Filtering messages with `silence`
v1.1

Paul Isambert
zappathustra@free.fr

2009/02/20

'Errare humanum est, perseverare diabolicum'
Proverb (attributed to Seneca)
'Marginpar on page 3 moved.'
LaTeX

Abstract

This package is designed to filter out unwanted warnings and error messages. Entire packages (including LaTeX) can be censored, but very specific messages can be targeted too. TeX's messages are left untouched.

Contents

1 Introduction

When working with LaTeX, messages are utterly important.

Ok, this was the well-behaved LaTeX user's offering to the gods. Now, the fact is, when processing long documents, repeated messages can really get boring, especially when there's nothing you can do about them. For instance, everybody knows the following:

Marginpar on page x *moved.*

When you encounter that message, it means that you have to keep in mind for the next compilations that 'there is a warning that I can ignore,' which sounds pretty paradoxical. When your document is 5 pages long, it doesn't matter, but when you're typesetting a book-length text, it's different, because you can have tens of such messages. And if you're hacking around, error messages might also appear that you don't want to consider. This package is meant to filter out those unwanted messages.

You probably know all there is to know about warnings and errors, in which case you may want to skip this paragraph. If you don't, here's some information. When LaTeX processes your document, it produces error messages and warnings, and packages and the class you're using do too. Errors generally refer to wrong operations such that the desired result won't be attained, while warnings usually inform you that there might be unexpected but benign modifications. Those messages are sent to the `log` file along with other bits of information. You're probably using a LaTeX-oriented text editor to create your document and at the end of a compilation, it certainly gives you the number of errors and warnings encountered. Hopefully, your editor also has a facility to browse errors and warnings. With `silence`, unwanted messages won't appear anymore.

Those messages always have one of the following structures:

`Package` ⟨*Name of the package*⟩ `Error:` ⟨*message text*⟩

or

`Class` ⟨*Name of the class*⟩ `Error:` ⟨*message text*⟩

or

`LaTeX Error:` ⟨*message text*⟩

Replace *Error* with *Warning* and you have warnings. The important part for `silence` is ⟨*Name of the package*⟩ or ⟨*Name of the class*⟩, because the information is needed when filtering messages. In case of a LaTeX message, ⟨*Name of the package*⟩ is supposed to be `latex`. For filters, ⟨*message text*⟩ will be crucial too.

If a message doesn't begin as above, but displays its text immediately, then it has been sent by TeX; `silence` cannot filter it out, and anyway it would probably be a very bad idea. For instance, there's nothing you can do about something like:

```
Missing number, treated as zero.
```

and the only thing you want to do is correct your mistake.

Another very bad idea is to forget the following: *`silence` does not fix anything.*

2 Usage

2.1 Calling the package

The package may be called as any other package with:

`\usepackage[`⟨*options*⟩`]{silence}`

and as any other package it will be effective only after it has been called, which means that messages issued before will not be filtered. So put it at the very beginning of your preamble. However, if you also want to catch a message issued by the class when it is specified with `\documentclass`, you can't say

```
\usepackage[⟨options⟩]{silence}
⟨filters⟩
\documentclass{article}
```

but instead

```
\RequirePackage[⟨options⟩]{silence}
⟨filters⟩
\documentclass{article}
```

because `\usepackage` is not allowed before `\documentclass`. This way, you can filter for instance a warning issued by the `memoir` class according to which you don't have `ifxetex`, although you have it...

2.2 The brutal way

`\WarningsOff[⟨list⟩]`
`\WarningsOff*`
`\ErrorsOff[⟨list⟩]`
`\ErrorsOff*`

There is one very simple and very effective way to get rid of messages: `\WarningsOff` and `\ErrorsOff`. In their starred versions, no warning or error will make it to your log file. Without a star, and without argument either, they filter out all messages except LaTeX's. Finally, you can specify a list of comma-separated packages whose messages you don't want. For that purpose, LaTeX is considered a package whose name is `latex`. For instance `\WarningsOff[superpack,packex]` will remove all warnings issued by the `superpack` and `packex` packages. Messages issued by classes are also accessible in the same way, so to avoid the warning mentioned above, you can say:

```
\RequirePackage[⟨options⟩]{silence}
\WarningsOff[memoir]
\documentclass{memoir}
```

If you want to filter out LaTeX's warnings only, you can say `\WarningsOff[latex]`. Note that issuing `\WarningsOff[mypack]` after `\WarningsOff`, for instance, is useless, because all warnings are already filtered out. But this won't be mentioned to you. And `silence` won't check either whether a censored package exists or not. So if you say `\WarningsOff[mypak]` and mean `\WarningsOff[mypack]`, this will go unnoticed.

`\WarningsOn[⟨list⟩]`
`\ErrorsOn[⟨list⟩]`

These commands allow messages to be output again. If there is no ⟨*list*⟩, all packages are affected, otherwise only specified packages are affected. So, for instance

```
\WarningsOff
Blah blah blah...
\WarningsOn[superpack,packex]
Blah blah blah...
```

will have all warnings turned off, except those issued by LaTeX (because `\WarningsOff` has no star) and by `superpack` and `packex` in the second part of the document.

Note that the command described in this section are independant of the filters described in the following one, and vice-versa. That is, `\WarningsOn` has no effect on the filters, and if you design a filter to avoid a specific message issued by, say, `superpack`, `\WarningsOn[superpack]` will not deactivate that filter and that one message will still be filtered out.

2.3 The exquisite way

Turning off all messages might be enough in many cases, especially if you turn them on again rapidly, but it is a blunt approach and it can lead to unwanted results. So `silence` allows you to create filters to target specific messages.

`\WarningFilter[⟨family⟩]{⟨package⟩}{⟨message⟩}`
`\ErrorFilter[⟨family⟩]{⟨package⟩}{⟨message⟩}`

These commands will filter out messages issued by ⟨*package*⟩ and beginning with ⟨*message*⟩. The optional ⟨*family*⟩ specification is used to create groups of filters that will be (de)activated together. If there is no family, the filter will be immediately active and you won't be able to turn if off. So, for instance:

```
\WarningFilter{latex}{Marginpar on page}
```

will filter out all LaTeX warnings beginning with '`Marginpar on page`'. On the other hand,

```
\WarningFilter[myfam]{latex}{Marginpar on page}
```

will filter out those same warnings if and only if `myfam` filters are active (but see the `immediate` package option below).

You can be quite specific with the text of the message. For instance,

```
\WarningFilter{latex}{Marginpar on page 2 moved}
```

will filter out marginpar warnings issued on page 2.

In this version (contrary to the starred version below), ⟨*message*⟩ should reproduce the (beginning of) the displayed message. For instance, suppose that you have the following error message:

```
Package superpack Error: The command \foo should not be used here.
```

To filter it out, you can simply say:

```
\ErrorFilter[⟨family⟩]{superpack}{The command \foo should not}
```

although you might know that `superpack` didn't produce it so easily, but instead must have said something like:

```
\PackageError{superpack}{The command \string\foo\space should not be used here}{}
```

Here, you don't have to know how the message was produced, but simply what it looks like. The starred versions of those commands below work differently.

