\PackageError{FiXme}}

```

`\@fxaddtolist` {⟨*list*⟩}{⟨*elt*⟩}

Add ⟨*elt*⟩ at the end of ⟨*list*⟩. We should check for duplicates, but this is not currently done.

```
\newcommand*\@fxaddtolist[2]{%
  \expandafter\ifx\csname #1\endcsname\relax%
    \expandafter\def\csname #1\endcsname{#2}%
  \else%
    \expandafter\ifx\csname #1\endcsname\empty%
      \expandafter\g@addto@macro\csname #1\endcsname{#2}%
    \else%
      \expandafter\g@addto@macro\csname #1\endcsname{,#2}%
    \fi%
  \fi}

```

### 6.2.2 Key-value management (xkeyval)

#### 6.2.2.1 Shortcuts

The following macros are simple shortcuts for using `xkeyval` with the `fx` prefix.

`\@fxkeyifundefined` {⟨*families*⟩}{⟨*key*⟩}{⟨*then*⟩}{⟨*else*⟩}

```
\newcommand\@fxkeyifundefined{\key@ifundefined[fx]}
```

`\@fxdefinekey` {⟨*family*⟩}{⟨*key*⟩}[⟨*default*⟩]{⟨*function*⟩}

```
\newcommand\@fxdefinekey{\define@key[fx]}
```

`\@fxdefinecmdkey` {⟨*family*⟩}[⟨*mp*⟩]{⟨*key*⟩}[⟨*default*⟩]{⟨*function*⟩}

```
\newcommand\@fxdefinecmdkey{\define@cmdkey[fx]}
```

`\@fxdefinechoicekey` {⟨*family*⟩}{⟨*key*⟩}[⟨*bin*⟩]{⟨*alternatives*⟩}[⟨*default*⟩]{⟨*function*⟩}

```
\newcommand\@fxdefinechoicekey{\define@choicekey[fx]}
```

`\@fxsetkeys` {⟨*families*⟩}[⟨*na*⟩]{⟨*keys*⟩}

```
\newcommand\@fxsetkeys{\setkeys[fx]}
```

`\@fxpresetkeys` {⟨*families*⟩}{⟨*head keys*⟩}{⟨*tail keys*⟩}

```
%%       Note: currently unused
%%       \newcommand\@fxpresetkeys{\presetkeys[fx]}

```

#### 6.2.2.2 Wrappers

`\@fxvoidkeyerror` {⟨*key*⟩}{⟨*value*⟩}
Issue a FiXme error about a void key misuse (see below):

```
\newcommand*\@fxvoidkeyerror[2]{%
  \@fxpkgerror{misuse of key '#1'}{%
    You have given the key '#1' the argument '#2' but it takes
    none.\MessageBreak
    Type X to quit, fix that key and re-run LaTeX.\MessageBreak}}
```

`\@fxdefinevoidkey` {⟨*family*⟩}{⟨*name*⟩}{⟨*func*⟩}
A FiXme "void key" is an `xkeyval` key that doesn't expect any argument.

```
\newcommand*\@fxdefinevoidkey[3]{%
  \define@key[fx]{#1}{#2}[]{%
    \ifthenelse{\equal{##1}{}}{%
      #3}{%
      \@fxvoidkeyerror{#2}{##1}}}}

```

`\@fxdefineboolkey` [⟨*func*⟩]{⟨*family*⟩}{⟨*name*⟩}
A FiXme "boolean key" is like an `xkeyval` one, with the addition that for every such `key`, there is a `nokey` void key counterpart.

```
\newcommand*\@fxdefineboolkey[3][]{%
  \define@boolkey[fx]{#2}{#3}[true]{#1}
  \@fxdefinevoidkey{#2}{no#3}{\@nameuse{fx@#2@#3}{false}}}

```

## 6.3 List macros

### 6.3.1 Contents lines

`\l@fixme` We use the same layout as for the list of figures.

```
\let\l@fixme\l@figure
```

`\@fxdottedtocline` {⟨*tocdepth*⟩}{⟨*indent*⟩}{⟨*numwidth*⟩}{⟨*contents*⟩}{⟨*target*⟩}

This macro is copied almost verbatim from LaTeX's core. The intent is to do a similar layout, but replacing the last argument, normally a page number, by arbitrary text (in our case, a note's target). The original macro defines a restricted width to typeset the page number which is much too short for us, so we just let the ⟨*target*⟩ text take all the space it needs.

```
\newcommand*\@fxdottedtocline[5]{%
  \ifnum #1>\c@tocdepth \else
    \vskip \z@ \@plus.2\p@
    {\leftskip #2\relax \rightskip \@tocrmarg \parfillskip -\rightskip
     \parindent #2\relax\@afterindenttrue
     \interlinepenalty\@M
     \leavevmode
     \@tempdima #3\relax
     \advance\leftskip \@tempdima \null\nobreak\hskip -\leftskip
     {#4}\nobreak
     \leaders\hbox{$\m@th
       \mkern \@dotsep mu\hbox{.}\mkern \@dotsep
       mu$}\hfill
     \nobreak
     #5\par}%
  \fi}
```

`\fxcontentsline` {⟨*contents*⟩}{⟨*target*⟩}

Similar to LaTeX's `\contentsline` macro, but temporarily bind `\@dottedtocline` to our own version. The nice thing about this implementation is that we can still use `\l@fixme` (remember that it is bound to `\l@figure`) without exactly knowing what its definition is. This macro is at the user level because `\contentsline` is, but it is not currently documented in the user manual.

```
\newcommand*\fxcontentsline[2]{%
  \begingroup%
  \let\@dottedtocline\@fxdottedtocline%
  \l@fixme{#1}{#2}%
  \endgroup}

```

`\fxaddcontentsline` {⟨*contents*⟩}

Wrapper around LaTeX's `\addcontentsline` macro to handle the `target` option. If a specific target is provided, we can't use the normal `\addcontentsline` macro for reasons explained above, so we use our own version of `\contentsline` instead. This macro is at the user level because `\addcontentsline` is, but it is not currently documented in the user manual.

```
\newcommand*\fxaddcontentsline[1]{%
  \ifthenelse{\equal{\cmdfx@note@target}{thepage}}{%
    \addcontentsline{lox}{fixme}{#1}}{%
    \addtocontents{lox}{\protect\fxcontentsline{#1}{\cmdfx@note@target}}}}

```

### 6.3.2 List headers

FiXme recognizes the standard `article`, `report` and `book` classes, the AMS ones, and adapts the list header accordingly. It also detects when the package `basictoc` is loaded and uses it, which notably makes it compliant with the KOMA-Script classes as well. Otherwise, the standard `article` layout is used.

#### 6.3.2.1 `article` version

`\@lox@prtc@article`
`\@lox@psttc@article`

```
\newcommand\@lox@prtc@article{%
  \section*{\@fxlistfixmename%
    \@mkboth{\MakeUppercase\@fxlistfixmename}{\MakeUppercase\@fxlistfixmename}}}
\let\@lox@psttc@article\relax

```

#### 6.3.2.2 `report` version

`\@lox@prtc@report`
`\@lox@psttc@report`

```
\newcommand\@lox@prtc@report{%
  \if@twocolumn
    \@restonecoltrue\onecolumn
  \else
    \@restonecolfalse
  \fi
  \chapter*{\@fxlistfixmename%
    \@mkboth{\MakeUppercase\@fxlistfixmename}{\MakeUppercase\@fxlistfixmename}}}
\newcommand\@lox@psttc@report{\if@restonecol\twocolumn\fi}

```

#### 6.3.2.3 `book` version

`\@lox@prtc@book`
`\@lox@psttc@book`

```
\newcommand\@lox@prtc@book{%
  \if@twocolumn
    \@restonecoltrue\onecolumn
  \else
    \@restonecolfalse
  \fi
  \chapter*{\@fxlistfixmename%
    \@mkboth{\MakeUppercase\@fxlistfixmename}{\MakeUppercase\@fxlistfixmename}}}
\newcommand\@lox@psttc@book{\if@restonecol\twocolumn\fi}

```

### 6.3.3 Status/class-dependent implementation

`\lox@final`
`\lox@draft`

In the new implementation of the `draft` mode below, we not only check that the `lox` file exists, but also that it is not empty before actually typesetting anything.

```
\let\lox@final\relax

\newread\lox@file
\newif\iflox@typeset
\def\lox@eol{\par}
```

```
\newcommand\lox@draft{%
  \lox@typesetfalse%
  \openin\lox@file\jobname.lox\relax
  \ifeof\lox@file\else
    \read\lox@file to \lox@maybeeol
    \ifeof\lox@file
      \ifx\lox@maybeeol\lox@eol\else\lox@typesettrue\fi
    \else
      \lox@typesettrue
    \fi
  \fi
  \closein\lox@file
  \iflox@typeset\@lox@prtc\@starttoc{lox}\@lox@psttc\else\@starttoc{lox}\fi}
```

`\lox@draft@ams` The `amsbook` and `amsart` classes have the very ugly idea of redefining the `\@starttoc` macro to take two arguments. Therefore, I need to provide a specific version of the `\listoffixmes` macro:

```
\newcommand\lox@draft@ams{\@starttoc{lox}\@fxlistfixmename}

```

## 6.4 Faces

`\fxsetface` `{`⟨*name*⟩`}{`⟨*value*⟩`}`

```
\newcommand*\fxsetface[2]{\@fxsetkeys{face}{#1face=#2}}
```

`\@fxnewface` `[`⟨*default*⟩`]{`⟨*name*⟩`}`

A face is just a command key:

```
\newcommand*\@fxnewface[2][]{%
  \@fxdefinecmdkey{face}{#2face}{}%
  \fxsetface{#2}{#1}}
```

`\@fxuseface` `{`⟨*name*⟩`}`

```
\newcommand*\@fxuseface[1]{\@nameuse{cmdfx@face@#1face}}

```

## 6.5 Annotation layouts

### 6.5.1 Layout modes

`multiuser` `singleuser` `mode` These options specify whether FiXme should function in standalone or collaborative mode, allowing the different layouts to tweak their output.

