

Le paquetage `makegobbler` v0.3.2 (2026-02-15)

Vincent Belaïche

15 février 2026

Table des matières

1	Introduction	2
2	Usage et limitations	2
2.1	Motivations	2
2.2	Limitations de <code>makegobbler</code>	4
3	Installation	5
3.1	Installation à partir du CTAN	5
3.2	Installation à partir du dépôt SVN	6
4	Chargement du paquetage	6
5	Options du paquetage	6
6	Interface utilisateur	7
6.1	Macros pour un gobage ou dégobage conditionnel	7
6.2	Macros pour un gobage ou dégobage inconditionnel	8
7	Remerciements	9
8	Le code	9
8.1	Options du paquetage	9
8.2	Interface utilisateur	9
8.2.1	Macros de gobage/dégobage conditionnel	9
8.2.2	Macros de gobage/dégobage inconditionnel	10
8.3	Gestion des crochets de fin de traitement	10
8.4	Gestion de la pile de macros crochets	11
8.5	Gobage/dégobage dans le cas d'une condition à la \TeX	12
8.6	Gobage/dégobage dans le cas d'une condition à la \LaTeX	13
8.7	Macros auxiliaires de <code>\MGB@g</code>	13

Liste des exemples

1	Exemple introductif	2
2	Gobage conditionnel classique à la \TeX	2

3	Gobage conditionnel classique à la \LaTeX	3
4	Limitation d' \ifthenelse avec des caractères spéciaux	3
5	Caractères spéciaux avec une condition à la \TeX	3
6	Limitations d' $\if\dots$	4

1 Introduction

Le but du paquetage `makegobbler` est de faciliter l’omission conditionnelle d’une portion de document. Un exemple valant mieux qu’un long discours :

```
\newif\ifgarde
...
\MGBkeep*\ifgarde
\LaMacroQuiGobeTout
Bla Bla bla, texte inséré conditionnellement seulement si
\gardetrue.
\LaMacroQuiGobeTout
```

EXEMPLE 1 – *Exemple introductif*

Dans l’exemple ci-dessus la macro `\LaMacroQuiGobeTout` est ci-après désignée par *délimiteur*, et on peut utiliser n’importe quel nom \langle délimiteur \rangle de macro \langle délimiteur \rangle au lieu de `\LaMacroQuiGobeTout`. Dans le cas où `\gardefalse` tout ce qui est entre les deux \langle délimiteur \rangle s est supprimé (gobé), et dans le cas inverse, quand `\gardetrue`, c’est conservé (dégobé).

Dans le premier cas, le premier \langle délimiteur \rangle est défini comme une macro de gobage, ou gobeur, gobant tout jusqu’à une marque de fin qui est le second \langle délimiteur \rangle vu comme une string¹, et c’est la raison du nom `makegobbler` du paquetage, parce qu’il fait du premier \langle délimiteur \rangle un gobeur.

2 Usage et limitations

2.1 Motivations

Le paquetage `makegobbler` n’est pas fait pour remplacer les constructions conditionnelles de \TeX du genre de :

```
\newcommand*\Un{2}%
...
\ifnum\Un=1 C’est un\else C’est deux\fi
```

EXEMPLE 2 – *Gobage conditionnel classique à la \TeX*

ni plus la macro `\ifthenelse` du paquetage \LaTeX `ifthen` qu’on peut utiliser ainsi :

1. `string` en \LaTeX signifie une chaîne d’unités lexicales qui sont des caractères de catcode `other` ou `space`, dans le cas de `makegobbler` le \langle délimiteur \rangle de fin ne comprend que des caractères de catcode `other`.

```
\newcommand*\Un{2}%
...
\ifthenelse{\Un=1}{C'est un}{C'est deux}
```

EXEMPLE 3 – Gobage conditionnel classique à la \LaTeX

bien au contraire, `makegobbler` s'appuie justement sur ces constructions pour tester les conditions de gobage ou dégobage, et son objectif est plutôt de pallier certaines limitations du gobage effectué par ces méthodes classiques. L'exemple 4 illustre² cela : si vous décommentez le `\alternativetrue` il produit une erreur `You can't use 'macro parameter character #' in horizontal mode..`

```
\newif\ifalternative
%\alternativetrue
...
Viens petite fille dans mon comic strip
...
\ifthenelse{\boolean{alternative}}{\verb+#!+}{SHEBAM !} POW !
BLOP ! WIZZ !
```