`\ActivateWarningFilters[⟨list⟩]`
`\ActivateErrorFilters[⟨list⟩]`

These macros activate the filters which belong to the families specified in ⟨*list*⟩. If there is no such list, all filters are activated. Indeed, unless the `immediate` option is turned on (see below), filters are not active when created, except those that don't belong to a family. Note that ⟨*list*⟩ contains the name of the family specified in `\WarningFilter[`⟨*family*⟩`]{`⟨*package*⟩`}{`⟨*message*⟩`}` (and similarly for `\ErrorFilter`) and not the name of the package (i.e. ⟨*package*⟩), although you can freely use the same name for both. So for instance:

```
\WarningFilter[myfam]{packex}{You can't do that}
\ActivateWarningFilters[packex]
```

will not activate the desired filter, but instead will try to activate the filters belonging to the `packex` family. If this family doesn't exist, you won't be warned.[1] So the proper way instead is:

```
\WarningFilter[myfam]{packex}{You can't do that}
\ActivateWarningFilters[myfam]
```

Finally, if a filter is created while its family is active, it will be active itself from its creation.

`\DeactivateWarningFilters[⟨list⟩]`
`\DeactivateErrorFilters[⟨list⟩]`

These are self-explanatory. Once again, if there is no list, all filters are deactivated. Note that the `immediate` option implicitly turns on families associated with the filters that are created, so:

```
\WarningFilter[myfam]{packex}{You can't do that}
\DeactivateWarningFilters[myfam]
\WarningFilter[myfam]{superpack}{This is very bad}
```

will make all filters belonging to `myfam` active if `immediate` is on. In this example, both filters will be active, although one might have intended only the second one to be.

`\ActivateFilters[⟨list⟩]`
`\DeactivateFilters[⟨list⟩]`

I bet you know what I'm going to say. These two macros activate or deactivate all filters, or all filters belonging to the specified families in case there's an argument. So you can create error filters and warning filters with the same family name and control them all at once. Isn't it amazing? Note that, just like above, `\ActivateFilters[myfam]` won't complain if there's no `myfam` family, or if there's just a warning family and no error family, and so on and so forth.

[1] You won't be warned either that the `packex` family will really be active, which means that if you create (warning) filters with that family name afterwards, they will take effect immediately, even if you're not in `immediate` mode.

`\WarningFilter*[`*family*`]{`*name*`}{`*message*`}`
`\ErrorFilter*[`*family*`]{`*name*`}{`*message*`}`

These are the same as the starless versions above, except that they target the message not as it appears in the log file but as it was produced. For instance, suppose you have an undefined citation, that is you wrote `\cite{Spinoza1677}` and LaTeX complains as follows:

```
LaTeX Warning: Citation 'Spinoza1677' on page 4 undefined on input line 320.
```

You know you have to bring your bibliography up to date, but right now you're working on a terrific idea and you don't have time to waste. So you say:

```
\WarningFilter{latex}{Citation 'Spinoza1677'}
```

and everything's ok (since you've turned on the `save` option—see below—you will not forget to fix that reference). So you go on but then, as you're trying to link string theory with german philology, you stumble on that paper with bold new ideas and tens of fascinating references that you can't read but definitely have to cite. As expected, LaTeX will start whining, because it doesn't like undefined citations. How are you going to shut it up? You might try

```
\WarningFilter{latex}{Citation}
```

but that's dangerous because all warnings beginning with `Citation` will be filtered out, and maybe there are other types of messages which begin with `Citation` and that you don't want to avoid. So the solution is `\WarningFilter*`. Indeed, you can say:

```
\WarningFilter*{latex}{Citation `\@citeb' on page \thepage \space undefined}
```

That is, you target the way the message was produced instead of the way it appears. Of course, you have to know how the message was produced, but that's easy to figure out. In case of a LaTeX message, just check the source (available on the web). In case the message was issued by a package or a class, just give a look at the corresponding files.

As a rule of thumb, remember that a command that appears verbatim in the message was probably prefixed with `\protect`, `\noexpand` or `\string`, and you can try them all. Commands that are expanded are likely to be followed by `\space`, to avoid unwanted gobbling. So if a message says '`You can't use \foo here`', it is likely to be produced with '`You can't use \protect\foo here`'.

Here's a comparison of starred and starless filters:

`\WarningFilter{latex}{Marginpar on page 3 moved}` will filter off marginpar warnings concerning page 3.

`\WarningFilter{latex}{Marginpar on page \thepage\space moved}` will be inefficient because it will search messages that actually look like the specified text.

`\WarningFilter*{latex}{Marginpar on page \thepage\space moved}` will filter off all marginpar warnings.

`\WarningFilter*{latex}{Marginpar on page 3 moved}` will miss everything because when the the warning is produced the page number is not specified.

`\SafeMode`
`\BoldMode`

As you might have guessed, evaluating messages as they appear means expanding them. Normally, this should be harmless, but one never knows. So these two commands allow you to turn that process on and off. When you say `\SafeMode`, messages are not expanded. In that case, starless filters might miss their goal if the message contains expanded material. Starred filters are unaffected. So if you encounter an avalanche of unexplained error messages, just try some `\SafeMode`. `\BoldMode` is used to switch back to the default mode.

This expansion process concerns messages, not filters. That is, there is no need to protect your filter definitions with `\SafeMode`. Instead, use this command when you suspect that a message is being sent and `silence` gets everything wrong. Use `\BoldMode` to switch back to normal when messages are not troublesome anymore. Here's an example:

```
\WarningFilter{latex}{Marginpar on page 3 moved}
\WarningFilter*{latex}{Citation `\@citeb' undefined}
...
```

```
\SafeMode
...
Here a strange message is being sent by the strangex package.
If \SafeMode was not active, we would be in big trouble.
The Marginpar warnings all go through, because our starless filter
is too specific.  Fortunately, the Citation warnings are correctly
avoided.
...
\BoldMode
...
Everything is back to normal.
```

3 Package options

Here are the options that may be loaded with the package:

debrief At the end of the document, `silence` will output a warning saying how many messages were issued during the compilation and how many were filtered out. This warning will not appear if no message was output or if none were filtered out.

immediate Makes filters active as soon as they are created. This does not affect filters created without a family, since they always behave so. If a filter is created with family `myfam`, and if that family has been previously deactivated, it will be reactivated.

safe Makes safe mode the default mode. `\BoldMode` and `\SafeMode` are still available.

save Messages that were filtered out are written to an external file called `jobname.sil`.

saveall All messages, including those that were left untouched, are written to `jobname.sil`.

showwarnings Warnings are left untouched. That is, `silence`'s commands become harmless. Note that warnings are not written to `jobname.sil`, even if the `saveall` option is loaded. This command is useful if you want to recompile your document as usual.

showerrors Same as `showwarnings` for errors.

4 It doesn't work!

Messages can be tricky. This package was originally designed to take care of marginpar warnings, and I wanted to do something like:

```
\WarningsOff*
\marginpar{A marginpar that will move}
\WarningsOn
```

Unfortunately, this doesn't work. Indeed, marginpar warnings are not issued when the `\marginpar` command is used but at the output routine, that is when LaTeX builds the page, which happens at some interesting breakpoint that you're not supposed to know. That's the reason why those messages must be filtered out with warnings that are always active. Of course, this means that you can't filter out just one particular marginpar warning, unless it's the only one on its page.

Now, messages aren't always what they seem to be. More precisely, two attributes do not really belong to them: the final full stop and the line number (only for some warnings). For instance, the following message *does not* containt a full stop:

```
Package packex Error: You can't do that.
```

The full stop is added by LaTeX. So

```
\ErrorFilter{packex}{You can't do that.}
```

won't do. You have to remove the stop. This goes the same with the phrase '`on input line` ⟨*number*⟩.' (including the stop once again). That is, the message

```
Package superpack Warning: Something is wrong on input line 352.
```

was actually produced with

```
\PackageWarning{superpack}{Something is wrong}
```

The end of it was added by LaTeX. You know what you have to do. Unfortunately, this means that warnings can't be filtered according to the line they refer to.