```
\@fxdefineboolkey[%
  \ifthenelse{\equal{#1}{true}}{%
    \fx@mode@singleuserfalse}{%
    \fx@mode@singleusertrue}]{%
    mode}{multiuser}
\@fxdefineboolkey[%
  \ifthenelse{\equal{#1}{true}}{%
    \fx@mode@multiuserfalse}{%
    \fx@mode@multiusertrue}]{%
    mode}{singleuser}
\@fxdefinechoicekey{mode}{mode}{multiuser,singleuser}{\@fxsetkeys{mode}{#1}}

```

### 6.5.2 Layout creation

Separating between "early" and "late" layouts is needed in starred context, that is, when we are using targeted commands or environments.

`\@fxearlylayouts` `\@fxlatelayouts` Comma-separated lists of available early and late layouts.

```
\let\@fxearlylayouts\empty
\let\@fxlatelayouts\empty
```

`\FXProvidesLayout` `{`⟨*name*⟩`}[`⟨*release information*⟩`]`

```
\newcommand*\FXProvidesLayout[1]{\ProvidesPackage{fxlayout#1}}
```

`\@fxrecordlayoutmutex` `{`⟨*layout*⟩`}{`⟨*layouts*⟩`}`

Record mutual exclusion between ⟨*layout*⟩ and the comma-separated list of ⟨*layouts*⟩. For each ⟨*layout*⟩, the mutual exclusion list is stored in `\@fxlayout@`⟨*layout*⟩`@mutex`.

```
\newcommand*\@fxrecordlayoutmutex[2]{%
  \edef\@fxlts{\zap@space#2 \@empty}%
  \def\@fxexpr{\@fxaddtolist{@fxlayout@#1@mutex}}%
  \expandafter\@fxexpr\expandafter{\@fxlts}%
  \@for\@fxlt:=\@fxlts\do{\@fxaddtolist{@fxlayout@\@fxlt @mutex}{#1}}}
```

`\@fxhandlelayoutmutex` `{`⟨*layout*⟩`}`

Handle ⟨*layout*⟩'s mutual exclusion list.

```
\newcommand*\@fxhandlelayoutmutex[1]{%
  \ifthenelse{\boolean{fx@layout@#1}}{%
    \def\@fxexpr{\@for\@fxlt:=}%
    \expandafter\@fxexpr\csname @fxlayout@#1@mutex\endcsname\do{%
      \@ifundefined{iffx@layout@\@fxlt}{}{%
        \ifthenelse{\boolean{fx@layout@\@fxlt}}{%
          \@fxpkgwarning{%
            #1 layout requested;\MessageBreak
            turning \@fxlt\space layout off}%
          \@nameuse{fx@layout@\@fxlt}{false}}{}}}}{}}

```

`\@FXRegisterLayout` `{`⟨*when*⟩`}[`⟨*mutex*⟩`]{`⟨*name*⟩`}{`⟨*funcname*⟩`}`

Register a new layout with FiXme. This currently involves creating the boolean layout option with an optional function argument, constructing the translation macro to call the actual layout macro, and updating the appropriate layout list (early or late). The translation macro can't be `\let` to the real one, because themes might want to redefine latter. An optional mutual exclusion list may also be given.

```
\def\@FXRegisterLayout#1[#2]#3#4{%
  \@fxkeyifundefined{layout}{#3}{%
    \@fxrecordlayoutmutex{#3}{#2}%
    \@fxdefineboolkey[\@fxhandlelayoutmutex{#3}]{layout}{#3}%
    \expandafter\def\csname @fxlayout@#3\endcsname{#4}%
    \@fxaddtolist{@fx#1layouts}{#3}}{%
    \@fxpkgerror{layout '#3' already registered}{%
      You have called \string\FXRegisterLayout\space with a name already
      in use.\MessageBreak
      If you want to modify an existing layout, renew its
      command.\MessageBreak
      Otherwise, you must choose a different name.}}}
```

\FXRegisterLayout
\FXRegisterLayout*

⟨*⟩[⟨*boolfunc*⟩]{⟨*name*⟩}{⟨*funcname*⟩}
And the use-level interface:

```
\newcommand\FXRegisterLayout{%
  \@ifstar{%
    \@ifnextchar[%]
    {\@FXRegisterLayout{early}}{\@FXRegisterLayout{early}[]}}{%
    \@ifnextchar[%]
    {\@FXRegisterLayout{late}}{\@FXRegisterLayout{late}[]}}}

```

### 6.5.3 Standard textual dispositions

\@fxtextstd

{⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}
The standard text disposition.

```
\newcommand*\@fxtextstd[3]{\ignorespaces#3 \fxnotename{#1}: #2}
```

\@fxsignature

{⟨*author*⟩}
Typeset the signature part unless ⟨*author*⟩ is empty. Note that `\ifthenelse` is fragile, so we need to make the signature stuff robust.

```
\DeclareRobustCommand*\@fxsignature[1]{%
  \ifthenelse{\equal{#1}{}}{}{ -- {\@fxuseface{signature}#1}}}
```

\@fxsigstd

{⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}
The standard signature disposition.

```
\newcommand*\@fxsigstd[3]{\fxnotename{#1}: #2\@fxsignature{#3}}
```

### 6.5.4 Built-in layouts

Let's start with the early layouts, and continue with the late ones.

#### 6.5.4.1 Margin

margin

```
\@fxnewface{margin}
```

\FXLayoutMargin

{⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutMargin[3]{%
  \marginpar[\raggedleft\@fxuseface{margin}\@fxtextstd{#1}{#2}{#3}]{%
    \raggedright\@fxuseface{margin}\@fxtextstd{#1}{#2}{#3}}}
```

\@fxlayout@margin
[no]margin

```
\FXRegisterLayout*{margin}{\FXLayoutMargin}
```

#### 6.5.4.2 Margin clue

{⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

\FXLayoutMarginCLue

```
\newcommand*\FXLayoutMarginClue[3]{%
  \marginpar[%
  {\raggedleft\@fxuseface{margin}\ignorespaces#3 \fxnotename{#1}!}]{%
    \raggedright\@fxuseface{margin}\ignorespaces#3 \fxnotename{#1}!}}
```

\@fxlayout@marginclue
[no]marginclue

```
\FXRegisterLayout*[margin]{marginclue}{\FXLayoutMarginClue}
```

#### 6.5.4.3 Footnote

\FXLayoutFootnote

{⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutFootnote[3]{\footnote{\@fxtextstd{#1}{#2}{#3}}}
```

\@fxlayout@footnote

[no]footnote

```
\FXRegisterLayout{footnote}{\FXLayoutFootnote}
```

#### 6.5.4.4 Inline

inline

```
\@fxnewface{inline}
```

\FXLayoutInline

{⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutInline[3]{ \@fxuseface{inline}\@fxtextstd{#1}{#2}{#3}}
```

\@fxlayout@inline

[no]inline

```
\FXRegisterLayout{inline}{\FXLayoutInline}
```

#### 6.5.4.5 Index

\fixmeindexname

```
\newcommand\fixmeindexname{\fixmelogo}
```

\@wrindex

{⟨*contents*⟩}

A replacement for LaTeX's standard `\@wrindex` macro to deal with the `target` option. When given, it is supposed to replace the page number, just as in the list of fixmes.

```
\def\@wrindex#1{%
  \ifthenelse{\equal{\cmdfx@note@target}{thepage}}{%
    \protected@write\@indexfile{}{\string\indexentry{#1}{\thepage}}}{%
    \protected@write\@indexfile{}{\string\indexentry{#1}{\cmdfx@note@target}}}%
  \endgroup
  \@esphack}
```

\@fxnotekey
\@fxwarningkey
\@fxerrorkey
\@fxfatalkey

The keys used to sort indexed annotations by importance level:

```
\newcommand\@fxnotekey{***a}
\newcommand\@fxwarningkey{***b}
\newcommand\@fxerrorkey{***c}
\newcommand\@fxfatalkey{***d}
```

\FXLayoutIndex

{⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutIndex[3]{%
  \iffx@mode@multiuser%
    \index{***@\fixmeindexname:%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !\@nameuse{thefx#1count}: #3: #2}%
    \index{***#3@\fixmeindexname{} (#3):%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !\@nameuse{thefx#1count}: #2}%
  \else%
    \index{***@\fixmeindexname:%
```

```
    !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
    !\@nameuse{thefx#1count}: #2}%
\fi}
```

`\@fxlayout@index`

`[no]index`

```
\FXRegisterLayout{index}{\FXLayoutIndex}
```

#### 6.5.4.6 Contents line

The contents of the `lox` file is handled through this pseudo-layout. It follows the normal layout design, but is not registered the usual way because we don't want to give the user control over it. It is triggered explicitely by `\@@@fxnote@late@draft`.

`\FXLayoutContentsLine` {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutContentsLine[3]{%
  \iffx@mode@multiuser%
    \fxaddcontentsline{\@fxtextstd{#1}{#2}{#3}}%
  \else%
    \fxaddcontentsline{\fxnotename{#1}: #2}%
  \fi}

```

### 6.5.5 Layout loading

`\fxloadlayouts` {⟨*name,...*⟩}

```
\newcommand*\fxloadlayouts[1]{%
  \edef\@fxlts{\zap@space#1 \@empty}%
  \@for\@fxlt:=\@fxlts\do{\usepackage{fxlayout#1}}}

```

### 6.5.6 Layout control

`\@fxsetlayoutkeys` {⟨*keys*⟩} This macro would probably be overkill if we didn't need to `\expandafter` it at some point (See `\@fxhandleinnermode`).

```
\newcommand\@fxsetlayoutkeys{\@fxsetkeys{layout}}
```

`\@fxparselayout` Utility macro to detect the `no`⟨*name*⟩ form of layout options. The drawback of this technique is that layout options must be at least 3 characters long. No big deal though...

```
\def\@fxparselayout#1#2#3\relax{\def\@fxltprefix{#1#2}\def\@fxltrest{#3}}
% \begin{macro}{\fxuselayouts}
%   \marg{[no]names}\\
%   First, ensure that those layouts are available, then activate them.
%   \cs{\FXRequireLayouts} is a better style for theme programming.
%    \begin{macrocode}
\newcommand*\fxuselayouts[1]{%
  \edef\@fxlts{\zap@space#1 \@empty}%
  \@for\@fxlt:=\@fxlts\do{%
    \expandafter\@fxparselayout\@fxlt\relax%
    \ifthenelse{\equal{\@fxltprefix}{no}}{%
      \let\@fxltname\@fxltrest}{%
      \let\@fxltname\@fxlt}%
```

```
    \@fxkeyifundefined{layout}{\@fxltname}{\fxloadlayouts{\@fxltname}}{}}%
  \@fxsetkeys{layout}{#1}}
\let\FXRequireLayouts\fxuselayouts

```

innerlayout The alternative inner mode layout:

```
\@fxdefinecmdkey{layout}{innerlayout}{}
```

morelayout The `morelayout` option adds to the existing layout configuration. The implementation is trivial, as it simply boils down to calling `\setkeys` on its argument. There are several advantages in doing this.