EXEMPLE 4 – Limitation d'`\ifthenelse` avec des caractères spéciaux

Cela se produit parce que `\verb` fonctionne en changeant le régime de catcode de ce qui suit, mais pour que cela fonctionne il faut que `\verb` soit exécuté³. Or si on passe `\verb+#!+` en argument à `\ifthenelse` le catcode de `#` est fixé au moment de la capture de l'argument, car celle-ci impose de passer le premier étage⁴ de \TeX , celui qui convertit les caractères en unités lexicales, et donc `#` conserve son catcode initial 6 (qui veut dire argument de macro) et `\verb` n'y pourra rien changer.

Le paquetage `makegobbler` résout le problème de l'exemple 4 parce que les sections dégobées ne sont pas passées en argument de macro, et d'ailleurs pour la même raison l'utilisation d'une condition à la \TeX aurait aussi résolu le problème, comme dans l'exemple 5 ci-dessous :

```
\newif\ifalternative
\alternativetrue
...
Viens petite fille dans mon comic strip
...
\ifalternative\verb+#!+\else SHEBAM !\fi\space POW ! BLOP ! WIZZ !
```

EXEMPLE 5 – Caractères spéciaux avec une condition à la \TeX

Bien sûr dans ces exemples simplistes il aurait suffi d'écrire `\texttt{\#!}` au lieu de `\verb+#!+` et cela aurait aussi fonctionné, mais le prix de cela est d'écrire

2. Aux non-francophones, c'est extrait des paroles de la chanson *Comic strip* de Serge Gainsbourg, la drôlerie des paroles vient qu'en français *strip* n'est connu que dans son sens *dévêtir* et non dans son sens *bande*.

3. Le deuxième étage de \TeX , c-à-d. la gueule du lion.

4. Les yeux du lion.

un code moins lisible.

Utiliser des conditions à la T_EX n'est toutefois pas une solution ultime, en effet il reste des cas qui ne fonctionnent pas même avec les conditions à la T_EX, l'exemple 6 en illustre un :

```
\newif\ifok
\oktrue
...
\ifok À minuit les carrosses redeviennent des
c\verb+\iftrue+illes.\fi
```

EXEMPLE 6 – Limitations d'*\if*...

Si vous mettez en commentaire le `\oktrue` alors il y aura une erreur de compilation `Incomplete \iffalse; all text was ignored after line ...`. La raison pour laquelle cela échoue est que le `\ifok` pour gober tout ce qui suit jusqu'au `\fi` suivant teste tout de même s'il y a des unités lexicales de type `\if...`, de sorte à tomber sur le bon délimiteur de fin `\fi`. En faisant cela il considère par erreur que le `\iftrue` qui est passé dans `\verb+...+` attend un `\fi`, et donc il considère que le `\fi` suivant correspond à ce `\iftrue` et non à `\ifok`.

Avec `makegobbler` ce problème est résolu parce que l'utilisateur donne un délimiteur `\langle délimiteur \rangle` arbitraire, et donc en cas de gobage avec imbrications de plusieurs sections en gobage/dégobage conditionnels il suffit d'utiliser un `\langle délimiteur \rangle` propre à chaque niveau d'imbrication.

Dans cette rubrique j'ai surtout mentionné les problèmes qu'on peut avoir avec `\verb` que vous n'utilisez peut-être jamais. Sachez que c'est loin d'être le seul cas, et que toutes macros ou environnements qui jouent avec le régime de catcode sont sujets aux mêmes types de limitations, et d'ailleurs les questions posées par les débutants sur les forums ne concernent pas rarement la survenue de cette famille de problèmes. Notons par exemple⁵ que :

- `\includegraphics` avec les paquetages `graphicx`, `grffile` est capable de traiter correctement certains caractères spéciaux, comme par ex. `~`, au sein d'un nom de fichier en changeant son régime de catcode,
- `\href` et `\url` du paquetage `hyperref` sont également concernés pour traiter correctement certains caractères spéciaux au sein d'URLs, ou encore
- l'environnement `lstlisting` et la macro `\lstinline` de l'excellent paquetage `listings`, par exemple avec un listing en langage C on peut avoir typiquement des directives préprocesseur `#if CONDITION` qui poseraient problème si vous essayez de les dégober avec `\ifthenelse`.