Another difficulty concerns line breaking in messages. If a new line begins in a message, it was either explicitely created or it's a single line wrapped by your text editor. When properly written, messages use LaTeX's `\MessageBreak` command to create new lines, which LaTeX formats as a nicely indented line with the name of the package at the beginning, between parentheses. So if you encounter such a display, you know that there's something more behind the scene. You have two solutions: either you make the text of your filter shorter than the first line, which in most cases will be accurate enough, or you use a starred filter and explicitely write `\MessageBreak`. Unfortunately, you can't use `\MessageBreak` in a starless filter. Note that some stupid people (including the author of this package) sometimes use `^^J` instead of `\MessageBreak`, which is a TeX construct to create a new line. In that case, the line break will be indistinguishable from a single line wrapped by your text editor (although no wrapping occur in the log file).

The most efficient filters are the starred ones (unless you're aiming at a specific value for a variable) whose text has simply been pasted from the source. E.g., if `superpack` tells you:

```
Package superpack Error: You can't use \@terrific@command in
(superpack)              a \@fantastic@environment, because
(superpack)              unbelievable@parameter is off.
```

this was probably produced with:

```
\PackageError{superpack}{
  You can't use \protect\@terrific@command in\MessageBreak
  a \protect\@fantastic@environment, because\MessageBreak
  \superparameter\space is off}
  {}
```

with `\superparameter` `\def`ined to `unbelievable@parameter` beforehand. So the simplest way to filter out such a message is to open `superpack.sty`, look for it, and copy it as is in a starred filter.

There remains one problematic case, if a primitive control sequence appears in the message to be avoided. Imagine for instance that a package sends the warning '`You can't use \def here`'. It will not be reachable with a starless filter, because the package may have said `\def` without any prefix, since primitive commands can be used as such in messages, where they appear verbatim. On the other hand, when you create a starless filter with a command in it, `silence` considers this command simply as a string of characters beginning with a backslash devoid of its usual 'escapeness'—as are control sequences prefixed with `\protect`, `\string` and `\noexpand` in messages. So `\def` won't be reached. A starred filter might do, but in this case you shouldn't prefix `\def` with any of the three commands just mentioned. Now if `\def` is the result of an expansion, you'll be forced to rely on the previous techniques. Fortunately, this is very rare.

Now, in case there's a message you really can't reach, although you pasted it in your filter, just let me know: it's an opportunity to refine `silence`.

5 Implementation

5.1 Basic definitions

The options only turn on some conditionals, except `save` and `saveall`, which set the count `\sl@save`, to be used in a `\ifcase` statement.

```
\makeatletter

\newcount\sl@Save
\newif\ifsl@Debrief
\newif\ifsl@ShowWarnings
\newif\ifsl@ShowErrors
\newif\ifsl@Immediate
\newif\ifsl@SafeMode

\DeclareOption{debrief}{\sl@Debrieftrue}
\DeclareOption{immediate}{\sl@Immediatetrue}
\DeclareOption{safe}{\sl@SafeModetrue}
\DeclareOption{save}{\sl@Save1%
\newwrite\sl@Write%
\immediate\openout\sl@Write=\jobname.sil}
\DeclareOption{saveall}{\sl@Save2%
\newwrite\sl@Write%
\immediate\openout\sl@Write=\jobname.sil}
\DeclareOption{showwarnings}{\sl@ShowWarningstrue}
\DeclareOption{showerrors}{\sl@ShowErrorstrue}
\ProcessOptions\relax
```

Here are the counts, token lists and conditionals, that will be used by warnings and errors.

```
\newcount\sl@StateNumber
\newcount\sl@MessageCount
\newcount\sl@Casualties

\newtoks\sl@Filter
\newtoks\sl@Message
\newtoks\sl@UnexpandedMessage
\newtoks\sl@Mess@ge

\newif\ifsl@Check
\newif\ifsl@Belong
\newif\ifsl@KillMessage
\newif\ifsl@SafeTest
```

And here are some keywords and further definitions. `\sl@PackageName` is used to identify the name of the package, but in case `\GenericError` or `\GenericWarning` were directly used, it would be undefined (or defined with the name of the last package that issued a message), which would lead to some trouble, hence its definition here.

```
\def\sl@end{sl@end}
\def\sl@latex{latex}
\def\sl@Terminator{č}
\gdef\sl@active{active}
\gdef\sl@safe{safe}
\gdef\sl@PackageName{NoPackage}
\def\SafeMode{\global\sl@SafeModetrue}
\def\BoldMode{\global\sl@SafeModefalse}
\def\sl@Gobble#1sl@end,{}
```

5.2 Warnings

Now these are the counts, token lists, conditionals and keywords specific to warnings. `sl@family` is actually the family of those familyless filters. It is made active as wanted.

```
\newcount\sl@WarningCount
\newcount\sl@WarningNumber
\newcount\sl@WarningCasualties

\newtoks\sl@TempBOW
\newtoks\sl@BankOfWarnings

\newif\ifsl@WarningsOff
\newif\ifsl@NoLine

\expandafter\gdef\csname sl@family:WarningState\endcsname{active}
\def\sl@WarningNames{}
\def\sl@UnwantedWarnings{}
\def\sl@ProtectedWarnings{}
```

5.2.1 Brutal commands

`\WarningsOn` The basic mechanism behind `\WarningsOn` and `\WarningsOff` is a conditional, namely `\ifsl@WarningsOff`. When a warning is sent, `silence` checks the value of this conditional and acts accordingly: if it is set to `true`, then the warning is filtered out unless it belongs to the `\sl@ProtectedWarnings` list; if it is set to `false`, the warning is output unless it belongs to the `\sl@UnwantedWarnings` list.

Without argument, `\WarningsOn` empties both lists and sets this conditional to `false`. (Emptying `\sl@ProtectedWarnings` is useless but it keeps the list clean.) If it has an argument, its behavior depends on the conditional; if it is set to `true`, the argument is added to `\sl@ProtectedWarnings`; otherwise, it is removed from `\sl@UnwantedWarnings`.

```
\def\WarningsOn{%
  \@ifnextchar[%
  {\ifsl@WarningsOff%
      \def\sl@next{\sl@Add\sl@ProtectedWarnings}%
    \else%
      \def\sl@next{\sl@Remove\sl@UnwantedWarnings}%
    \fi\sl@next}%
  {\global\sl@WarningsOfffalse%
  \gdef\sl@ProtectedWarnings{}%
  \gdef\sl@UnwantedWarnings{}}}
```

`\WarningsOff` `\WarningsOff` does the same as `\WarningsOn`, but in the other way. If it has no star and no argument, `\sl@ProtectedWarnings` is overwritten with only `latex` in it.

```
\def\WarningsOff{%
  \@ifstar%
  {\global\sl@WarningsOfftrue
    \gdef\sl@UnwantedWarnings{}%
    \gdef\sl@ProtectedWarnings{}}%
  {\@ifnextchar[{%
    \ifsl@WarningsOff%
      \def\sl@next{\sl@Remove\sl@ProtectedWarnings}%
    \else%
      \def\sl@next{\sl@Add\sl@UnwantedWarnings}%
    \fi\sl@next}%
    {\global\sl@WarningsOfftrue%
    \gdef\sl@UnwantedWarnings{}%
    \gdef\sl@ProtectedWarnings{latex,}}}}
```

Note that the \WarningsOn and \WarningsOff don't really take any argument. If an opening bracket is present, they launch \sl@Add or \sl@Remove on the adequate list.