1. It is possible to disable a layout by using the `no`⟨*layout*⟩ form. For example, `morelayout={inline,nomargin}` will work.
2. A wrong layout name (for instance, misspelled) will trigger an `xkeyval` error.

```
\@fxdefinekey{layout}{morelayout}{\fxuselayouts{#1}}
```

layout The `layout` option lets the user specify exactly which ones she wants to use. Not very difficult to implement either: it works by first deactivating all layouts, and then activating the provided ones as before. Note that the use of the `no`⟨*layout*⟩ form is valid but has no effect.

```
\@fxdefinekey{layout}{layout}{%
  \edef\@fxlayouts{\@fxearlylayouts,\@fxlatelayouts}%
  \@for\@fxlt:=\@fxlayouts\do{%
    \@nameuse{fx@layout@\@fxlt}{false}}%
  \fxuselayouts{#1}}

```

## 6.6 Environment Layouts

### 6.6.1 Layout creation

\FXProvidesEnvLayout `{`⟨*name*⟩`}[`⟨*release information*⟩`]`

```
\newcommand*\FXProvidesEnvLayout[1]{\ProvidesPackage{fxenvlayout#1}}
```

\FXRegisterEnvLayout `{`⟨*name*⟩`}{`⟨*beginfuncname*⟩`}{`⟨*endfuncname*⟩`}`

Register a new environment layout with FiXme. This currently only involves constructing the translation macros. The translation macros in question can't be `\let` to the real ones, because themes or users might want to redefine the latter.

```
\newcommand*\FXRegisterEnvLayout[3]{%
  \@ifundefined{@fxenvlayout@#1@begin}{%
    \expandafter\def\csname @fxenvlayout@#1@begin\endcsname{#2}%
    \expandafter\def\csname @fxenvlayout@#1@end\endcsname{#3}}{%
    \@fxpkgerror{environment layout '#2' already registered}{%
      You have called \string\FXRegisterEnvLayout\space with a name already in
      use.\MessageBreak
      If you want to modify an existing environment layout, renew its
      commands.\MessageBreak
      Otherwise, you must choose a different name.}}}

```

### 6.6.2 Built-in layouts

#### 6.6.2.1 Plain

env

```
\@fxnewface{env}
```

\FXEnvLayoutPlainBegin {⟨*type*⟩}{⟨*author*⟩}
\FXEnvLayoutPlainEnd

```
\newcommand*\FXEnvLayoutPlainBegin[2]{%
  \@fxuseface{env}\ignorespaces#2 \fxnotename{#1}: \ignorespaces}
\newcommand*\FXEnvLayoutPlainEnd[2]{}
```

\@fxenvlayout@plain@begin
\@fxenvlayout@plain@end

```
\FXRegisterEnvLayout{plain}{\FXEnvLayoutPlainBegin}{\FXEnvLayoutPlainEnd}

```

#### 6.6.2.2 Signature

signature
signature

```
\@fxnewface[\itshape]{signature}
```

\FXEnvLayoutSignatureBegin {⟨*type*⟩}{⟨*author*⟩}
\FXEnvLayoutSignatureEnd

```
\newcommand*\FXEnvLayoutSignatureBegin[2]{%
  \@fxuseface{env}\fxnotename{#1}: \ignorespaces}
\newcommand*\FXEnvLayoutSignatureEnd[2]{\@fxsignature{#2}}
```

\@fxenvlayout@signature@begin
\@fxenvlayout@signature@end

```
\FXRegisterEnvLayout{signature}{%
  \FXEnvLayoutSignatureBegin}{\FXEnvLayoutSignatureEnd}

```

### 6.6.3 Layout selection

\@fxselectenvlayout {⟨*name*⟩}

\@fxenvlayout@begin {⟨*type*⟩}{⟨*author*⟩}
\@fxenvlayout@end

This is much simpler than standard layout management because only one environment layout at a time is possible. Using a specific environment layout boils down to possibly loading it, and binding the beginning and ending macros to the proper translation ones.

```
\newcommand*\@fxselectenvlayout[1]{%
  \expandafter\let\expandafter\@fxenvlayout@begin%
    \csname @fxenvlayout@#1@begin\endcsname%
  \expandafter\let\expandafter\@fxenvlayout@end%
    \csname @fxenvlayout@#1@end\endcsname}

```

### 6.6.4 Layout loading

\fxloadenvlayouts {⟨*name,...*⟩}

```
\newcommand*\fxloadenvlayouts[1]{%
  \edef\@fxlts{\zap@space#1 \@empty}%
  \@for\@fxlt:=\@fxlts\do{\usepackage{fxenvlayout#1}}}

```

#### 6.6.5 Layout control

\fxuseenvlayout {⟨*name*⟩}

\FXRequireEnvLayout `\FXRequireEnvLayout` is a better style for theme programming.

```
\newcommand*\fxuseenvlayout[1]{%
  \@ifundefined{@fxenvlayout@#1@begin}{\fxloadenvlayouts{#1}}{}%
  \@fxselectenvlayout{#1}}
\let\FXRequireEnvLayout\fxuseenvlayout
```

envlayout

```
\@fxdefinekey{envlayout}{envlayout}{\fxuseenvlayout{#1}}

```

### 6.7 Target Layouts

#### 6.7.1 Layout creation

\FXProvidesTargetLayout {⟨*name*⟩}[⟨*release information*⟩]

```
\newcommand*\FXProvidesTargetLayout[1]{\ProvidesPackage{fxtargetlayout#1}}
```

\FXRegisterTargetLayout {⟨*name*⟩}{⟨*funcname*⟩}

Register a new target layout with FiXme. This currently only involves constructing the translation macro. The translation macro in question can't be `\let` to the real one, because themes or user might want to redefine the latter.

```
\newcommand*\FXRegisterTargetLayout[2]{%
  \@ifundefined{@fxtargetlayout@#1}{%
    \expandafter\def\csname @fxtargetlayout@#1\endcsname{#2}}{%
    \@fxpkgerror{target layout '#1' already registered}{%
      You have called \string\FXRegisterTargetLayout\space with a name
      already in use.\MessageBreak
      If you want to modify an existing target layout, renew its
      command.\MessageBreak
      Otherwise, you must choose another name.}}}

```

#### 6.7.2 Built-in layouts

##### 6.7.2.1 Plain

target

```
\@fxnewface{target}
```

\FXTargetLayoutPlain {⟨*target*⟩}

```
\newcommand\FXTargetLayoutPlain[2]{\@fxuseface{target}#2}
```

\@fxtargetlayout@plain

```
\FXRegisterTargetLayout{plain}{\FXTargetLayoutPlain}

```

### 6.7.3 Layout selection

`\@fxselecttargetlayout` {⟨*name*⟩}

`\@@fxtargetlayout` {⟨*target*⟩}

This is much simpler than standard layout management because only one target layout at a time is possible. Using a specific target layout boils down to possibly loading it, and binding the layout macro to the proper translation one.

```
\newcommand*\@fxselecttargetlayout[1]{%
  \expandafter\let\expandafter\@@fxtargetlayout%
    \csname @fxtargetlayout@#1\endcsname}

```

### 6.7.4 Target layout loading

`\fxloadtargetlayouts` {⟨*name,...*⟩}

```
\newcommand*\fxloadtargetlayouts[1]{%
  \edef\@fxlts{\zap@space#1 \@empty}%
  \@for\@fxlt:=\@fxlts\do{\usepackage{fxtargetlayout#1}}}

```

### 6.7.5 Target layout control

`\fxusetargetlayout` {⟨*name*⟩}

`\FXRequireTargetLayout` `\FXRequireTargetLayout` is a better style for theme programming.

```
\newcommand*\fxusetargetlayout[1]{%
  \@ifundefined{@fxtargetlayout@#1}{\fxloadtargetlayouts{#1}}{}%
  \@fxselecttargetlayout{#1}}
\let\FXRequireTargetLayout\fxusetargetlayout
```

`targetlayout`

```
\@fxdefinekey{targetlayout}{targetlayout}{\fxusetargetlayout{#1}}

```

### 6.7.6 Status-dependant versions

`\@fxtargetlayout@final` {⟨*target*⟩}

`\@fxtargetlayout@draft` In `final` mode, the target is typeset as-is. In `draft` mode, we use the selected layout.

```
\newcommand\@fxtargetlayout@final[2]{#2}
\newcommand\@fxtargetlayout@draft[2]{%
  \begingroup\@@fxtargetlayout{#1}{#2}\endgroup}