2.2 Limitations de `makegobbler`

Si vous avez lu attentivement la rubrique §2.1 alors vous savez déjà qu'utiliser les macros de `makegobbler` au sein d'un argument de macro est voué à l'échec : comme `\verb` les macros de `makegobbler` jouent avec les catcodes, et donc produisent le même type de limitations.

Au sein de la plupart des environnements, cela au contraire va fonctionner. Et d'ailleurs si cela ne fonctionnait pas au sein de l'environnement `document`, alors

5. Cette liste n'est en aucun cas exhaustive.

le paquetage serait inutilisable. Toutefois avec certains environnements cela ne fonctionnera pas. En effet un environnement $\langle env \rangle$ peut être de deux types :

- soit c'est un environnement *fil-de-l'eau*, dans ce cas le $\backslash begin\{\langle env \rangle\}$ et sa contrepartie $\backslash end\{\langle env \rangle\}$ sont juste comme deux macros qui activent et désactivent certaines conditions de traitement de ce qu'il y a entre les deux, par exemple la police, ou bien le régime de catcode ou encore le paramétrage de formation de paragraphes,
- soit c'est un environnement *aspirant*, cela signifie que tout ce qui se trouve entre $\backslash begin\{\langle env \rangle\}$ et $\backslash end\{\langle env \rangle\}$ est préalablement aspiré comme si c'était l'argument d'une macro de sorte à avoir une liste d'unités lexicales, et qu'ensuite cette liste est post-traitée.

Les macros de `makegobbler` ne vont donc fonctionner correctement pour gober/dégober des sections de code entre $\backslash begin\{\langle env \rangle\}$ et $\backslash end\{\langle env \rangle\}$ que si $\langle env \rangle$ est un environnement fil-de-l'eau, mais elles ne fonctionneront pas avec un environnement aspirant, puisqu'on se retrouve dans la même situation que si le code était au sein d'un argument de macro. Voici quelques environnements aspirants :

- la plupart des environnements du paquetage `amsmath`,
- tous les environnements créés avec la macro `\NewEnviron` du paquetage `environ`, et
- l'environnement `frame` de la classe `beamer`, sauf dans le cas où il reçoit l'argument optionnel `fragile`.

3 Installation

Ci-dessous la variable d'environnement `TEXMF_INSTALL_DIR` pointe sur le répertoire `TEXMF` où l'on désire effectuer l'installation.

3.1 Installation à partir du CTAN

Le script bash suivant peut être utilisé pour installer le paquetage à partir du CTAN, vous aurez besoin d'outils tels que `wget`, `latexmk`, `perl` (pour faire tourner `latexmk`) et d'une distribution \LaTeX :

```
# crée un répertoire temporaire pour faciliter le nettoyage
work_dir=$(mktemp -d /tmp/makegobbler-install.XXXXXX)
pushd $work_dir
# obtient le paquetage
wget https://mirror.ctan.org/macros/latex/contrib/makegobbler.zip
# unzip
unzip makegobbler.zip
cd makegobbler
# set TEXMF_INSTALL_DIR to the installation target TEXMF
[ -d "$TEXMF_INSTALL_DIR" ] || TEXMF_INSTALL_DIR=$(HOME)/texmf
pdfTeX makegobbler.ins
mkdir -p "$TEXMF_INSTALL_DIR/tex/latex/makegobbler"
cp makegobbler.sty "$TEXMF_INSTALL_DIR/tex/latex/makegobbler"
# génère la doc, sautez cette étape pour ne pas recompiler le manuel
latexmk -pdf makegobbler.dtx
mkdir -p "$TEXMF_INSTALL_DIR/doc/latex/makegobbler"
```

```
cp makegobbler.pdf "$TEXMF_INSTALL_DIR/doc/latex/makegobbler/makegobbler-do
# nettoyage
popd
rm -fr $work_dir
```