\sl@Add

\sl@Add is no more than an \xdef of the list on itself, plus the new item.

```
\def\sl@Add#1[#2]{%
  \xdef#1{#1#2,}}
```

\sl@Remove

\sl@Remove is slightly more complicated. It stores the items to be removed and then launches the recursive \sl@@Remove on the expanded list, with a terminator to stop it. When \sl@@Remove has done its job, the list will be \let to the new one.

```
\def\sl@Remove#1[#2]{%
  \def\sl@Items{#2}
  \def\sl@TempNewList{}%
  \expandafter\sl@@Remove#1sl@end,%
  \let#1\sl@TempNewList}
```

\sl@@Remove

This macro takes each element of the list to be updated, checks it against the items to be removed, and builds a new list containing the element currently tested if and only if it has not been matched with items to be removed.

First, we check the current element of the list, and if it's the terminator we end recursion.

```
\def\sl@@Remove#1,{%
  \def\sl@Tempa{#1}%
  \ifx\sl@Tempa\sl@end%
    \let\sl@next\relax%
```

Otherwise, we launch \sl@ListCheck on the element.

```
  \else%
    \sl@Checkfalse%
    \expandafter\sl@ListCheck\sl@Items,sl@end,%
```

If the check is positive, we do nothing. If not, we add the element to \sl@TempNewList, which is itself repeated to keep the former elements.

```
    \ifsl@Check%
    \else%
      \xdef\sl@TempNewList{\sl@TempNewList#1,}%
    \fi%
    \let\sl@next\sl@@Remove%
  \fi\sl@next}%
```

\sl@ListCheck

Here's the internal checking mechanism. It takes an argument (from the expansion of \sl@Items above) and compares it to the element under scrutiny. In case they match, the proper conditional is turned to true, and we launch \sl@Gobble to discard remaining items. Otherwise, we proceed to the next item.

First we check for the terminator, as usual, and the recursion is ended in case we find it.

```
\def\sl@ListCheck#1,{%
  \def\sl@Tempb{#1}%
  \ifx\sl@Tempb\sl@end%
    \let\sl@next\relax%
```

If the item is not the terminator, we compare it to the current element. If they match, we confirm the test and gobble.

```
  \else%
    \ifx\sl@Tempa\sl@Tempb%
      \sl@Checktrue
      \let\sl@next\sl@Gobble%
```

Otherwise we repeat.

```
    \else%
      \let\sl@next\sl@ListCheck%
    \fi%
  \fi\sl@next}
```

5.2.2 Filters

\WarningFilter

Let's now turn to filters. When created, each warning filter is associated with a `\sl@WarningNumber` to retrieve it and to specify its status. In case of `\WarningFilter*`, this status, referred to with `\csname\the\sl@WarningNumber:WarningMode\endcsname` (i.e. `\`⟨*number*⟩`:WarningMode`, where ⟨*number*⟩ is the unique number associated with the filter), is set to "`safe`". Otherwise, we don't bother to define it, because when evaluated the above command will then be equal to `\relax` (as are all undefined commands called with `\csname... \endcsname`). This will be checked in due time to know whether the expanded or the unexpanded version of the target message must be tested.

```
\def\WarningFilter{%
  \global\advance\sl@WarningNumber1%
  \@ifstar%
    {\expandafter\gdef\csname\the\sl@WarningNumber:WarningMode\endcsname{safe}%
    \sl@WarningFilter}%
    {\sl@WarningFilter}}
```

\sl@WarningFilter

Once the star has been checked, we look for an optional argument. In case there is none, we assign the `sl@family` to the filter.

```
\def\sl@WarningFilter{%
  \@ifnextchar[%
  {\sl@@WarningFilter}%
  {\sl@@WarningFilter[sl@family]}}
```

\sl@@WarningFilter

Here comes the big part. First, we update the list of filters associated with a package, stored in `\`⟨*package*⟩`@WarningFilter`. The list itself is a concatenation of comma-separated pairs of the form ⟨*number*⟩`:sl@`⟨*family*⟩ referring to filters. When ⟨*package*⟩ issues a warning, `silence` checks the filters contained in the associated list. ⟨*number*⟩ is used to determine the `WarningMode` (`safe` or undefined, as noted above), so that the right test will be run, while ⟨*family*⟩ is used to ensure that this family is active.

First we update the list:

```
\def\sl@@WarningFilter[#1]#2{%
  \expandafter\ifx\csname #2@WarningFilter\endcsname\relax%
    \expandafter\xdef\csname #2@WarningFilter\endcsname{\the\sl@WarningNumber:sl@#1}%
  \else%
    \expandafter\xdef\csname #2@WarningFilter\endcsname{%
      \csname #2@WarningFilter\endcsname,\the\sl@WarningNumber:sl@#1}%
  \fi%
```

Now, if `\`⟨*family*⟩`:WarningState` is undefined, this means that this is the first time we encounter that family. So we store its name, which will be useful for `\ActivateWarningFilters` and its deactivating counterpart.

```
  \expandafter\ifx\csname #1:WarningState\endcsname\relax%
    \sl@Add\sl@WarningNames[#1]%
  \fi%
```

Next, we check `WarningState` again; if it's not active, we change it, depending on whether we're in `immediate` mode or not.

```
  \expandafter\ifx\csname #1:WarningState\endcsname\sl@active%
```

```
\else%
  \ifsl@Immediate%
    \expandafter\gdef\csname #1:WarningState\endcsname{active}%
  \else%
    \expandafter\gdef\csname #1:WarningState\endcsname{inactive}%
  \fi%
\fi%
```

Finally, we prepare the storage of the filter's message. We open a group, and if the filter is starred, we turn `@` into a letter, so that it will be able to form macro names. If the filter has no star, we turn the escape character into a normal one, so that control sequences won't be formed. Messages will then be tested token by token, and if a message contains e.g. `\foo`, then it has (most likely) been `\string`'ed, so it's really a sequence of characters with `\catcode` 12 (including the escape character) that the starless filter will match.

```
\begingroup%
\expandafter\ifx\csname\the\sl@WarningNumber:WarningMode\endcsname\sl@safe%
  \makeatletter%
\else%
  \catcode`\\12\relax%
\fi%
\sl@AddToBankOfWarnings}
```

`\sl@AddToBankOfWarnings`

Now we add the filter to the `\sl@BankOfWarnings` token list, delimited by `(:sl@`⟨*number*⟩`:)`, where ⟨*number*⟩ is the number assigned to it above. Thus, it might be easily retrieved. Following a technique explained by Victor Eijkhout in *TEX by Topic*, we make use of `\edef` to add an item to a token list. First, we store it in the temporary `\sl@TempBOW` token list. Then we `\edef \sl@act` such that its definition will be:

`\sl@BankOfWarnings{`⟨*previous content*⟩`(:sl@`⟨*number*⟩`:)`⟨*filter*⟩`(:sl@`⟨*number*⟩`:)}`

Thus, the `\sl@BankOfWarnings` token list will add the new filter to itself. This works because token lists prefixed with `\the` in an `\edef` expand to their content unexpanded, unlike normal macros. The macro is made `\long` just in case.

```
\long\def\sl@AddToBankOfWarnings#1{%
  \sl@TempBOW{#1}%
  \edef\sl@act{%
  \global\noexpand\sl@BankOfWarnings{%
    \the\sl@BankOfWarnings%
    (:sl@\the\sl@WarningNumber:)\the\sl@TempBOW(:sl@\the\sl@WarningNumber:)}}%
  \sl@act%
  \endgroup}
```