```

## 6.8 Logging

### 6.8.1 Logging macros

`\FXLogNote` {⟨*msg*⟩}

`\FXLogWarning`

`\FXLogerror`

`\FXLogFatal`

```
\newcommand*\FXLogNote[1]{%
  \GenericInfo{%
    (FiXme)\@spaces\@spaces\@spaces\@spaces}{%
    FiXme Note: '#1'}}
```

```
\newcommand*\FXLogWarning[1]{%
  \GenericWarning{%
    (FiXme)\@spaces\@spaces\@spaces\@spaces}{%
    FiXme Warning: '#1'}}
\newcommand*\FXLogError[1]{%
  \GenericWarning{%
    (FiXme)\@spaces\@spaces\@spaces\@spaces}{%
    FiXme Error: '#1'}}
\newcommand*\FXLogFatal[1]{%
  \GenericWarning{%
    (FiXme)\@spaces\@spaces\@spaces\@spaces}{%
    FiXme Fatal Error: '#1'}}

```

`\@fxlog@note` `\@fxlog@warning` `\@fxlog@error` `\@fxlog@fatal`

In order for the generic note dispatcher to be able to call the logging macros (see section 6.9.3 on page 42), we need an easier translation mechanism from the annotation type to the actual macro name. The translation macros in question can't be `\let` to the real one, because users might want to redefine the actual log macros later.

```
\def\@fxlog@note{\FXLogNote}
\def\@fxlog@warning{\FXLogWarning}
\def\@fxlog@error{\FXLogError}
\def\@fxlog@fatal{\FXLogFatal}

```

#### 6.8.2 Logging control

`[no]silent` Whether to log the annotations:

```
\@fxdefineboolkey{log}{silent}

```

### 6.9 FiXme notes

#### 6.9.1 Note parameters

`fixmecount` `fxnotecount` `fxwarningcount` `fxerrorcount` `fxfatalcount`

`fixmecount` maintains the total of all annotations, regardless of their level. Each note type also gets its own counter:

```
\newcounter{fixmecount}
\newcounter{fxnotecount}
\newcounter{fxwarningcount}
\newcounter{fxerrorcount}
\newcounter{fxfatalcount}

```

`author` An annotation "author" allows to distinguish notes from different persons in collaborative mode.

```
\@fxdefinecmdkey{note}{author}{}
```

`target` An annotation "target" may replace the page number in the list of corrections or in the index (see also section 6.5.4.6 on page 35).

```
\@fxdefinecmdkey{note}{target}{}
```

#### 6.9.2 Layout dispatch

`\@fxhandleinnermode` Handle the case where TEX is in inner mode. We use the alternative layout provided by the `innerlayout` option, and we make sure to disable both the `margin` and `marginclue` layout forms. This is done by appending `nomargin` and `nomarginclue` to the inner layout value (this also renders nasty user settings harmless). Before that, we provide some informative message if risky layout forms were active.

```
\newcommand\@fxhandleinnermode{%
  \ifinner%
    \ifthenelse{\boolean{fx@layout@margin}}{%
      \@fxpkginfo{%
        inner mode detected;\MessageBreak
        turning margin layout form off}}{%
      \ifthenelse{\boolean{fx@layout@marginclue}}{%
        \@fxpkginfo{%
          inner mode detected;\MessageBreak
          turning marginclue layout form off}}{}}%
    \expandafter\@fxsetlayoutkeys\expandafter{%
      \cmdfx@layout@innerlayout,nomargin,nomarginclue}%
  \fi}
```

`\@fxissueearlydraftlayouts` `\@fxissuelatedraftlayouts` {⟨*type*⟩}{⟨*note*⟩}
Dispatch all active draft mode layouts. `\@fxissueearlydraftlayouts` takes care of dispatching early layouts, but before that, handles the inner mode case. `\@fxissuelatedraftlayouts` just dispatches late layouts.

```
\newcommand*\@fxissueearlydraftlayouts[2]{%
  \@fxhandleinnermode%
  \@for\@fxlt:=\@fxearlylayouts\do{%
    \@nameuse{iffx@layout@\@fxlt}%
      \@nameuse{@fxlayout@\@fxlt}{#1}{#2}{\cmdfx@note@author}%
    \fi}}
\newcommand*\@fxissuelatedraftlayouts[2]{%
  \@for\@fxlt:=\@fxlatelayouts\do{%
    \@nameuse{iffx@layout@\@fxlt}%
      \@nameuse{@fxlayout@\@fxlt}{#1}{#2}{\cmdfx@note@author}%
    \fi}}
```

`\@fxissuecommonlayouts` {⟨*type*⟩}{⟨*note*⟩}
Dispatch all mode-independent layouts (actually, "layout" is to be taken in a slightly broader sense here). This macro executes all operations that need to be performed regardless of the document status. This currently means logging the annotations. Previously, this code also updated the `lox` file, but this could lead to typesetting artifacts even in final mode (because of the whatsit introduced by `\write`), which is highly undesirable, and besides, there's no point in keeping that information up to date, since it won't be typeset. So from now on, the contents lines are only generated in `draft` mode by `\@@@fxnote@late@draft`.

```
\newcommand*\@fxissuecommonlayouts[2]{%
  \iffx@log@silent\else\@nameuse{@fxlog@#1}{#2}\fi}

```

### 6.9.3 Status-dependent implementation

`\@@@fxnote@early@final` `\@@@fxnote@late@final` `\@@@fxnote@early@draft` `\@@@fxnote@late@draft`

{⟨*type*⟩}{⟨*note*⟩}
The lower-level macros that perform the real job. In `final` mode, early work is only to check for remaining fatal annotations and late work is to dispatch common layouts.

```
\newcommand*\@@@fxnote@early@final[2]{%
  \ifthenelse{\equal{#1}{fatal}}{%
    \@fxpkgerror{'#2' fatal error left in final version}{%
      You are currently processing in final mode,\MessageBreak
      but you still have some FiXme fatal errors left behind.\MessageBreak
      Type X to quit, fix your document (or switch back to draft
      mode),\MessageBreak
      and rerun LaTeX.}}{}}
\newcommand*\@@@fxnote@late@final[2]{\@fxissuecommonlayouts{#1}{#2}}
```

In `draft` mode, early work is to dispatch early layouts, while late work is to dispatch both late *and* common layouts, and update the `lox` file.

```
\newcommand*\@@@fxnote@early@draft[2]{%
  \@fxissueearlydraftlayouts{#1}{#2}}
\newcommand*\@@@fxnote@late@draft[2]{%
  \@fxissuelatedraftlayouts{#1}{#2}%
  \FXLayoutContentsLine{#1}{#2}{\cmdfx@note@author}%
  \@fxissuecommonlayouts{#1}{#2}}

```

### 6.9.4 Standard version

`\@fxpostconfigure`

This macro is used in `\@@fxnote@early` below, after processing user options (even when there is none), to postconfigure some aspects of the annotations. Currently, this involves two things: setting the author to `\fixmelogo` if it still is `fixme`, and automatically tracking the current language if required (note that all other language options turn tracking off, meaning that one can override language tracking locally by providing a language explicitely). Since environments need the post-configuration done sooner, they perform it themselves and rebind this macro to `\relax`.

```
\newcommand*\@fxpostconfigure{%
  \ifthenelse{\equal{\cmdfx@note@author}{fixme}}{%
    \@fxsetkeys{note}{author=\fixmelogo}}{}%
  \iffx@lang@langtrack%
    \@fxkeyifundefined{lang}{\languagename}{%
      \@fxpkgwarning{unknown language '\languagename';\MessageBreak
        falling back to \@fxdefaultlang}%
      \@fxsetkeys{lang}{\@fxdefaultlang}}{%
      \@fxsetkeys{lang}{\languagename}}
  \fi}

```

`\@fxendgroup`

This macro is used in `\@@fxnote@late` below to close the group opened at the user level. Since environments need the group opened for a longer time, they rebind it to `\relax` and close the group themselves later on.

```
\let\@fxendgroup\endgroup
```

`\@@fxnote@early` {⟨*type*⟩}{⟨*note*⟩}

Counters need to be updated regardless of the mode.

```
\def\@@fxnote@early#1#2{%
    \@fxpostconfigure%
    \stepcounter{fixmecount}%
    \stepcounter{fx#1count}%
    \@@@fxnote@early{#1}{#2}}
```

`\@@fxnote@late`

```
\def\@@fxnote@late#1#2{%
    \@@@fxnote@late{#1}{#2}%
  \@fxendgroup}
```

`\@@fxnote` {⟨*type*⟩}{⟨*note*⟩}

This macro is used everywhere outside a starred context, because in that case, we do early and late work in a row.

```
\def\@@fxnote#1#2{%
  \@@fxnote@early{#1}{#2}%
  \@@fxnote@late{#1}{#2}}
```

`\@fxnote` {⟨*type*⟩}[⟨*options*⟩]{⟨*note*⟩}

```
\def\@fxnote#1[#2]#3{%
  \@fxsetkeys{mode,status,lang,log,note,face,layout}{#2}%
  \@@fxnote{#1}{#3}}

```

### 6.9.5 Starred version

`\@@fxsnote` {⟨*type*⟩}{⟨*note*⟩}{⟨*text*⟩}

Post-configuration is done here because it's the code path confluent for all starred commands. Relaxing post-configuration afterwards is to prevent `\@@fxnote@early` from doing it again. Note that this is the only place where we actually do early and late work not in a row.

```
\long\def\@@fxsnote#1#2#3{%
  \@fxpostconfigure\let\@fxpostconfigure\relax%
  \@@fxnote@early{#1}{#2}\@fxtargetlayout{#1}{#3}\@@fxnote@late{#1}{#2}}
```

`\@fxsnote` {⟨*type*⟩}[⟨*options*⟩]{⟨*note*⟩}{⟨*text*⟩}

Note the `targetlayout` family here.

```
\long\def\@fxsnote#1[#2]#3#4{%
  \@fxsetkeys{mode,status,lang,log,note,face,layout,targetlayout}{#2}%
  \@@fxsnote{#1}{#3}{#4}}

```

### 6.9.6 User-level interface generation

`\@fxpreconfigure` {⟨*author*⟩}

This macro is used at the beginning of every user-level entry point (here for notes, and also in the environments section), to preconfigure some aspects of the annotations, before possibly processing options. Currently, this only involves presetting the note's author to the one specified in the call to `\FXRegisterAuthor`. This however is not done for the built-in `fixme` author, because this one should honor a global setting.

```
\newcommand*\@fxpreconfigure[1]{%
  \ifthenelse{\equal{#1}{fixme}}{}{\@fxsetkeys{note}{author=#1}}}
```

`\@fxnewnotemacro` {⟨*prefix*⟩}{⟨*type*⟩}{⟨*author*⟩}

This macro defines the user-level interface:

```
\newcommand*\@fxnewnotemacro[3]{%
  \expandafter\DeclareRobustCommand\csname #1#2\endcsname{%
    \begingroup%
      \@fxpreconfigure{#3}%
      \@ifstar{%
        \@ifnextchar[%]
        {\@fxsnote{#2}}{\@@fxsnote{#2}}}{%
        \@ifnextchar[%]
        {\@fxnote{#2}}{\@@fxnote{#2}}}}}
```

## 6.10 FiXme environments

A FiXme environment's summary is laid out by the corresponding macro, but the `inline` layout is disabled. This is as easy as appending `noinline` to the end of the options list.