3.2 Installation à partir du dépôt SVN

Si vous désirez — à vos risques et périls — la toute dernière version de développement possiblement encore non publiée sur le CTAN alors vous pouvez utiliser le script bash suivant. Vous avez besoin d'un client `svn` ainsi que du `make` de GNU.

```
# crée un répertoire temporaire pour faciliter le nettoyage
work_dir=$(mktemp -d /tmp/makegobbler-install.XXXXXX)
pushd $work_dir
# obtient les sources du paquetage
svn checkout https://svn.gnu.org.ua/sources/latex-makegobbler/trunk .
# set TEXMF_INSTALL_DIR to the installation target TEXMF
[ -d "$TEXMF_INSTALL_DIR" ] || TEXMF_INSTALL_DIR=$(HOME)/texmf
export TEXMF_INSTALL_DIR
# make est le GNUmake
make install README_EXT=org
# nettoyage
popd
rm -fr $work_dir
```

4 Chargement du paquetage

Charger le paquetage en plaçant dans votre préambule :

```
\usepackage{makegobbler}
```

pour un chargement sans options, ou bien :

```
\usepackage[options]{makegobbler}
```

pour passer *options*, cf. § 5.

5 Options du paquetage

Le paquetage a deux options :

ifthen Juste une commodité pour charger le paquetage **ifthen**, ceci est nécessaire pour utiliser les macros de (dé)gobage conditionnel avec une condition à la \LaTeX .

tense Ne pas utiliser sauf si vous savez ce que vous faites. Voir la documentation du code § 8.5 page 13, après la ligne de code 70.

6 Interface utilisateur

6.1 Macros pour un gobage ou dégobage conditionnel

Le paquetage offre quatre macros pour le (dé)gobage conditionnel d'un bout de code L^AT_EX : `\MGBkeep`, `\MGBdrop`, `\MGBkeepelse`, et `\MGBdropelse`. Les deux premières utilisent deux `\langle délimiteur \rangle`s, et les deux dernières trois.

Chacune de ces macros a pour premier argument une condition. La syntaxe est la suivante :

```
\MGBkeep⟨étoile⟩⟨condition⟩⟨délimiteur⟩
⟨dégobé⟩
⟨délimiteur⟩
```

`⟨dégobé⟩` peut être n'importe quoi entre les deux `\langle délimiteur \rangle`s, et il est dégobé si `⟨condition⟩` est vraie, et gobé sinon. Avec `\MGBdrop` c'est le contraire, la condition de débogage est ⁶ `¬⟨condition⟩`.

Avec les deux autres macros, `\MGBkeepelse` et `\MGBdropelse`, on a deux branches, l'une contenant `⟨gobé⟩` et l'autre `⟨dégobé⟩` :

```
\MGBkeepelse⟨étoile⟩⟨condition⟩⟨délimiteur⟩
⟨dégobé⟩
⟨délimiteur⟩
⟨gobé⟩
⟨délimiteur⟩
```

Dans ce cas si `⟨condition⟩` est vraie, `⟨dégobé⟩` est dégobé, et `⟨gobé⟩` est gobé, et sinon, si `⟨condition⟩` est faux, c'est le contraire. En utilisant `\MGBdropelse`, la branche `⟨gobé⟩` correspondant au gobage sous `⟨condition⟩` est donnée en premier, et celle `⟨dégobé⟩` correspondant au dégobage en second.

```
\MGBdropelse⟨étoile⟩⟨condition⟩⟨délimiteur⟩
⟨gobé⟩
⟨délimiteur⟩
⟨dégobé⟩
⟨délimiteur⟩
```

`⟨étoile⟩` peut être soit `*` soit vide, dans le premier cas on dit que la macro est étoilée, et dans le second qu'elle est non-étoilée. Si la macro est non-étoilée, alors la condition est une condition à la L^AT_EX, c'est-à-dire une condition telle que la prend en premier argument la macro `\ifthenelse` du paquetage `ifthen`, par exemple ⁷ `{\boolean{true}}` ou `{\equal{⟨chaîne 1⟩}{⟨chaîne 2⟩}}`. Se reporter à la documentation de ce paquetage pour plus ample information. Avec les formes non étoilées il faut donc que le paquetage `ifthen` ait été chargé, ce que vous pouvez faire en passant l'option `ifthen` à `makeglobber`.

