# The LaTeX 2ε "msg" package for package localization*

## *Package writer's guide*

Bernard Gaulle

As of 2006/10/03

## Contents

**1 Introduction** **2**

**2 Macros to be used in a LaTeX package** **2**

**3 The message files** **5**

**4 The macros to use in message files** **7**

**5 Testing a message file** **11**

**6 Output options** **11**

**7 A tracing option** **12**

**8 A special option** **12**

**9 Language options** **12**

**10 Migration scheme** **13**

**11 Generated files** **13**

**12 Volunteers** **14**

**13 Thanks** **14**

*This file has version V0.40 last revised 2006/10/03

# 1 Introduction

Since a LaTeX package issues various messages, mostly in English (but, unfortunately, often in an English-like language), it is usefull to provide a feature to localize any LaTeX $2_\varepsilon$ package or document class. The "msg" package is designed for that. Messages are in dedicated files and retrieved when needed.

Packages writers (as well as document class writers) just have to create their native messages file and ask for any message when needed.

`\issuemsgio`

# 2 Macros to be used in a LaTeX package

Basically, three macro commands can be coded for package localization: `\issuemsg`, `\getmsg` and `\retrievemsg`. Another macro, `\issuemsgx` is given for specific cases, we will see that later, page 8.

## Output a message

Here is the main macro which will issue a message "id" via the command `\issuemsgio` defaultly set to `\typeout` .

`\issuemsg` `[#1]#2(#3)[#4]`

`\issuemsgx` `[#1]#2(#3)[#4]`

One can provide another command to issue the message by the way of the first optional argument. The second argument is the message "id"; the third is the name of the package (or document class[1]) which provides that message through a message file whose name is `[language_]package-msg.tex`. Lastly an optional parameter can be set to "`#1`[2]" to forward an argument directly inside the message content. I thought the syntax would be too much complicated to offer much more parameters through that mean. We will see later that the message can also be split in four parts, allowing anyone to display the message differently but not as `\issuemsg` does.

`\issuemsg[`⟨*message_macro*⟩`]{`⟨*id*⟩`}(`⟨*package*⟩`)[#1]`

The message macro could be the usual `\typeout` or any other output macro with one argument such as `\message` or `\wlog`. You can also code a macro with two arguments, such as `\ClassWarning`, `\ClassWarningNoLine`, `\PackageWarning`,

[1] Each time we are talking about a LaTeX package, please consider it applies also to any LaTeX document class.

[2] You can put here any replacement text instead of this `#1` parameter. Be careful, this parameter or replacement text will be, usually, expanded; so protect any string which should not with a `\string` prefix.

`\PackageWarningNoLine`, `\ClassInfo` or `\PackageInfo` which have just a name as first argument, like this:

`\issuemsg[⟨message_macro{arg1}⟩]{⟨id⟩}(⟨package⟩)[#1]`

That way the first argument is not localized (usually this is a class or package name) and the second argument is provided by the message file entry and so localized.

You can also use special error macros with 3 arguments as explained below.

## Willing to issue a `\PackageError`?

The "msg" package is designed for basic macro messages which have just only one text argument to localize. The `\PackageError` is one exception; it has 3 arguments: the first one (name of package in error: <*package1*>) which is given as the following:

`\issuemsg[\PackageError{⟨package1⟩}]{⟨id⟩}(⟨package2⟩)[#1]`

the other two arguments will be retrieved from the message file of ⟨*package2*⟩ and localized.

The same coding can be used for `\ClassError`:

`\issuemsg[\ClassError{⟨class1⟩}]{⟨id⟩}(⟨class2⟩)[#1]`

## Examples & tests (using the message files listed page 6)

`\issuemsg1(msg)` will give at the console:

```
2006/10/03 V0.40 package to issue localized messages, now loaded.
```

but `\issuemsg01(msg)` will give: `erroneous message id ‘‘01’’`.

While defining `\def\test#1{‘\issuemsg4(msg)[#1]’}` the following call `\emph{\test{SPECIAL}}` will print:

*‘This is to test the SPECIAL feature’*

showing that the argument was inserted inside the message at the exact area, replacing `#1`.