### 6.10.1 Status-dependent implementation

`\@@@@fxbeginenv@final`
`\@@@@fxbeginenv@draft`
`\@fxendenv@final`
`\@fxendenv@draft`

{⟨*type*⟩}

In final mode, `verbatim`'s `comment` environment is used to suppress output.

```
\def\@@@@fxbeginenv@final#1{\comment}
\def\@@@@fxbeginenv@draft#1{\@fxenvlayout@begin{#1}{\cmdfx@note@author}}
\def\@fxendenv@final#1{\endcomment}
\def\@fxendenv@draft#1{\@fxenvlayout@end{#1}{\cmdfx@note@author}}

```

### 6.10.2 Standard versions

`\@@@fxbeginenv`
`\@@fxbeginenv`

{⟨*type*⟩}{⟨*summary*⟩}

Post-configuration is done here (it's the code path confluent for all non-starred environments). Relaxing post-configuration afterwards is to prevent `\@@fxnote` from doing it again.

```
\def\@@@fxbeginenv#1#2{%
  \@fxpostconfigure\let\@fxpostconfigure\relax%
  \@@fxnote{#1}{#2}%
  \@@@@fxbeginenv{#1}}
\def\@@fxbeginenv#1#2{%
  \@fxsetkeys{layout}{noinline}%
  \@@@fxbeginenv{#1}{#2}}
```

`\@fxbeginenv` {⟨*type*⟩}[⟨*options*⟩]{⟨*summary*⟩}

```
\def\@fxbeginenv#1[#2]#3{%
  \@fxsetkeys{mode,status,lang,log,note,face,layout,envlayout}{#2,noinline}%
  \@@@fxbeginenv{#1}{#3}}

```

#### 6.10.3 Starred versions

`\@@@fxbeginsenv` `\@@fxbeginsenv` {⟨*type*⟩}{⟨*summary*⟩}{⟨*text*⟩}

Post-configuration is done here (it's the code path confluent for all starred environments). Relaxing post-configuration afterwards is to prevent `\@@fxsnote` from doing it again.

```
\long\def\@@@fxbeginsenv#1#2#3{%
  \@fxpostconfigure\let\@fxpostconfigure\relax%
  \@@fxsnote{#1}{#2}{#3}%
  \@@@@fxbeginenv{#1}}
\long\def\@@fxbeginsenv#1#2#3{%
  \@fxsetkeys{layout}{noinline}%
  \@@@fxbeginsenv{#1}{#2}{#3}}
```

`\@fxbeginenv` {⟨*type*⟩}[⟨*options*⟩]{⟨*summary*⟩}{⟨*text*⟩}

Note the `targetlayout` family here.

```
\long\def\@fxbeginsenv#1[#2]#3#4{%
  \@fxsetkeys{mode,status,lang,log,note,face,layout,envlayout,targetlayout}{%
    #2,noinline}%
  \@@@fxbeginsenv{#1}{#3}{#4}}

```

#### 6.10.4 User-level interface generation

`\@fxnewnoteenvs` {⟨*prefix*⟩}{⟨*type*⟩}{⟨*author*⟩}

This macro defines the user-level interface. The ending macros are identical. Also, the environments close their own group, so we prevent `\@@fxnote` from doing so by temporarily rebinding `\@fxendgroup` to `\relax`.

```
\newcommand*\@fxnewnoteenvs[3]{%
  \expandafter\def\csname #1#2\endcsname{%
    \begingroup%
      \let\@fxendgroup\relax%
      \@fxpreconfigure{#3}%
      \@ifnextchar[%]
        {\@fxbeginenv{#2}}{\@@fxbeginenv{#2}}}
  \expandafter\def\csname end#1#2\endcsname{%
      \@fxendenv{#2}%
    \endgroup}%
  \expandafter\long\expandafter\def\csname #1#2*\endcsname{%
    \begingroup%
      \let\@fxendgroup\relax%
      \@fxpreconfigure{#3}%
      \@ifnextchar[%]
        {\@fxbeginsenv{#2}}{\@@fxbeginsenv{#2}}}
  \expandafter\def\csname end#1#2*\endcsname{%
      \@fxendenv{#2}%
    \endgroup}}

```

### 6.11 FiXme authors

`\FXRegisterAuthor` {⟨*cmdprefix*⟩}{⟨*envprefix*⟩}{⟨*name*⟩}

This macro creates the whole user-level interface for a particular author:

```
\newcommand*\FXRegisterAuthor[3]{%
  \@ifundefined{#1note}{}{%
    \@fxpkgerror{command prefix '#1' already in use}{%
      You have called \string\FXRegisterAuthor\space with a command prefix
      already in use.\MessageBreak
      Please choose another one.}}%
  \@ifundefined{#2note}{}{%
    \@fxpkgerror{environment prefix '#2' already in use}{%
      You have called \string\FXRegisterAuthor\space with an environment
      prefix already in use.\MessageBreak
      Please choose another one.}}%
  \@fxnewnotemacro{#1}{note}{#3}%
  \@fxnewnotemacro{#1}{warning}{#3}%
  \@fxnewnotemacro{#1}{error}{#3}%
  \@fxnewnotemacro{#1}{fatal}{#3}%
  \@fxnewnoteenvs{#2}{note}{#3}%
  \@fxnewnoteenvs{#2}{warning}{#3}%
  \@fxnewnoteenvs{#2}{error}{#3}%
  \@fxnewnoteenvs{#2}{fatal}{#3}}

```

`\fx...[*]` `anfx...[*]`

And we use it to create the FiXme default user:

```
\FXRegisterAuthor{fx}{anfx}{fixme}
```

`\fixme`

[⟨*options*⟩]{⟨*note*⟩}
Deprecate `\fixme`:

```
\DeclareRobustCommand\fixme{%
  \@fxpkgwarning{\string\fixme\space is deprecated;\MessageBreak
    please use \string\fxfatal\space instead}%
  \fxfatal}
```

`afixme`

Deprecate the `afixme` environment:

```
\def\afixme{%
  \@fxpkgwarning{The 'afixme' environment is deprecated;\MessageBreak
    please use 'anfxfatal' instead}%
  \anfxfatal}
\let\endafixme\endanfxfatal
```

## 6.12 Internationalization

`\@fxlanguages`

This macro lists all the supported languages, including aliases:

```
\newcommand*\@fxlanguages{%
  english,french,francais,spanish,italian,german,ngerman,danish,croatian}

```

### 6.12.1 Language definitions

#### 6.12.1.1 English

`english`
`\fxenglish...[s]name`

```
\newcommand\fxenglishnotename{Note}
\newcommand\fxenglishnotesname{Notes}
\newcommand\fxenglishwarningname{Warning}
```



`\englishlistfixmename`

```
\newcommand\fxenglishwarningsname{Warnings}
\newcommand\fxenglisherrorname{Error}
\newcommand\fxenglisherrorsname{Errors}
\newcommand\fxenglishfatalname{Fatal}
\newcommand\fxenglishfatalsname{Fatal errors}
\newcommand\englishlistfixmename{List of Corrections}

```

#### 6.12.1.2 French

french
francais
\fxfrench...[s]name

```
\newcommand\fxfrenchnotename{Note}
\newcommand\fxfrenchnotesname{Notes}
\newcommand\fxfrenchwarningname{Attention}
\newcommand\fxfrenchwarningsname{Avertissements}
\newcommand\fxfrencherrorname{Erreur}
\newcommand\fxfrencherrorsname{Erreurs}
\newcommand\fxfrenchfatalname{Fatal}
\newcommand\fxfrenchfatalsname{Erreurs fatales}
\newcommand\frenchlistfixmename{Liste des Corrections}

```

\frenchlistfixmename

#### 6.12.1.3 Spanish

spanish
\fxspanish...[s]name

```
\newcommand\fxspanishnotename{Nota}
\newcommand\fxspanishnotesname{Notas}
\newcommand\fxspanishwarningname{Aviso}
\newcommand\fxspanishwarningsname{Avisos}
\newcommand\fxspanisherrorname{Error}
\newcommand\fxspanisherrorsname{Errores}
\newcommand\fxspanishfatalname{Fatal}
\newcommand\fxspanishfatalsname{Errores fatales}
\newcommand\spanishlistfixmename{Lista de Correcciones}

```

\spanishlistfixmename

#### 6.12.1.4 Italian

italian
\fxitalian...[s]name

```
\newcommand\fxitaliannotename{Nota}
\newcommand\fxitaliannotesname{Note}
\newcommand\fxitalianwarningname{Avviso}
\newcommand\fxitalianwarningsname{Avvisi}
\newcommand\fxitalianerrorname{Errore}
\newcommand\fxitalianerrorsname{Errori}
\newcommand\fxitalianfatalname{Fatale}
\newcommand\fxitalianfatalsname{Errori fatali}
\newcommand\italianlistfixmename{Corrigenda}