`\ActivateWarningFilters`
`\DeactivateWarningFilters`

This macro launches a recursive redefinition of its expanded argument, which is either a list defined by the user or the list containing all warning families (updated in `\WarningFilter` above). We set `\sl@StateNumber` to register what the redefinition should be: activate or deactivate `WarningState`, or activate or deactivate `ErrorState`.

```
\def\ActivateWarningFilters{%
  \sl@StateNumber0\relax%
  \@ifnextchar[%
  {\sl@ChangeState}%
  {\sl@ChangeState[\sl@WarningNames]}}

\def\DeactivateWarningFilters{%
  \sl@StateNumber1\relax%
  \@ifnextchar[%
  {\sl@ChangeState}%
  {\sl@ChangeState[\sl@WarningNames]}}
```

`\sl@ChangeState`
`\sl@@ChangeState`

`\sl@ChangeState` only calls `\sl@@ChangeState` on the expanded argument, adding a ter-

minator. `\sl@@ChangeState` checks whether its argument is the terminator, in which case it stops, or else it sets its state to the appropriate value, depending on `\sl@StateNumber`

```
\def\sl@ChangeState[#1]{%
  \expandafter\sl@@ChangeState#1,sl@end,}

\def\sl@@ChangeState#1,{%
  \def\sl@Tempa{#1}%
  \ifx\sl@Tempa\sl@end%
    \let\sl@next\relax%
  \else%
    \ifcase\sl@StateNumber%
      \expandafter\gdef\csname #1:WarningState\endcsname{active}%
    \or%
      \expandafter\gdef\csname #1:WarningState\endcsname{inactive}%
    \or%
      \expandafter\gdef\csname #1:ErrorState\endcsname{active}%
    \or%
      \expandafter\gdef\csname #1:ErrorState\endcsname{inactive}%
    \fi%
    \let\sl@next\sl@@ChangeState%
  \fi\sl@next}
```

`\ActivateFilters` `\DeactivateFilters` `\sl@RetrieveArgument`

This aren't just shorthands, something more is needed in case there's an argument (because we have to retrieve it to pass it to two macros). However, it's rather straightforward, we're just using `\sl@StateNumber` to keep track of what is needed.

Here we just rely on the value required for `WarningState`, i.e. 0 or 1, and the value for `ErrorState` follows suit.

```
\def\ActivateFilters{%
  \@ifnextchar[%
  {\sl@StateNumber0%
  \sl@RetrieveArgument}%
  {\sl@StateNumber0\relax%
  \sl@ChangeState[\sl@WarningNames]%
  \sl@StateNumber2\relax%
  \sl@ChangeState[\sl@ErrorNames]}}

\def\DeactivateFilters{%
  \@ifnextchar[%
  {\sl@StateNumber1%
  \sl@RetrieveArgument}%
  {\sl@StateNumber1\relax%
  \sl@ChangeState[\sl@WarningNames]%
  \sl@StateNumber3\relax%
  \sl@ChangeState[\sl@ErrorNames]}}

\def\sl@RetrieveArgument[#1]{%
  \def\sl@Argument{#1}
  \ifcase\sl@StateNumber%
    \sl@ChangeState[\sl@Argument]%
    \sl@StateNumber2\relax%
    \sl@ChangeState[\sl@Argument]%
  \or%
    \sl@ChangeState[\sl@Argument]%
    \sl@StateNumber3\relax%
    \sl@ChangeState[\sl@Argument]%
  \fi}
```

5.2.3 String testing

Now, here comes the crux of the biscuit, as some guitarist from California once said. Here are the macros to test the messages. They will be used in the redefinition of `\GenericWarning`.

`\sl@GetNumber` When packages send a warning, `silence` launches `\sl@GetNumber` on the expanded list of ⟨*number*⟩`:sl@`⟨*family*⟩ pairs associated with this package (created in `\WarningFilter` above). Remember that ⟨*number*⟩ refers to a filter and ⟨*family*⟩ to its family.

If ⟨*number*⟩ is not 0 (which is associated with the terminator added when `\sl@GetNumber` is launched), we test whether the family is active. If so, `\sl@GetMessage` is called; otherwise, we proceed to the next pair.

The command

```
\csname #2:\ifcase\sl@StateNumber Warning\or Error\fi State\endcsname
```

reduces to \⟨*family*⟩`:WarningState` or \⟨*family*⟩`:ErrorState`, depending on the value of `\sl@StateNumber`, which is turned to 0 if we're testing a warning or to 1 if we're testing an error.

```
\def\sl@GetNumber#1:sl@#2,{%
  \ifnum#1>0%
    \expandafter%
    \ifx\csname #2:\ifcase\sl@StateNumber Warning\or Error\fi State\endcsname\sl@active%
      \sl@GetMessage{#1}%
    \else%
      \let\sl@next\sl@GetNumber%
    \fi%
  \else%
    \let\sl@next\relax%
  \fi\sl@next}
```

`\sl@GetMessage` Now we're going to retrieve the filter from the `\sl@BankOfWarnings` token list. This list has the following structure, as you might remember:

`(:sl@1:)`⟨*first filter*⟩`(:sl@1:)(:sl@2:)`⟨*second filter*⟩`(:sl@2:)...(:sl@n:)`⟨n*th filter*⟩`(:sl@n:)`

To do so, `\sl@GetMessage` defines a new macro on the fly, `\sl@@GetMessage`, which takes three arguments, delimited by `(:sl@`⟨*number*⟩`:)`, where ⟨*numbers*⟩ depends on the pair we're in. That is, suppose `\sl@GetNumber` is examining the pair `15:sl@myfam` and `myfam` is active. Then we feed `15` to `\sl@GetMessage`, which in turn creates `\sl@@GetMessage` with the arguments delimited by `(:sl@15:)`. The first and the third arguments are discarded, while the second one, which is the message of the filter, is stored in the `\sl@Filter` token list, along with a terminator.[2]

```
\def\sl@GetMessage#1{%
  \def\sl@@GetMessage##1(:sl@#1:)##2(:sl@#1:)##3(:sl@end:){\sl@Filter={##2č}}%
```

Now we launch this command on the bank of warnings (or errors) expanded one step, thanks to the same `\edef` trick as above.

```
  \ifcase\sl@StateNumber%
    \edef\sl@act{\noexpand\sl@@GetMessage\the\sl@BankOfWarnings(:sl@end:)}%
  \or%
    \edef\sl@act{\noexpand\sl@@GetMessage\the\sl@BankOfErrors(:sl@end:)}%
  \fi%
  \sl@act%
```

When a warning is sent, its message is stored in two forms, expanded and unexpanded. Depending on the mode we're currently in (safe or bold), we retrieve the right form; in safe mode, we take the unexpanded form; in bold mode, we take it too if the `WarningMode` of

[2] This terminator is the £ symbol, unfortunately totally unverbatimizable. So it appears as č.

this particular filter is `safe` (i.e. if it was created with `\WarningFilter*`), otherwise we take the expanded version.

`\sl@Message` is the token list containing the expanded version, `\sl@UnexpandedMessage` contains the unexpanded version, and `\sl@Mess@ge` stores the adequate version for the test to come followed by a terminator.