In a French document, the same codes will issue:

```
2006/10/03 chargement de l’extension de localisation (V0.40).
          le message id ‘‘01’’ n’est pas répertorié
```

*‘Ceci est pour tester le dispositif SPECIAL’*

## Get a message for typesetting

Sometimes we only want to get the message and typeset it. The syntax is the same as `\issuemsg` except there is no first optional argument for providing the macro name to issue the message since it is not issued at all.

`\getmsg{⟨id⟩}(⟨package⟩)[#1]`

## Examples & tests

‘`\textsl{\getmsg1(msg)}`’ will insert at this point for typesetting the message: ‘*2006/10/03 V0.40 package to issue localized messages, now loaded.*’; nothing is issued at the console nor in the log, except if the <*id*> is not found in the message file.

In a French document, the same code will issue:

‘*2006/10/03 chargement de l'extension de localisation (V0.40).*’

`\getmsg` `#1(#2)[#3]`

## The message file input routine

To avoid having superfluous file names listed in the `log` each time we request a message from a file, defaultly we read the file with the TEX `\read` command.

NOTICE: all messages read until the requested one (included) are expanded (before parsing) in the following macro. Thus, each (`\msg`) macro call should contain significant value or `\protect` a macro. The same reason applies to `\msgheader` and `\msgtrailer` we will discuss later (cf p 10).

## Just retrieve `\themsg`

One can also want to retrieve the message from the file and save it in a macro for later use. In fact, if this is a one-part message it will be saved in `\themsg`, otherwise this will be the first part message and other parts will be saved in `\themsgi`, `\themsgii` and `\themsgiii`.

`\retrievemsg` `#1(#2)[#3]`

`\themsg`
`\themsgi`
`\themsgii`
`\themsgiii`
`\msgid`

The `\retrievemsg` command is the heart of all macros to obtain the wanted message. It will input the message file, depending on the language to use, searching for the message “id”. If this is a valid language (i.e. defined in the `language.dat` file in use) and the corresponding language file exists it is inputed otherwise it is the package default message file which is inputed (which should be usually in English). In case the message “id” is still not found in that file and no “`*`” message “id” exists we will try to access message number 6 of the “msg” package. And again, if still not found we terminate the process with a final package error; this is the only English message hard coded in the “msg” package. Localization might occur but should be also hard coded; not really usefull since it is not addressed to the end user but to a package or class writer.

`\retrievemsg{`⟨*id*⟩`}(`⟨*package*⟩`)[#1]`

### Examples & tests

`\retrievemsg1(msg)` will set `\themsg` with the value of the corresponding message ; nothing is issued at the console nor in the log, except if the <*id*> is not found in the message file.
`\show\themsg` will explain:

```
> \themsg=macro:
->2006/10/03 V0.40 package to issue localized messages, now loaded.
```

In a French document, the same code will issue:

```
> \themsg=macro:
->2006/10/03 chargement de l'extension de localisation (V0.40).
```

In case the message file is empty or do not contain neither the message "id" nor any `\msg{*}` macro then we will obtain:

```
> \themsg=macro:
->msg package:  UNUSUAL end of file reached when loading msg-msg.tex
file!
```

It may also arrive that we don't find that former message (`#6`) at all then we will issue the usual LaTeX `\PackageError` macro.

## 3 The message files

The default (English) message files should have the name *package*`-msg.tex` and localized ones should be *language*`_`*package*`-msg.tex`. I would have prefered the file names begin with a dot which is a hidden file in unix and thus avoid visual pollution inside packages directories but, unfortunately, writing them with `doctrip` is generally forbiden due to '`openout_any = p`' in `texmf.cnf` configuration file.

These files contain only the messages which could be requested by the associated *package*. It is important to say now that when a message is split on multiple lines, each line must end with "␣%" to avoid to loose the ending space when any is required; this is due to the special `\read`ing process shown .