```

\italianlistfixmename

#### 6.12.1.5 German

german
ngerman
\fxgerman...[s]name

```
\newcommand\fxgermannotename{Anm}
\newcommand\fxgermannotesname{Anmerkungen}
\newcommand\fxgermanwarningname{Warnung}
\newcommand\fxgermanwarningsname{Warnungen}
\newcommand\fxgermanerrorname{Fehler}
\newcommand\fxgermanerrorsname{Fehler}
\newcommand\fxgermanfatalname{Verh\"angnisvoll}
\newcommand\fxgermanfatalsname{Verh\"angnisvolle fehler}
\newcommand\germanlistfixmename{Verzeichnis der Korrekturen}

```

#### 6.12.1.6 Danish

danish
\fxdanish...[s]name
\danishlistfixmename

```
\newcommand\fxdanishnotename{Note}
\newcommand\fxdanishnotesname{Noter}
\newcommand\fxdanishwarningname{Advarsel}
\newcommand\fxdanishwarningsname{Advarsler}
\newcommand\fxdanisherrorname{Fejl}
\newcommand\fxdanisherrorsname{Fejl}
\newcommand\fxdanishfatalname{Fatal}
\newcommand\fxdanishfatalsname{Fatale fejl}
\newcommand\danishlistfixmename{Rettelser}

```

#### 6.12.1.7 Croatian

croatian
\fxcroatian...[s]name
\croatianlistfixmename

```
\newcommand\fxcroatiannotename{Poruka}
\newcommand\fxcroatiannotesname{Poruke}
\newcommand\fxcroatianwarningname{Upozorenja}
\newcommand\fxcroatianwarningsname{Upozorenje}
\newcommand\fxcroatianerrorname{Gre\v ska}
\newcommand\fxcroatianerrorsname{Greske}
\newcommand\fxcroatianfatalname{Fatalan}
\newcommand\fxcroatianfatalsname{Kobne gre\v ske}
\newcommand\croatianlistfixmename{Popis korekcija}

```

### 6.12.2 Language tracking

`langtrack` Whether to track the value of `\languagename` automatically:

```
\@fxdefineboolkey{lang}{langtrack}
```

`defaultlang` Which language to use when tracking leads to an unsuported language:

```
\def\@fxexpr{\@fxdefinechoicekey{lang}{defaultlang}[\@fxdefaultlang]}
\expandafter\@fxexpr\expandafter{\@fxlanguages}{}

```

### 6.12.3 Language options

`lang` `\@fxlang` Store the current language in `\@fxlang` after having handled language aliases, and disable language tracking:

```
\def\@fxexpr{\@fxdefinechoicekey{lang}{lang}[\@fxlang]}
\expandafter\@fxexpr\expandafter{\@fxlanguages}{%
  \ifthenelse{\equal{#1}{francais}}{\def\@fxlang{french}}{%
    \ifthenelse{\equal{#1}{ngerman}}{\def\@fxlang{german}}{}}%
  \@fxsetkeys{lang}{langtrack=false}}

```

english french francais spanish italian german ngerman danish croatian

Create individual language options:

```
\@for\@fxlg:=\@fxlanguages\do{
  \def\@fxexprone{\@fxdefinevoidkey{lang}}
  \edef\@fxexprtwo{{\@fxlg}{\noexpand\@fxsetkeys{lang}{lang=\@fxlg}}}
  \expandafter\@fxexprone\@fxexprtwo}

```

### 6.12.4 Language abstraction layer

\@fxlistfixmename

Construct the “list of fixmes” title in a language dependent fashion:

```
\newcommand*\@fxlistfixmename{\@nameuse{\@fxlang listfixmename}}
```

\fxnotename \fxnotesname

{⟨*type*⟩}

Construct the notes names in a language dependent fashion:

```
\newcommand*\fxnotename[1]{\@nameuse{fx\@fxlang#1name}}
\newcommand*\fxnotesname[1]{\@nameuse{fx\@fxlang#1sname}}

```

## 6.13 Document status processing

\@@@fxnote@early \@@@fxnote@late \@@@@fxbeginenv \@fxendenv \@fxtargetlayout \listoffixmes final draft status

Select draft or final versions of internal macros (some of them also depending on the document class):

```
\@fxdefinevoidkey{status}{final}{%
  \let\@@@fxnote@early\@@@fxnote@early@final%
  \let\@@@fxnote@late\@@@fxnote@late@final%
  \let\@@@@fxbeginenv\@@@@fxbeginenv@final
  \let\@fxendenv\@fxendenv@final%
  \let\@fxtargetlayout\@fxtargetlayout@final%
  \let\listoffixmes\lox@final}
\@fxdefinevoidkey{status}{draft}{%
  \let\@@@fxnote@early\@@@fxnote@early@draft%
  \let\@@@fxnote@late\@@@fxnote@late@draft%
  \let\@@@@fxbeginenv\@@@@fxbeginenv@draft
  \let\@fxendenv\@fxendenv@draft%
  \let\@fxtargetlayout\@fxtargetlayout@draft%
  \let\listoffixmes\lox@draft}
\@fxdefinechoicekey{status}{status}{final,draft}{\@fxsetkeys{status}{#1}}

```

## 6.14 Theme support

\FXProvidesTheme

{⟨*name*⟩}[⟨*release information*⟩]

```
\newcommand*\FXProvidesTheme[1]{\ProvidesPackage{fxtheme#1}}
```

\fxusetheme

{⟨*name*⟩}

```
\newcommand*\fxusetheme[1]{\usepackage{fxtheme#1}}
```

theme

```
\@fxdefinekey{theme}{theme}{\fxusetheme{#1}}
```

## 6.15 Finale

### 6.15.1 Class-dependent settings

Currently, our class dependencies only matter in draft mode, so one could argue that it is not optimal to handle this here. However, it would be incorrect to do it in the `draft` option code because this option can be switched at any point in the document (remember that it is understood even by the annotation macros and environments) and the stuff below should only be executed once. Besides, `\@ifclassloaded` is an `\@onlypreamble` macro...

As documented, marginal notes are incompatible with the ACM SIG classes. Initially, I thought I would detect these classes and issue an error if marginal layout (or clue) is active. However, I changed my mind, because nothing prevents somebody to write a new class on top of these ones and authorize `\marginpar` back again. Normally these classes issue an error if `\marginpar` is used. However, the 2.3 / June 2007 versions are buggy and the error actually triggers a stack overflow in LaTeX... (patch submitted). Oh boy, these classes are a mess.

\@lox@prtc
\@lox@psttc
\@lox@draft

```
\@ifclassloaded{article}{%
  \let\@lox@prtc\@lox@prtc@article%
  \let\@lox@psttc\@lox@psttc@article}{%
  \@ifclassloaded{report}{%
    \let\@lox@prtc\@lox@prtc@report%
    \let\@lox@psttc\@lox@psttc@report}{%
    \@ifclassloaded{book}{%
      \let\@lox@prtc\@lox@prtc@book%
      \let\@lox@psttc\@lox@psttc@book}{%
      \@ifclassloaded{amsbook}{%
        \let\lox@draft\lox@draft@ams}{%
        \@ifclassloaded{amsart}{%
          \let\lox@draft\lox@draft@ams}{%
          %% Use the article layout by default.
          \let\@lox@prtc\@lox@prtc@article%
          \let\@lox@psttc\@lox@psttc@article}}}}}

```

This overrides any previous class-based settings but makes the list of corrections compliant with the Koma-Script classes and any document using the `tocbasic` package.

```
\@ifpackageloaded{tocbasic}{%
  \addtotoclist[fixme]{lox}%
  \renewcommand\lox@draft{\listoftoc[\@fxlistfixmename]{lox}}}{}
```

### 6.15.2 Options Processing

First, we execute some options to initialize FiXme to something sensible, and then we process the user ones. Note the abscence of the `theme` family here.

```
\ExecuteOptionsX[fx]<%
  mode,status,lang,log,note,face,layout,envlayout,targetlayout>{%
```

```
  mode=singleuser,%
  status=final,%
  lang=english,%
  langtrack=false,%
  defaultlang=english,%
  nosilent,%
  author=fixme,%
  target=thepage,%
  layout=margin,%
  innerlayout={layout=inline},%
  envlayout=plain,%
  targetlayout=plain,%
  inlineface=\bfseries,%
  marginface=\footnotesize,%
  envface=\bfseries,%
  targetface=\itshape}
\ProcessOptionsX*[fx]<%
  mode,status,lang,log,note,face,layout,envlayout,targetlayout>

```

### 6.15.3 The `\fxsetup` macro

`\fxsetup` {⟨*options*⟩}

The inevitable setup macro, extremely impressive yet as trivial as can be with the `xkeyval` package. . . `\fxsetup` is the only place where the `theme` family is processed.

```
\newcommand*\fxsetup[1]{%
  \@fxsetkeys{%
    mode,status,lang,log,note,face,layout,envlayout,targetlayout,theme}{%
    #1}}

```

### 6.15.4 FiXme summary

Finally, output a summary giving the number of fixme notes at the end of the compilation:

```
\AtEndDocument{%
  \iffx@log@silent\else
    \GenericWarning{%
      (FiXme)\@spaces\@spaces}{%
      FiXme Summary: Number of notes: \thefxnotecount,\MessageBreak%
      Number of warnings: \thefxwarningcount,\MessageBreak%
      Number of errors: \thefxerrorcount,\MessageBreak%
      Number of fatal errors: \thefxfatalcount,\MessageBreak%
      Total: \thefixmecount\@gobble}%
  \fi}
⟨/fixme⟩
```

# A External Layouts

## A.1 Annotation layouts

### A.1.1 The marginnote layout

marginnote

```
⟨*fxlayoutmarginnote⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{marginnote}

\RequirePackage{marginnote}