At the end of this macro, we call the string tester.

```
\ifsl@SafeMode%
  \sl@SafeTesttrue%
  \edef\sl@act{\noexpand\sl@Mess@ge{\the\sl@UnexpandedMessageč}}%
\else%
  \expandafter%
  \ifx\csname #1:\ifcase\sl@StateNumber Warning\or Error\fi Mode\endcsname\sl@safe%
    \sl@SafeTesttrue%
    \edef\sl@act{\noexpand\sl@Mess@ge{\the\sl@UnexpandedMessageč}}%
  \else%
    \sl@SafeTestfalse%
    \edef\sl@act{\noexpand\sl@Mess@ge{\the\sl@Messageč}}%
  \fi%
\fi%
\sl@act%
\sl@TestStrings}
```

`\sl@TestStrings` This test is a recursive token by token comparison of `\sl@Filter` and `\sl@Message`, i.e. it compares two strings.

First we take the first token of each token list thanks, once again, to an `\edef`. They are stored in `\sl@FilterToken` and `\sl@MessageToken`, which are macros, not token lists, by the way (see below the definition of `\sl@Slice`).

```
\def\sl@TestStrings{%
  \edef\sl@act{%
  \noexpand\sl@Slice\the\sl@Filter(:sl@mid:)\noexpand\sl@Filter\noexpand\sl@FilterToken%
  \noexpand\sl@Slice\the\sl@Mess@ge(:sl@mid:)\noexpand\sl@Mess@ge\noexpand\sl@MessageToken}%
  \sl@act%
```

Then we simply run some conditional. If we reach the terminator in the filter, this means that it matches the warning (otherwise we wouldn't have gone so far), so we turn a very sad conditional to `true`, stop the recursion of `\sl@TestString` (thanks to `\sl@@next`) and gobble the remaining ⟨*number*⟩`:sl@`⟨*family*⟩ pairs waiting to be evaluated by `\sl@GetNumber` (thanks to `\sl@next`). Our job is done, and `silence` grins with cruelty.

```
  \ifx\sl@FilterToken\sl@Terminator%
    \sl@KillMessagetrue%
    \let\sl@@next\relax%
    \let\sl@next\sl@Gobble%
```

On the other hand, if we reach the terminator in the warning text, this means that it is shorter than the filter (unless we also reach the terminator in the filter, but this was taken care of in the previous case), so they don't match. We stop recursion (there's nothing left to test) and make `\sl@GetNumber` consider the following pair.

```
  \else
    \ifx\sl@MessageToken\sl@Terminator%
      \let\sl@@next\relax%
      \let\sl@next\sl@GetNumber%
```

Now, if none of the tokens is a terminator, then we have to compare them. The test will depend on the value of `\ifsl@SafeTest` which was turned to `true` in case we're in safe mode or the filter is a starred one. In that case we run an `\ifx` test, so that even control sequences can be properly compared. Since the message is not expanded, this is vital. If

the tokens match, we proceed to the next ones; otherwise, this means that the filter and the message are different, so we stop recursion (there's no reason to test further), and we call `\sl@GetNumber` on the next pair.

```
    \else%
      \ifsl@SafeTest%
        \ifx\sl@FilterToken\sl@MessageToken%
          \let\sl@@next\sl@TestStrings%
        \else%
          \let\sl@@next\relax%
          \let\sl@next\sl@GetNumber%
        \fi%
```

If we're in bold mode and the filter is starless, then we simply compare character codes with `\if`. Thus, the letters of, say, `\foo`, in the filter, will match their counterpart in the warning (where the command has probably been `\string`'ed), although their category codes are different: it's 11 in the filter (no control sequence was ever created: `\` was turned to a normal character before the filter was stored) and 12 in the message (like all `\string`'ed characters).

```
      \else%
        \if\sl@FilterToken\sl@MessageToken%
          \let\sl@@next\sl@TestStrings%
        \else%
          \let\sl@@next\relax%
          \let\sl@next\sl@GetNumber%
        \fi%
      \fi%
    \fi%
  \fi\sl@@next}
```

`\sl@Slice` And here's the final cog in this testing. To put it quite unintelligibly, `\sl@Slice` defines its fourth argument to expand to the first, while the third, which is a token list, is set to the second, and you should not forget that the first two arguments are just an expansion of the third. Copy that?

Let's get things clear. Remember that `\sl@act` at the beginning of `\sl@TestStrings` above was `\edef`ined to:

`\noexpand\sl@Slice\the\sl@Filter(:sl@mid:)\noexpand\sl@Filter\noexpand\sl@FilterToken`

and similarly for the message. This means that its definition text is:

`\sl@Slice`⟨*content of* `\sl@Filter`⟩`(:sl@mid:)\sl@Filter\sl@FilterToken`

The first argument of `\sl@Slice` is undelimited. This means that it will be the first token of ⟨*content of* `\sl@Filter`⟩. Its second argument will be the rest of this content. Now, as explained, `\sl@Slice` defines its fourth argument, namely `\sl@FilterToken`, to expand to its first one, namely the first token in the `\sl@Filter` token list, and sets this token list to the second argument, i.e. the to rest of itself. In short, we're emptying `\sl@Filter` token by token and we compare them along the way, as described above.

```
\def\sl@Slice#1#2(:sl@mid:)#3#4{\def#4{#1}#3={#2}}
```

5.2.4 Redefining warnings

`\sl@Belong` We have two more macros to create before we can redefine warnings themselves. `\sl@Belong` is used to check whether a package belongs to the `\sl@UnwantedWarnings` list or the `\sl@ProtectedWarnings` list (correspondingly for errors). Its argument is an item of those lists expanded, that is, the name of a package. It is compared to `\sl@PackageName`, which is defined to the name of the package sending the message. If they don't match, we relaunch

the command on the next item. If they do, we turn `\ifsl@Belong` to `true` and gobble the following items.

```
\def\sl@Belong#1,{%
  \def\sl@Tempa{#1}
  \ifx\sl@Tempa\sl@end%
    \let\sl@next\relax%
  \else%
    \ifx\sl@Tempa\sl@PackageName%
      \sl@Belongtrue%
      \let\sl@next\sl@Gobble%
    \else%
      \let\sl@next\sl@Belong%
    \fi%
  \fi\sl@next}
```

`\sl@StoreMessage` Finally, we need a mechanism to store the message being sent. In safe mode, we store it unexpanded. In bold mode, we also store it unexpanded for starred filters, but we also store an expanded version where `\protect` and `\noexpand` are `\let` to `\string`, so that the control sequences they prefix will be turned into sequences of characters (remember that no control sequence is formed in the text of a starless filter, only strings of characters). We do this in a group because, well, you know, you shouldn't do that...

```
\def\sl@StoreMessage#1{%
  \ifsl@SafeMode%
    \sl@UnexpandedMessage{#1}%
  \else%
    \sl@UnexpandedMessage{#1}%
    \bgroup%
    \let\protect\string%
    \let\noexpand\string%
    \edef\sl@Tempa{#1}%
    \global\expandafter\sl@Message\expandafter{\sl@Tempa}%
    \egroup%
  \fi}
```

Now we're ready for the big redefinitions. First, if the `showwarnings` option is on, we simply redefine nothing, otherwise we begin by retrieving the current definitions of the commands used to issue warnings, in order to patch them. For instance, we `\let \sl@PackageWarning` to `\PackageWarning`, and thus we'll be able to write:

```
\def\PackageWarning#1#2{%
  ⟨additional code⟩
  \sl@PackageWarning{#1}{#2}}
```

and this will launch `\sl@PackageWarning`, i.e. the original definition of `\PackageWarning`, which will do the job it was meant to do in the first place.