6. « `¬C` » signifie « non `C` ».

7. `⟨condition⟩` correspond à `{⟨test⟩}` dans le manuel de `ifthen`.

Avec la forme étoilée, le premier argument doit être une condition à la T_EX, c'est-à-dire :

- soit une condition⁸ `\iftoto` instanciée avec une commande `\newif\iftoto` et pouvant être commutée avec les macros `\totofalse` et `\tototrue`,
- soit une condition utilisant l'un des opérateurs de branchement conditionnel prédéfini dans T_EX, par ex. `\ifmode`, `{\ifnum(a)=}`, `{\ifx\<a>\}`, etc. Notez que dans les cas où `<condition>` consiste en plusieurs unités lexicales il faut⁹ l'écrire entre accolades.

6.2 Macros pour un gobage ou dégobage inconditionnel

Le paquetage fournit également deux macros de plus bas niveau `\makegobbler` et `\makeungobbler` qu'en général l'utilisateur *lambda* n'a pas besoin d'utiliser directement, et qui servent respectivement au gobage et au dégobage inconditionnel d'un bout de code L^AT_EX.

Je décris ci-après ces deux macros parce qu'elles constituent le mécanisme sous-jacent du paquetage, c'est-à-dire que sous le capot, les macros pour le gobage et/ou dégobage conditionnel, `\MGBkeep`, `\MGBdrop`, `\MGBkeepelse`, et `\MGBdropelse`, finissent par se développer, selon la `<condition>` qui leur est passée, en un `\makegobbler` ou un `\makeungobbler`

Le nom du paquetage provient d'ailleurs de ce que le délimiteur `\<délimiteur>` peut être utilisé avec `\makegobbler` pour supprimer tout ce qui suit jusqu'au prochain `\<délimiteur>`, comme ci-dessous, où pour l'exemple `\<délimiteur>` est nommé `\LaMacroQuiGobeTout` :

```
\makegobbler\LaMacroQuiGobeTout
Bla Bla bla, tout le texte ici est supprimé.
\LaMacroQuiGobeTout
```

On appelle cette suppression *gobage*, et on dit que `\LaMacroQuiGobeTout` est un gobeur, la macro `\makegobbler` sert donc à fabriquer un gobeur du nom de `\LaMacroQuiGobeTout` qui va tout gober jusqu'à rencontrer une marque de fin également nommé `\LaMacroQuiGobeTout`, sauf que du point de vue *catcodique* la première occurrence de `\LaMacroQuiGobeTout` est une séquence de contrôle, alors que la seconde est une `string`, c'est-à-dire une suite de caractères de catcode `other`.

Si on avait utilisé `\makeungobbler` à la place de `\makegobbler` alors le texte aurait été inséré au lieu d'être supprimé :

```
\makeungobbler\LaMacroQuiGobeTout
Bla Bla bla, tout le texte ici est inséré.
\LaMacroQuiGobeTout
```

Et donc dans l'exemple ci-dessus on parle de *dégobage* et on dit qu'on a défini la macro `\LaMacroQuiGobeTout` comme un dégobeur, c'est-à-dire qu'elle conserve

8. Toto a un cousin anglais complètement foo.

9. Ceci n'a rien à voir avec le paquetage `makeglobber`, c'est une règle générale de T_EX, lorsqu'un argument de macro consiste en une séquence de plusieurs unités lexicales on le met entre accolades pour le passer d'un seul tenant à la macro. Lorsque il y a une seule unité lexicale, alors les accolades sont optionnelles, on peut passer `{\iftoto}` aussi bien que `\iftoto`.

tout ce qui suit jusqu'à la prochaine occurrence de `\LaMacroQuiGobeTout` qui sert de marque de fin, et que cette marque de fin est supprimée.