```

\FXLayoutMarginNote {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutMarginNote[3]{%
  \marginnote[\raggedleft\@fxuseface{margin}\@fxtextstd{#1}{#2}{#3}]{%
    \raggedright\@fxuseface{margin}\@fxtextstd{#1}{#2}{#3}}}
```

\@fxlayout@marginnote

[no]marginnote

```
\FXRegisterLayout*[margin,marginclue]{marginnote}{\FXLayoutMarginNote}
⟨/fxlayoutmarginnote⟩
```

### A.1.2 The pdfnote layout

pdfnote

```
⟨*fxlayoutpdfnote⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{pdfnote}

\RequirePackage{pdfcomment}

```

\FXLayoutPDFNote {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutPDFNote[3]{%
  \pdfcomment[author={#3}]{\@fxtextstd{#1}{#2}{#3}}}
```

\@fxlayout@pdfnote

[no]pdfnote

```
\FXRegisterLayout{pdfnote}{\FXLayoutPDFNote}
⟨/fxlayoutpdfnote⟩
```

### A.1.3 The pdfmargin layout

pdfmargin

```
⟨*fxlayoutpdfmargin⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{pdfmargin}

\RequirePackage{pdfcomment}

```

\FXLayoutPDFMargin {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutPDFMargin[3]{%
  \pdfmargincomment[author={#3}]{\@fxtextstd{#1}{#2}{#3}}}
```

`\@fxlayout@pdfmargin`

`[no]pdfmargin`

```
\FXRegisterLayout*[margin,marginclue,marginnote]{pdfmargin}{%
  \FXLayoutPDFMargin}
⟨/fxlayoutpdfmargin⟩
```

### A.1.4 The `pdfsignote` layout

`pdfsignote`

```
⟨*fxlayoutpdfsignote⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{pdfsignote}

\RequirePackage{pdfcomment}

```

`\FXLayoutPDFSigNote` {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutPDFSigNote[3]{%
  \pdfcomment[author={#3}]{\@fxsigstd{#1}{#2}{#3}}}
```

`\@fxlayout@pdfsignote`

`[no]pdfsignote`

```
\FXRegisterLayout[pdfnote]{pdfsignote}{\FXLayoutPDFSigNote}
⟨/fxlayoutpdfsignote⟩
```

### A.1.5 The `pdfsigmargin` layout

`pdfsigmargin`

```
⟨*fxlayoutpdfsigmargin⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{pdfsigmargin}

\RequirePackage{pdfcomment}

```

`\FXLayoutPDFSigMargin` {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutPDFSigMargin[3]{%
  \pdfmargincomment[author={#3}]{\@fxsigstd{#1}{#2}{#3}}}
```

`\@fxlayout@pdfsigmargin`

`[no]pdfsigmargin`

```
\FXRegisterLayout*[margin,marginclue,marginnote,pdfmargin]{pdfsigmargin}{%
  \FXLayoutPDFSigMargin}
⟨/fxlayoutpdfsigmargin⟩
```

### A.1.6 The `pdfcnote` layout

`pdfcnote`

```
⟨*fxlayoutpdfcnote⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{pdfcnote}

\RequirePackage{pdfcomment}
\RequirePackage{xcolor}

```

`fxnote` `fxwarning` `fxerror` `fxfatal`

Environments use the same colors as the notes themselves because their contents really is a longer note.

```
\definecolor{fxnote}{rgb}{0.0000,0.6000,0.0000}
\definecolor{fxwarning}{rgb}{1.0000,0.5490,0.0000}
\definecolor{fxerror}{rgb}{1.0000,0.2706,0.0000}
\definecolor{fxfatal}{rgb}{1.0000,0.0000,0.0000}

```

`\@fxdocolon` {⟨*author*⟩}

Add a colon after the author tag, unless empty.

```
\providecommand*\@fxdocolon[1]{%
  \ifthenelse{\equal{#1}{}}{\def\@fxcolon{}}{\def\@fxcolon{: }}}

```

`\FXLayoutPDFCNote` {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutPDFCNote[3]{%
  \@fxdocolon{#3}%
  \pdfcomment[author={#3},color={fx#1}]{\ignorespaces#3\@fxcolon#2}}
```

`\@fxlayout@pdfcnote` `[no]pdfcnote`

```
\FXRegisterLayout[pdfnote]{pdfcnote}{\FXLayoutPDFCNote}
⟨/fxlayoutpdfcnote⟩
```

### A.1.7 The `pdfcmargin` layout

`pdfcmargin`

```
⟨*fxlayoutpdfcmargin⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{pdfcmargin}

\RequirePackage{pdfcomment}
\RequirePackage{xcolor}

```

`fxnote` `fxwarning` `fxerror` `fxfatal`

Environments use the same colors as the notes themselves because their contents really is a longer note.

```
\definecolor{fxnote}{rgb}{0.0000,0.6000,0.0000}
\definecolor{fxwarning}{rgb}{1.0000,0.5490,0.0000}
\definecolor{fxerror}{rgb}{1.0000,0.2706,0.0000}
\definecolor{fxfatal}{rgb}{1.0000,0.0000,0.0000}

```

`\@fxdocolon` {⟨*author*⟩}

Add a colon after the author tag, unless empty.

```
\providecommand*\@fxdocolon[1]{%
  \ifthenelse{\equal{#1}{}}{\def\@fxcolon{}}{\def\@fxcolon{: }}}

```

`\FXLayoutPDFCMargin` {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutPDFCMargin[3]{%
  \@fxdocolon{#3}%
  \pdfmargincomment[author={#3},color={fx#1}]{\ignorespaces#3\@fxcolon#2}}
```

\@fxlayout@pdfcmargin

[no]pdfcmargin

```
\FXRegisterLayout*[margin,marginclue,marginnote,pdfmargin]{pdfcmargin}{%
  \FXLayoutPDFCMargin}
⟨/fxlayoutpdfcmargin⟩
```

### A.1.8 The `pdfcsignote` layout

pdfcsignote

```
⟨*fxlayoutpdfcsignote⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{pdfcsignote}

\RequirePackage{pdfcomment}
\RequirePackage{xcolor}

```

fxnote fxwarning fxerror fxfatal

Environments use the same colors as the notes themselves because their contents really is a longer note.

```
\definecolor{fxnote}{rgb}{0.0000,0.6000,0.0000}
\definecolor{fxwarning}{rgb}{1.0000,0.5490,0.0000}
\definecolor{fxerror}{rgb}{1.0000,0.2706,0.0000}
\definecolor{fxfatal}{rgb}{1.0000,0.0000,0.0000}

```

\FXLayoutPDFCSigNote {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutPDFCSigNote[3]{%
  \pdfcomment[author={#3},color={fx#1}]{#2\@fxsignature{#3}}}
```

\@fxlayout@pdfcsignote

[no]pdfcsignote

```
\FXRegisterLayout[pdfnote,pdfcnote]{pdfcsignote}{\FXLayoutPDFCSigNote}
⟨/fxlayoutpdfcsignote⟩
```

### A.1.9 The `pdfcsigmargin` layout

pdfcsigmargin

```
⟨*fxlayoutpdfcsigmargin⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesLayout{pdfcsigmargin}

\RequirePackage{pdfcomment}
\RequirePackage{xcolor}

```

fxnote fxwarning fxerror fxfatal

Environments use the same colors as the notes themselves because their contents really is a longer note.

```
\definecolor{fxnote}{rgb}{0.0000,0.6000,0.0000}
\definecolor{fxwarning}{rgb}{1.0000,0.5490,0.0000}
\definecolor{fxerror}{rgb}{1.0000,0.2706,0.0000}
\definecolor{fxfatal}{rgb}{1.0000,0.0000,0.0000}

```

`\FXLayoutPDFCSigMargin` {⟨*type*⟩}{⟨*note*⟩}{⟨*author*⟩}

```
\newcommand*\FXLayoutPDFCSigMargin[3]{%
  \pdfmargincomment[author={#3},color={fx#1}]{#2\@fxsignature{#3}}}
```

`\@fxlayout@pdfcsigmargin`
`[no]pdfcsigmargin`

```
\FXRegisterLayout*[margin,marginclue,marginnote,pdfmargin,pdfsigmargin]{%
  pdfcsigmargin}{%
  \FXLayoutPDFCSigMargin}
⟨/fxlayoutpdfcsigmargin⟩
```

## A.2 Environment layouts

### A.2.1 The color layout

`color`

```
⟨*fxenvlayoutcolor⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesEnvLayout{color}

\RequirePackage{color}

```

`\@fxdocolon` {⟨*author*⟩}

Add a colon after the author tag, unless empty.

```
\providecommand*\@fxdocolon[1]{%
  \ifthenelse{\equal{#1}{}}{\def\@fxcolon{}}{\def\@fxcolon{: }}}

```

`fxnote`
`fxwarning`
`fxerror`
`fxfatal`

Environments use the same colors as the notes themselves because their contents really is a longer note.

```
\definecolor{fxnote}{rgb}{0.0000,0.6000,0.0000}
\definecolor{fxwarning}{rgb}{1.0000,0.5490,0.0000}
\definecolor{fxerror}{rgb}{1.0000,0.2706,0.0000}
\definecolor{fxfatal}{rgb}{1.0000,0.0000,0.0000}

\fxsetface{env}{}

```

`\FXEnvLayoutColorBegin` {⟨*type*⟩}{⟨*author*⟩}
`\FXEnvLayoutColorEnd`

```
\newcommand*\FXEnvLayoutColorBegin[2]{%
  \@fxdocolon{#2}%
  \@fxuseface{env}\color{fx#1}\ignorespaces#2\@fxcolon\ignorespaces}
\newcommand*\FXEnvLayoutColorEnd[2]{}
```

`\@fxenvlayout@color@begin`
`\@fxenvlayout@color@end`

```
\FXRegisterEnvLayout{color}{\FXEnvLayoutColorBegin}{\FXEnvLayoutColorEnd}
⟨/fxenvlayoutcolor⟩
```