### Message files contents

A typical file content is the following `msg-msg.tex` file, used by the "msg" package itself:

⟨/code⟩

```
1 ⟨*english⟩
2 % File: msg-msg.tex
3 % Here are the English messages for the \msgname\  package.
4 %
5 % The following line is just for testing purpose:
6 \msgencoding{}\msgheader{}\msgtrailer{}
7 \msg{1}{\filedate\space \fileversion\space package to issue localized %
8         messages, now loaded.}{}
9 \msg{2}{invalid optional parameter provided:}{}
10 \msg{3}{invalid language requested: ``\CurrentOption''}{}
```

```
\msg{4}{This is to test the #1 feature}{}
\msg{5}{``msg'' package line number }{\msgparti{issues %
        \#\msgid\ #1 message}}
\msg{6}{msg package: UNUSUAL end of file reached when %
        \MessageBreak %
        loading \msg@filename\space file!}{}
\msg{7}{\string\msg\space syntax error}{\help{last % special test case
        argument is missing.}}
\msgheader{MESSAGE\space\msgid:\space``}\msgtrailer{''}
\msg{8}{here is a customized message}{}%
\msg{9}{here is a customized message %
        \MessageBreak which continuation is aligned}{}
\msgheader{Message\space\msgid\space(msg):\space}\msgtrailer{}
\msg*{10}{****** I emphasize: this is a WARNING! ****** %
          \MessageBreak ****** Be careful.^^J}{}
\msgheader{}\msgtrailer{}
\msg{11}{The msg package is in use with ``tracefiles'' option.}{}
\msg{12}{A risk of infinite loop arose; %
         \MessageBreak %
         please check the message file: \msg@filename}%
         {\help{Look at rules to apply in messages files.}}
\msg{*}{erroneous message id ``\msgid''}{}
```
⟨/english⟩

If necessary, one can link `english_msg-msg` to that file, but since English is always the default language for LaTeX this is useless.

The same messages, localized for French, are located in the following messages file (`french_msg-msg.tex`):

```
⟨*french⟩
% Fichier french_msg-msg.tex
% Ici on trouve les messages en francais pour l'extension \msgname\ .
%
\msgencoding{latin1}\msgheader{}\msgtrailer{}
\msg{1}{\filedate\space chargement de l'extension de %
        localisation (\fileversion).}{}
\msg{2}{le paramètre optionnel est invalide}{}
\msg{3}{le langage demandé (\CurrentOption) n'existe pas}{}
\msg{4}{Ceci est pour tester le dispositif #1}{}
\msg{5}{ligne }{\msgparti{de l'extension ``msg'' génère le %
                message de #1 \#\msgid}}
\msg{6}{extension msg \string: fin ANORMALE de fichier rencontrée %
        \MessageBreak %
        en chargeant le fichier \msg@filename\space\string!}{}
\msg{7}{erreur de syntaxe à l'appel de \string\msg}% cas special de test
       {\help{il manque le dernier argument.}}
\msgheader{MESSAGE\space\msgid\space\string:\space %
           \string<\string<\space}
\msgtrailer{\space\string>\string>}
\msg{8}{ceci est un message personnalisé}{}
\msg{9}{ceci est un message personnalisé %
```

```
        \MessageBreak et aligné}{}
\msgheader{Message\space\msgid\space(msg)\space\string: %
           \space}\msgtrailer{}
\msg*{10}{****** Je mets en valeur \string: %
          ceci est un AVERTISSEMENT ! ****** %
          \MessageBreak ****** Soyez prudent.^^J}{}
\msgheader{}\msgtrailer{}
\msg{11}{L'extension msg est en service avec l'option %
         \string<\string< tracefiles \string>\string>.}{}
\msg{12}{Un risque de boucle infinie a été rencontré \string; %
         \MessageBreak %
         vérifier le fichier des message \string: \msg@filename}%
       {\help{Voir les règles à appliquer dans les fichiers de messages.}}
\msg{*}{le message id ``\msgid'' n'est pas répertorié}{}
⟨/french⟩
```

⟨*code⟩

Notice that you can have messages which call any internal macro name since the `\catcode` for `@` is assigned to letter when the message file is read in.