For `\GenericWarning`, this needs some hacking. Indeed `\GenericWarning` is robust, which means that it actually does nothing except calling `\protect\GenericWarning`⟨*space*⟩, where `\GenericWarning`⟨*space*⟩ is defined to issue the warning as wanted. Thus, if we simply `\let \sl@GenericWarning` to `\GenericWarning` and write:

```
\DeclareRobustCommand{\GenericWarning}[2]{%
  ⟨additional code⟩
  \sl@GenericWarning{#1}{#2}}
```

then, because of the robust declaration, `\GenericWarning` will be defined to `\protect \GenericWarning`⟨*space*⟩, whose definition is the additional code followed by `\sl@GenericWarning` which, because it was `\let` to the older version of `\GenericWarning`, expands to `\GenericWarning`⟨*space*⟩—that is, we enter an infinite loop. The solution is to `\let`

`\sl@GenericWarning` directly to `\GenericWarning`⟨*space*⟩.

```
\ifsl@ShowWarnings
\else
\expandafter\let\expandafter\sl@GenericWarning\csname GenericWarning \endcsname
\let\sl@PackageWarning\PackageWarning
\let\sl@ClassWarning\ClassWarning
\let\sl@latex@warning\@latex@warning
```

`\PackageWarning` We redefine `\PackageWarning` so that it stores the name of the package calling it, and the message only if a certain conditional is true, that is if it has not been sent with `\PackageWarningNoLine`

```
\def\PackageWarning#1#2{%
  \def\sl@PackageName{#1}%
  \ifsl@NoLine%
    \sl@NoLinefalse%
  \else%
    \sl@StoreMessage{#2}%
  \fi%
  \sl@PackageWarning{#1}{#2}}
```

`\PackageWarningNoLine` For `\PackageWarningNoLine`, we simply store the message and send it to `\PackageWarning` with an additional `\@gobble` to discard the '`on input line...`' phrase added by LaTeX. (This `\@gobble` was already in the original definition, there is nothing new here.)

```
\def\PackageWarningNoLine#1#2{%
  \sl@StoreMessage{#2}%
  \sl@NoLinetrue%
  \PackageWarning{#1}{#2\@gobble}}
```

`\ClassWarning` `\ClassWarningNoLine` `\@latex@warning` `\@latex@warning@no@line` We do exactly the same for class warnings and LaTeX warnings, except that in the latter case we manually set `\sl@PackageName` to `latex`.

```
\def\ClassWarning#1#2{%
  \def\sl@PackageName{#1}%
  \ifsl@NoLine%
    \sl@NoLinefalse%
  \else%
    \sl@StoreMessage{#2}%
  \fi%
  \sl@ClassWarning{#1}{#2}}

\def\ClassWarningNoLine#1#2{%
  \sl@StoreMessage{#2}%
  \sl@NoLinetrue%
  \ClassWarning{#1}{#2\@gobble}}

\def\@latex@warning#1{%
  \def\sl@PackageName{latex}%
  \ifsl@NoLine%
    \sl@NoLinefalse%
  \else%
    \sl@StoreMessage{#1}%
  \fi%
  \sl@latex@warning{#1}}

\def\@latex@warning@no@line#1{%
  \sl@StoreMessage{#1}%
  \sl@NoLinetrue%
  \@latex@warning{#1\@gobble}}
```

\GenericWarning

Now we can redefine \GenericWarning. Originally, a message sent with, for instance, \PackageWarning, is edited, and sent to \GenericWarning, which edits it further and sends it to the log file. None of this is modified here, although some names have changed. Indeed, \PackageWarning now stores the name of the package and the message but then calls \sl@PackageWarning on them, which has been \let to the previous value of \PackageWarning. So the message is formatted as usual and sent to \GenericWarning which, if the message is not filtered out, will launch \sl@GenericWarning on the same arguments and thus LaTeX's original mechanism will pass unaffected—albeit somewhat spied upon... The original command is robust, so we make it robust too.

First, we increment \sl@WarningCount, which will be used if the debrief option is on. We also restore some conditional to their default values.

```
\DeclareRobustCommand{\GenericWarning}[2]{%
  \advance\sl@WarningCount1\relax%
  \sl@KillMessagefalse%
  \sl@Belongfalse%
```

Next, we launch \sl@Belong on the expanded list of protected warnings if warnings are off, or on the expanded list of unwanted warnings if they're on. Depending on the result (see the comment on \WarningsOn and \WarningsOff above), we might sentence the message to death. (Turn on the save option, please, so it may live a happy afterlife in jobname.sil.)

```
  \ifsl@WarningsOff%
    \expandafter\sl@Belong\sl@ProtectedWarnings sl@end,%
    \ifsl@Belong%
    \else%
      \sl@KillMessagetrue%
    \fi%
  \else%
    \expandafter\sl@Belong\sl@UnwantedWarnings sl@end,%
    \ifsl@Belong%
      \sl@KillMessagetrue%
    \fi%
  \fi%
```

If the preceding operation is not enough, we check whether some filters are associated with the package sending the message (in which case \⟨*package*⟩@WarningFilter is defined and contains ⟨*number*⟩:sl@⟨*family*⟩ pairs). If there are some, we test them with \sl@GetNumber as defined above.

```
  \ifsl@KillMessage%
  \else%
    \expandafter\ifx\csname\sl@PackageName @WarningFilter\endcsname\relax%
    \else%
      \sl@StateNumber0\relax%
      \expandafter\expandafter\expandafter%
      \sl@GetNumber\csname\sl@PackageName @WarningFilter\endcsname,0:sl@sl@end,%
    \fi%
  \fi%
```

Now the message's fate is sealed. If it has been doomed to filtering, we grimly step a sad sad counter. Otherwise, we happily sends everything to \sl@GenericWarning to enlighten the user's log file.

```
  \ifsl@KillMessage%
    \advance\sl@WarningCasualties1%
  \else%
    \sl@GenericWarning{#1}{#2}%
  \fi%
```

Finally, we consider saving it. `\sl@Save` is 0 by default, 1 if the `save` option is on, and 2 if `saveall` is on. We act accordingly: case 0 does nothing, case 1 sends the message to `jobname.sil` if it has been filtered out, and case 2 sends all messages. Instead of writing everything out properly, we simply 're-route' `\sl@GenericWarning` by `\let`ting `\@unused`, LaTeX's output stream, to `\sl@Write`, `silence`'s output stream directed to `jobname.sil`. We do this locally, of course.

```
  \ifcase\sl@Save%
  \or%
    \ifsl@KillMessage%
      \bgroup%
      \let\@unused\sl@Write%
      \sl@GenericWarning{#1}{#2}%
      \egroup%
    \fi%
  \or%
    \bgroup%
    \let\@unused\sl@Write%
    \sl@GenericWarning{#1}{#2}%
    \egroup%
  \fi%
```

Before closing, we set `\sl@PackageName` to `NoPackage`, just in case (very unlikely, to be sure) a package uses `\GenericWarning` directly. Thus, the former value of `\sl@PackageName` won't interfere. The final `\fi` matches `\ifsl@ShowWarnings` some hundred lines above.

```
  \gdef\sl@PackageName{NoPackage}}%
\fi
```

5.3 Errors

Errors are implemented just like warnings, so I don't command the code. See you at the end of the package for the last macro.

```
\newcount\sl@ErrorCount
\newcount\sl@ErrorNumber
\newcount\sl@ErrorCasualties

\newtoks\sl@TempBOE
\newtoks\sl@BankOfErrors

\newif\ifsl@ErrorsOff

\expandafter\gdef\csname sl@family:ErrorState\endcsname{active}
\expandafter\gdef\csname sl@end:ErrorState\endcsname{active}
\def\sl@ErrorNames{}
\def\sl@UnwantedErrors{}
\def\sl@ProtectedErrors{}
```