### A.2.2 The `colorsig` layout

colorsig

```
⟨*fxenvlayoutcolorsig⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesEnvLayout{colorsig}

\RequirePackage{color}

```

signature

```
\@fxnewface[\itshape]{signature}
```

fxnote fxwarning fxerror fxfatal

Environments use the same colors as the notes themselves because their contents really is a longer note.

```
\definecolor{fxnote}{rgb}{0.0000,0.6000,0.0000}
\definecolor{fxwarning}{rgb}{1.0000,0.5490,0.0000}
\definecolor{fxerror}{rgb}{1.0000,0.2706,0.0000}
\definecolor{fxfatal}{rgb}{1.0000,0.0000,0.0000}

\fxsetface{env}{}

```

\FXEnvLayoutColorSigBegin \FXEnvLayoutColorSigEnd

{⟨*type*⟩}{⟨*author*⟩}

```
\newcommand*\FXEnvLayoutColorSigBegin[2]{\@fxuseface{env}\color{fx#1}}
\newcommand*\FXEnvLayoutColorSigEnd[2]{\@fxsignature{#2}}
```

\@fxenvlayout@colorsig@begin \@fxenvlayout@colorsig@end

```
\FXRegisterEnvLayout{colorsig}{%
  \FXEnvLayoutColorSigBegin}{\FXEnvLayoutColorSigEnd}
⟨/fxenvlayoutcolorsig⟩
```

## A.3 Target Layouts

Since target layouts don't include author information, they're orthogonal to (and hence usable in) prefix/signature display.

### A.3.1 The `changebar` layout

changebar

```
⟨*fxtargetlayoutchangebar⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesTargetLayout{changebar}

\RequirePackage{changebar}
\setlength{\changebarsep}{5pt}

\fxsetface{target}{}
```

\FXTargetLayoutChangeBar

{⟨*target*⟩}

```
\newcommand\FXTargetLayoutChangeBar[2]{\cbstart\@fxuseface{target}#2\cbend}
```

\@fxtargetlayout@changebar

```
\FXRegisterTargetLayout{changebar}{\FXTargetLayoutChangeBar}
⟨/fxtargetlayoutchangebar⟩
```

### A.3.2 The `color` layout

`color`

```
⟨*fxtargetlayoutcolor⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesTargetLayout{color}

\RequirePackage{color}
\definecolor{fxnote}{rgb}{0.0000,0.6000,0.0000}
\definecolor{fxwarning}{rgb}{1.0000,0.5490,0.0000}
\definecolor{fxerror}{rgb}{1.0000,0.2706,0.0000}
\definecolor{fxfatal}{rgb}{1.0000,0.0000,0.0000}

```

`fxtarget`

```
\definecolor{fxtarget}{rgb}{0.3725,0.6196,0.6275}

\fxsetface{target}{}

```

`\FXTargetLayoutColor` {⟨*target*⟩}

```
\newcommand\FXTargetLayoutColor[2]{\@fxuseface{target}\color{fxtarget}#2}
```

`\@fxtargetlayout@color`

```
\FXRegisterTargetLayout{color}{\FXTargetLayoutColor}
⟨/fxtargetlayoutcolor⟩
```

### A.3.3 The `colorcb` layout

`colorcb`

```
⟨*fxtargetlayoutcolorcb⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesTargetLayout{colorcb}

\RequirePackage{color}

\RequirePackage[color]{changebar}
\setlength{\changebarsep}{5pt}

\fxsetface{target}{}
```

`\FXTargetLayoutColorCB` {⟨*target*⟩}

```
\newcommand\FXTargetLayoutColorCB[2]{%
  \cbstart\cbcolor{fx#1}\@fxuseface{target}#2\cbend}
```

`\@fxtargetlayout@colorcb`

```
\FXRegisterTargetLayout{colorcb}{\FXTargetLayoutColorCB}
⟨/fxtargetlayoutcolorcb⟩
```

# B Themes

## B.1 The signature theme

signature

```
⟨*fxthemesignature⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesTheme{signature}

\fxuseenvlayout{signature}

\renewcommand*\FXLayoutFootnote[3]{\footnote{\@fxsigstd{#1}{#2}{#3}}}
\renewcommand*\FXLayoutMargin[3]{%
  \marginpar[{\raggedleft\@fxuseface{margin}\@fxsigstd{#1}{#2}{#3}}]{%
    \raggedright\@fxuseface{margin}\@fxsigstd{#1}{#2}{#3}}}
\renewcommand*\FXLayoutMarginClue[3]{%
  \marginpar[{\raggedleft\@fxuseface{margin}\fxnotename{#1}!\@fxsignature{#3}}]{%
    \raggedright\@fxuseface{margin}\fxnotename{#1}!\@fxsignature{#3}}}
\renewcommand*\FXLayoutInline[3]{{ \@fxuseface{inline}\@fxsigstd{#1}{#2}{#3}}}
\renewcommand*\FXLayoutIndex[3]{%
  \iffx@mode@multiuser%
    \index{***@\fixmeindexname:%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !\@nameuse{thefx#1count}: #2\@fxsignature{#3}}%
    \index{***#3@\fixmeindexname{} (#3):%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !\@nameuse{thefx#1count}: #2}%
  \else%
    \index{***@\fixmeindexname:%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !\@nameuse{thefx#1count}: #2}%
  \fi}
\renewcommand*\FXLayoutContentsLine[3]{%
  \iffx@mode@multiuser%
    \fxaddcontentsline{\@fxsigstd{#1}{#2}{#3}}%
  \else%
    \fxaddcontentsline{\fxnotename{#1}: #2}%
  \fi}
⟨/fxthemesignature⟩
```

## B.2 The color theme

color

```
⟨*fxthemecolor⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesTheme{color}

\RequirePackage{color}

\FXRequireEnvLayout{color}
\FXRequireTargetLayout{color}

\fxsetface{inline}{}
```

```

\renewcommand*\FXLayoutFootnote[3]{%
  \@fxdocolon{#3}%
  \footnote{\color{fx#1}\ignorespaces#3\@fxcolon#2}}
\renewcommand*\FXLayoutMargin[3]{%
  \@fxdocolon{#3}%
  \marginpar[%
  {\raggedleft\@fxuseface{margin}\color{fx#1}\ignorespaces#3\@fxcolon#2}]{%
    \raggedright\@fxuseface{margin}\color{fx#1}\ignorespaces#3\@fxcolon#2}}
\renewcommand*\FXLayoutMarginClue[3]{%
  \marginpar[{\raggedleft\@fxuseface{margin}\color{fx#1}\ignorespaces#3!}]{%
    \raggedright\@fxuseface{margin}\color{fx#1}\ignorespaces#3!}}
\renewcommand*\FXLayoutInline[3]{%
  \@fxdocolon{#3}%
  { \textcolor{fx#1}{\@fxuseface{inline}\ignorespaces#3\@fxcolon#2}}}
\renewcommand*\FXLayoutIndex[3]{%
  \iffx@mode@multiuser%
    \index{***@\fixmeindexname:%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !{\color{fx#1}\@nameuse{thefx#1count}: #3: #2}}%
    \index{***#3@\fixmeindexname{} (#3):%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !{\color{fx#1}\@nameuse{thefx#1count}: #2}}%
  \else%
    \index{***@\fixmeindexname:%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !{\color{fx#1}\@nameuse{thefx#1count}: #2}}%
  \fi}

\renewcommand*\FXLayoutContentsLine[3]{%
  \@fxdocolon{#3}%
  \iffx@mode@multiuser%
    \fxaddcontentsline{\color{fx#1}\ignorespaces#3\@fxcolon#2}%
  \else%
    \fxaddcontentsline{\color{fx#1}#2}%
  \fi}
⟨/fxthemecolor⟩
```

## B.3 The colorsig theme

colorsig

```
⟨*fxthemecolorsig⟩
\NeedsTeXFormat{LaTeX2e}
\FXProvidesTheme{colorsig}

\RequirePackage{color}

\FXRequireEnvLayout{colorsig}
\FXRequireTargetLayout{color}

\fxsetface{inline}{}

\renewcommand*\FXLayoutFootnote[3]{\footnote{\color{fx#1}#2\@fxsignature{#3}}}
\renewcommand*\FXLayoutMargin[3]{%
```

```
  \marginpar[{\raggedleft\@fxuseface{margin}\color{fx#1}#2\@fxsignature{#3}}]{%
    \raggedright\@fxuseface{margin}\color{fx#1}#2\@fxsignature{#3}}}
\renewcommand*\FXLayoutMarginClue[3]{%
  \marginpar[{\raggedleft\@fxuseface{margin}\color{fx#1}!\@fxsignature{#3}}]{%
    \raggedright\@fxuseface{margin}\color{fx#1}!\@fxsignature{#3}}}
\renewcommand*\FXLayoutInline[3]{%
  { \textcolor{fx#1}{\@fxuseface{inline}#2\@fxsignature{#3}}}}
\renewcommand*\FXLayoutIndex[3]{%
  \iffx@mode@multiuser%
    \index{***@\fixmeindexname:%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !{\color{fx#1}\@nameuse{thefx#1count}: #2\@fxsignature{#3}}}%
    \index{***#3@\fixmeindexname{} (#3):%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !{\color{fx#1}\@nameuse{thefx#1count}: #2}}%
  \else%
    \index{***@\fixmeindexname:%
      !\@nameuse{@fx#1key}@\fxnotesname{#1}:%
      !{\color{fx#1}\@nameuse{thefx#1count}: #2}}%
  \fi}
\renewcommand*\FXLayoutContentsLine[3]{%
  \iffx@mode@multiuser%
    \fxaddcontentsline{\color{fx#1}#2\@fxsignature{#3}}%
  \else%
    \fxaddcontentsline{\color{fx#1}#2}%
  \fi}
⟨/fxthemecolorsig⟩
```

# Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

**Symbols**

**G**



**I**



**L**

**S**



**T**