# 4 The macros to use in message files

The simpliest way to code a message is:

`\msg{`⟨*id*⟩`}{`⟨*message*⟩`}{}`

The last message in the file should have ⟨*id*⟩ equal to `*` to say that, when reached, no valid ⟨*id*⟩ was found in the file and the "msg" package should issue (or get or retrieve) that error *message*, which could be e.g. `\msg{*}{erroneous message id ``\msgid''}`.

`\msg` `#1#2#3`

`\help` `#1`

When the `\msg{*}` is reached the "msg" package will issue a `\PackageWarning` with that message, but no line number is sent because the message file is still not closed and the current line number is that from the message file. The message is also forwarded to the TEX mouth even when an `\issuemsg` was requested.

## To build a `\PackageError` or `\ClassError` message

A `\PackageError` has a message part and a help part; these are given as the following:

---

`\msg{<`*id*`>}{<`*message-part*`>}{\help{<`*help-part*`>}}`

---

in that special case you can't build a multi-parts message as explained in the following section.

### To build a multi-parts message

When given, the optional argument provides 3 additional message parts. Here is the syntax :

---

`\msg{<`*id*`>}{<`*message-part1*`>}{\msgparti{<`*message-part2*`>}`
`\msgpartii{<`*message-part3*`>}`
`\msgpartiii{<`*message-part4*`>}}`

(Any `\msgparti*` can be omitted)

---

When retrieved, `\themsg` will contain *message-part1*, `\themsgi` the *message-part2*, `\themsgii` the *message-part3* and `\themsgiii` the *message-part4*. One can build the wanted message with the mix of these four parts and other materials. When requested via `\getmsg` or `\issuemsg` the four parts are sticked in the usual order.

`\msgparti #1`

`\msgpartii #1`

`\msgpartiii #1`

## Input encoding discussion

It is assumed that, defaultly, `\issuemsg` will finally provide a message to the console (without any encoding). The macro `\getmsg` is designed for typesetting (usually with an input encoding). Since `\retrievemsg`'s target is unknown, display or typesetting, we let the package or class maker to decide which input encoding has to be set up.

If you want to issue a message but not to the console, you should probably use the macro `\issuemsgx` in place of `\issuemsg`.

### For messages issued to the console

If your messages can be coded in 7bits, no problem except that you probably need to avoid macros like `\aa`, `\oe`, `\ae`, etc. which can't output as expected on the console or log file. If your messages use 8bits characters, these 8bits characters will be output asis (until the LaTeX team introduces a real output encoding for the console[3]).

[3]The "`msg`" package is already designed for any output encoding; this is the `\kbencoding` macro call which can do that (as currently done by my experimental `keyboard` package).

### For messages to be typeset

If any of your messages use at least one 8bits character, you need to specify which input encoding you are using:

`\msgencoding{⟨input encoding⟩}`

you just have to give the name of the input encoding, exactly like with `\inputencoding` for the `inputenc` package. This is usually the first command in the messages file.

`\msgencoding #1`

### Disadvantage using `\msgencoding`

As the `inputenc` package is automatically loaded when `\msgencoding` is executed, there is a risk you try loading again `inputenc` in the preamble via `\usepackage`. This is the case if any message was already issued by the "msg" package and then an option clash will occur and force you to put the encoding option as a global option in `\documentclass`.

Since "msg" is calling `inputenc` there will be real difficulties to localize the `inputenc` package itself.

## The messages file rules

Due to the `reading` process you should apply the following rules in a message file:

**Rule 1:** A line can be a comment (begining with %).

**Rule 2:** A line can begin with `\msg`,`\msgheader`,`\msgtrailer` or `\msgencoding`.

**Rule 3:** A `\msg` line can be continued on the next line(s), assuming each one ends with ␣%.

**Rule 4:** The following macros: `\msgheader`, `\msgtrailer`, or `\msgencoding` are always executed.

**Rule 5:** Spacing inside `\msgheader` and `\msgtrailer` should be made only by the use of the macro `\space`.

**Rule 6:** Any of these three macros should be expandable at any time.

**Rule 7:** No other macro command can begin a line.

**Rule 8:** All 8bits characters used should in the range specified by the macro command `\msgencoding`.

That's all!