`\ErrorsOn`

```
\def\ErrorsOn{%
  \@ifnextchar[%
  {\ifsl@ErrorsOff%
      \def\sl@next{\sl@Add\sl@ProtectedErrors}%
    \else%
      \def\sl@next{\sl@Remove\sl@UnwantedErrors}%
    \fi\sl@next}%
  {\global\sl@ErrorsOfffalse%
  \gdef\sl@ProtectedErrors{}%
  \gdef\sl@UnwantedErrors{}}}
```

```
\ErrorsOff
\def\ErrorsOff{%
  \@ifstar%
  {\global\sl@ErrorsOfftrue
    \gdef\sl@UnwantedErrors{}%
    \gdef\sl@ProtectedErrors{}}%
  {\@ifnextchar[{%
    \ifsl@ErrorsOff%
      \def\sl@next{\sl@Remove\sl@ProtectedErrors}%
    \else%
      \def\sl@next{\sl@Add\sl@UnwantedErrors}%
    \fi\sl@next}%
    {\global\sl@ErrorsOfftrue%
    \gdef\sl@UnwantedErrors{}%
    \gdef\sl@ProtectedErrors{latex,}}}}

\ErrorFilter
\sl@ErrorFilter
\sl@@ErrorFilter
\def\ErrorFilter{%
  \global\advance\sl@ErrorNumber1%
  \@ifstar%
    {\expandafter\gdef\csname\the\sl@ErrorNumber:ErrorMode\endcsname{safe}\sl@ErrorFilter}%
    {\sl@ErrorFilter}}

\def\sl@ErrorFilter{%
  \@ifnextchar[%
  {\sl@@ErrorFilter}%
  {\sl@@ErrorFilter[sl@family]}}

\def\sl@@ErrorFilter[#1]#2{%
  \expandafter\ifx\csname #2@ErrorFilter\endcsname\relax%
    \expandafter\xdef\csname #2@ErrorFilter\endcsname{\the\sl@ErrorNumber:sl@#1}%
  \else%
    \expandafter\xdef\csname #2@ErrorFilter\endcsname{%
      \csname #2@ErrorFilter\endcsname,\the\sl@ErrorNumber:sl@#1}%
  \fi%
  \expandafter\ifx\csname #1:ErrorState\endcsname\relax%
    \sl@Add\sl@ErrorNames[#1]%
  \fi%
  \expandafter\ifx\csname #1:ErrorState\endcsname\sl@active%
  \else%
    \ifsl@Immediate%
      \expandafter\gdef\csname #1:ErrorState\endcsname{active}%
    \else%
      \expandafter\gdef\csname #1:ErrorState\endcsname{inactive}%
    \fi%
  \fi%
  \begingroup%
  \expandafter\ifx\csname\the\sl@ErrorNumber:ErrorMode\endcsname\sl@safe%
    \makeatletter%
  \else%
     \catcode`\\12\relax%
  \fi%
  \sl@AddToBankOfErrors}

\sl@AddToBankOfErrors
\long\def\sl@AddToBankOfErrors#1{%
  \sl@TempBOE{#1}%
  \edef\sl@act{%
    \global\noexpand\sl@BankOfErrors{%
    \the\sl@BankOfErrors%
    (:sl@\the\sl@ErrorNumber:)\the\sl@TempBOE(:sl@\the\sl@ErrorNumber:)}}%
  \sl@act%
  \endgroup}
```

```
\ActivateErrorFilters
\def\ActivateErrorFilters{%
  \sl@StateNumber2\relax%
  \@ifnextchar[%
  {\sl@ChangeState}%
  {\sl@ChangeState[\sl@ErrorNames]}}

\DeactivateErrorFilters
\def\DeactivateErrorFilters{%
  \sl@StateNumber3\relax%
  \@ifnextchar[%
  {\sl@ChangeState}%
  {\sl@ChangeState[\sl@ErrorNames]}}

\ifsl@ShowErrors
\else
\expandafter\let\expandafter\sl@GenericError\csname GenericError \endcsname
\let\sl@PackageError\PackageError
\let\sl@ClassError\ClassError
\let\sl@latex@error\@latex@error

\PackageError
  \ClassError
\@latex@error
\def\PackageError#1#2#3{%
  \def\sl@PackageName{#1}%
  \sl@StoreMessage{#2}%
  \sl@PackageError{#1}{#2}{#3}}

\def\ClassError#1#2#3{%
  \def\sl@PackageName{#1}%
  \sl@StoreMessage{#2}%
  \sl@ClassError{#1}{#2}{#3}}

\def\@latex@error#1#2{%
  \def\sl@PackageName{latex}%
  \sl@StoreMessage{#1}%
  \sl@latex@error{#1}{#2}}

\GenericError
\DeclareRobustCommand{\GenericError}[4]{%
  \advance\sl@ErrorCount1\relax%
  \sl@KillMessagefalse%
  \sl@Belongfalse%
  \ifsl@ErrorsOff%
    \expandafter\sl@Belong\sl@ProtectedErrors,sl@end,%
    \ifsl@Belong%
    \else%
      \sl@KillMessagetrue%
    \fi%
  \else%
    \expandafter\sl@Belong\sl@UnwantedErrors,sl@end,%
    \ifsl@Belong%
      \sl@KillMessagetrue%
    \fi%
  \fi%
  \ifsl@KillMessage%
  \else%
    \expandafter\ifx\csname\sl@PackageName @ErrorFilter\endcsname\relax%
    \else%
      \sl@StateNumber1\relax%
      \expandafter\expandafter\expandafter%
      \sl@GetNumber\csname\sl@PackageName @ErrorFilter\endcsname,0:sl@sl@end,%
    \fi%
  \fi%
  \ifsl@KillMessage%
```

```
    \advance\sl@ErrorCasualties1%
  \else%
    \sl@GenericError{#1}{#2}{#3}{#4}%
  \fi%
  \ifcase\sl@Save%
  \or%
    \ifsl@KillMessage%
      \bgroup%
      \let\@unused\sl@Write%
      \sl@GenericError{#1}{#2}{#3}{#4}%
      \egroup%
    \fi%
  \or%
    \bgroup%
    \let\@unused\sl@Write%
    \sl@GenericError{#1}{#2}{#3}{#4}%
    \egroup%
  \fi%
  \gdef\sl@PackageName{NoPackage}}%
  \fi
```

5.4 Debrief

Finally, at the end of the document, we issue a debrief if the user requested it.

A `\clearpage` is needed because we want the final output routine to be processed so we don't miss the last messages if there are some.

```
\AtEndDocument{%
  \ifsl@Debrief%
    \clearpage%
```

Then we do some arithmetics and if messages appeared and some of them were filtered out, we output the warning.

And we say goodbye.

```
    \sl@MessageCount\sl@WarningCount%
    \advance\sl@MessageCount\sl@ErrorCount%
    \sl@Casualties\sl@WarningCasualties%
    \advance\sl@Casualties\sl@ErrorCasualties%
    \ifnum\sl@MessageCount>0%
      \ifnum\sl@Casualties>0%
        \advance\sl@WarningCount-1%
        \PackageWarningNoLine{silence}{%
        There were \the\sl@WarningCount\space warning(s)
        and \the\sl@ErrorCount\space error(s).\MessageBreak%
        \ifnum\sl@Casualties=\sl@MessageCount%
          None survived. This is a violent world%
        \else%
          I've killed \the\sl@WarningCasualties\space warning(s)
          and \the\sl@ErrorCasualties\space error(s)%
        \fi}%
      \fi%
    \fi%
  \fi}

\makeatother
```