## To build messages with header and/or trailer

`\msgheader` `#1`

`\msgtrailer` `#1`

Before any `\msg` call you can specify which header and/or trailer you want in the following message or messages. You just have to specify them:

---

`\msgheader{<`*my_header*`>}`
`\msgtrailer{<`*my_trailer*`>}`

---

When a message needs to be continued on the next line you just have to insert `\MessageBreak` where you want the new line will start and then the "msg" package will try to align the following text by adding the same number of `\space`s as tokens in the expanded `\msgheader`. That feature only applies with `\issuemsg` and only to the `<`*message-part1*`>` (the three other message parts can not have any header or trailer).

## To emphasize a message

When you want to emphasize a message you just have to code the star form of the `\msg` macro:

---

`\msg*{<`*id*`>}{<`*message-part1*`>}{<`*part2*`>>}`

---

and then the message will be issued after a line skip on the console and log. If your message ends with `^^J` another line will be skiped after. Obviously this feature only works with `\issuemsg` but not with `\getmsg` or `\retrievemsg`.

## Examples & tests

We use message # 5 of `msg-msg.tex` file as following:
`\def\foo#1{\retrievemsg5(msg)[#1]\themsg\the\inputlineno\ \themsgi}`

then `\emph{The \foo{test}}` will generate:

The *"msg" package line number 1194 issues #5 test message*

In a French document, the code `\emph{La \foo{test}}` will issue:

*La ligne 1200 de l'extension "msg" génère le message de test #5*

We will now use the message # 8 of `msg-msg.tex` file in order to show the customization set in that file.

`\getmsg8(msg)` will generate:
MESSAGE 8: "here is a customized message"

The following `\texttt{\getmsg9(msg)}` using `\MessageBreak` will give at the console:
`MESSAGE 9:  ``here is a customized message`

```
                which continuation is aligned''
```
(same message issued to the console).
And in a French document, the same calls will issue:
MESSAGE 8 : « ceci est un message personnalisé »
```
MESSAGE 9 :  << ceci est un message personnalisé
                et aligné >>
```
Below is a test of an emphasized message (`\texttt{\getmsg{10}(msg)}`):

```
Message 10 (msg):  ****** I emphasize:  this is a WARNING! ******
                   ****** Be careful.
```
(same message issued to the console).

# 5 Testing a message file

When you are building a messages file few typing errors can occur; so there is a need to test that file. You can do this by using the following macro call:

`\issueallmsg[`⟨*message_macro*⟩`](`⟨*package*⟩`)`

All messages will be retrieved and issued as requested (with the message macro) but not, perhaps, with the exact macro call which will be used in the LaTeX $2_\varepsilon$ package or document class. Specially, the `\help` macro call does nothing and `\msgparts` are listed in the order found in the file. You should also notice that all macros used inside the messages should be expandable. This macro is allways executed with the `tracefiles`[4] option.

Here is an example using the "msg" file (`\issueallmsg[\wlog](msg)`) please check the log file to find the output.

`\issueallmsg [#1](#2)`

# 6 Output options

`\usepackage[`⟨*output-options*⟩`]{msg}`

The output options provide to the "msg" package with a running macro name to issue any message, in replacement of the default `\issuemsgio` macro initialized at the begining (cf page 2). Currently the following macro names are defined as options:

`\usepackage[message|wlog|typeout|kbtypeout]{msg}`

These options are related to basic messages macros but this is not an exhaustive list of macros which can be called by `\issuemsg`. Specially, `\PackageError` can also be used, as already discussed p. 3 and p. 7. The last one macro name is coming from a package of mine (keyboard) doing input encoding and output decoding.

[4] I never found the right code to avoid a loop with `tracefiles` option...

# 7 A tracing option

\usepackage[⟨*tracefiles*⟩]{msg}

The `tracefiles` option changes the processing for reading messages files. Defaultly these files are not read with the usual `\input` macro and so the files are not listed in the `log` file. When giving the `tracefiles` option messages files are read with a `\input` macro like and then the full path names are listed.

# 8 A special option

\usepackage[⟨*noop*⟩]{msg}

The `noop` option changes the processing: don't read the messages files and just provides the "msgid" as text message with the following header: `0<msg noop>`.

# 9 Language options

\usepackage[⟨*output-options*⟩,⟨*language-name*⟩]{msg}

Defaultly messages are issued from the message file dedicated to the document running language. One can force the "msg" package to use a specific language (assuming it was defined in the `language.dat` file in use), just give it as the last option.

`\nativelanguage #1`

This optional macro is usefull to understand and/or translate correctly the message files. Each package writer can define it as i currently do for the "msg" package here. To obtain messages in writer's native language, just call the "msg" package with the option '`native`':

\usepackage[⟨*output-options*⟩,native]{msg}

The last option, if undeclared as an option, is assumed to be a language name. If that language is unknown (i.e. undefined in the `language.dat` file in use) we issue a warning with message number 3, otherwise that language name will be now the prefix for all the message files names searchs. It overides any previously set (in format, package, ...) language name for use by the "msg" package.

Notice that, when set (and forced) as option, the "msg" language name can't be changed any way inside the document (no language change can modify that).

# 10 Migration scheme

If you want to migrate a document class ou package issuing messages in order it could be localized for any language, you just have to follow the following steps:

1. Chose the native language and create the related messages file.
2. Chose the input encoding and define it in the messages file with the macro `\msgencoding`.
3. Insert the final error message (`\msg(*){...}{}`).
4. Isolate all messages in the code.
5. Attach to each message a unique *id*.
6. Comment each message argument(s) and put these arguments in the message file as:
   `\msg{`*id*`}{`*argument1*`}{<`*argument2*`>}`
7. Replace the message macro call (`\typeout` or `\wlog` or ...) by :
   `\issuemsg[<`*message macro*`>]{`*id*`}(`*package*`)`
8. Check the syntax carefully, specially, for error messages and when a parameter will be used.
9. Test each message independtly and tune it considering the display option, the header and trailer facilities, etc.
10. Test the messages file with `\issueallmsg`.
11. Consider other translations of your native messages file.
12. Release the new code along with its messages files.

# 11 Generated files

Currently the file `msg.ins` is the file to compile with (La)TeX to generate all the "msg" stuff. Here is `msg.ins` showing the `doctrip` generated files:

---

```
\def\batchfile{msg.ins}
\input docstrip.tex
\askonceonly
\keepsilent
\preamble

  File is part of the "msg" package for LaTeX
  which is designed to localize any LaTeX package
  or document class.
```

```
\endpreamble
\generateFile{README_msg_doc.txt} {t}{\from{msg.dtx}{README}}
\generateFile{msg.sty}            {t}{\from{msg.dtx}{code}}
\generateFile{msg-msg.tex}        {t}{\from{msg.dtx}{english}}
\generateFile{french_msg-msg.tex} {t}{\from{msg.dtx}{french}}
\generateFile{norsk_msg-msg.tex}  {t}{\from{msgfiles.dtx}{norsk}}
\generateFile{german_msg-msg.tex} {t}{\from{msgfiles.dtx}{german}}
\Msg{*************************************************}
\Msg{* "msg" package is now generated, please move msg.sty}
\Msg{* file and *-msg.tex messages files in the           }
\Msg{* appropriate directory where LaTeX can find them.   }
\Msg{*                                                    }
\Msg{* For TeX Live please do:  "make TL"                 }
\Msg{*************************************************}
\endinput
```

---

# 12 Volunteers

Volunteers are welcome to translate the "msg" package message file (the English one or the native one in French) in their mother language. Lot of thanks to them!

# 13 Thanks

The following people contributed to that project and we really appreciate their effort for testing, translating, documenting, etc.: Hans F. Nordhaug, Harald Harders.

Enjoy!

∗–∗