Il est à noter que la définition de `\<délimiteur>` comme une macro de gobage/dégobage est transitoire. Une fois la marque de fin consommée, `\<délimiteur>` reprend sa valeur initiale si elle en avait une, ou sinon redevient indéfini.

7 Remerciements

Un grand merci à [quark67](#) pour sa relecture attentive du manuel.

8 Le code

8.1 Options du paquetage

Les déclarations habituelles pour que L^AT_EX connaisse le paquetage.

```
1 \NeedsTeXFormat{LaTeX2e}[2020/10/01]
2 \ProvidesPackage{makegobbler}
3   [2026-02-15 v0.3.2 %
4   Gobbling macros maker]
```

`\MGB@process@ifthen@option` Une option juste pour dire de charger le paquetage `ifthen` qui permet d'avoir des conditions à la L^AT_EX. La macro `\MGB@process@ifthen@option` sert juste à faire le `\RequirePackage{ifthen}` après le `\ProcessOptions`, c'est-à-dire qu'on la développe à cet endroit là. Elle s'autodétruit assitôt sa tâche accomplie, vu qu'on n'en a plus besoin par la suite.

```
5 \DeclareOption{ifthen}{\def\MGB@process@ifthen@option
6   {\RequirePackage{ifthen}\let\MGB@process@ifthen@option\@undefined}}
```

Si l'option `ifthen` n'est pas passée, `\MGB@process@ifthen@option` ne fait rien mis à part s'autodétruire.

```
7 \def\MGB@process@ifthen@option{\let\MGB@process@ifthen@option\@undefined}
```

N'utilisez pas l'option `tense`, sauf si vous avez lu et compris le code qui l'utilise, et si vous recherchez un substitut à la caféine.

```
8 \newif\ifMGB@tense
9 \DeclareOption{tense}{\MGB@tensetrue}
```

```
10 \ProcessOptions*
11 \MGB@process@ifthen@option
```

8.2 Interface utilisateur

8.2.1 Macros de gobage/dégobage conditionnel

`\MGBkeep` Définition des macros d'interface utilisateur, les quatre définitions ont sensiblement la même gueule parce qu'en premier lieu on ne fait que sélectionner si `\MGBdrop` on passe une condition à la T_EX en étoilant la macro, ou bien à la L^AT_EX en ne `\MGBkeepelse` l'étoilant pas. `\MGBdropelse`

```
12 \def\MGBkeep{\MGB@stdhooks\@ifstar{\MGBkeep@tex}{\MGBkeep@latex}}
13 \def\MGBdrop{\MGB@stdhooks\@ifstar{\MGBdrop@tex}{\MGBdrop@latex}}
14 \def\MGBkeepelse{\MGB@elsehooks\@ifstar{\MGBkeep@tex}{\MGBkeep@latex}}
15 \def\MGBdropelse{\MGB@elsehooks\@ifstar{\MGBdrop@tex}{\MGBdrop@latex}}
```

8.2.2 Macros de gobage/dégobage inconditionnel

Les quatre macros de gobage/dégobage conditionnel sous le capot finissent tôt ou tard par développer l'une des deux macros de plus bas niveau `\MGB@g` dans le cas d'un gobage, ou `\MGB@u` dans le cas d'un dégobage. Ces deux dernières sont également accessible depuis l'interface utilisateur, via respectivement les macros `\makeglobber` et `\makeunglobber`, en effet `\makeglobber` pourrait être utile par exemple en DocTeX pour gober la section pilote du document `.dtx`.

`\makeglobber` Les macros `\makeglobber` et `\makeunglobber` délèguent le traitement proprement dit de gobage/dégobage à respectivement `\MGB@g` et `\MGB@u` avec le préliminaire de développer en premier lieu `\MGB@stdhooks`, ce dont on verra l'utilité en § 8.3 :

```
16 \def\makeglobber{\MGB@stdhooks\MGB@g}%
17 \def\makeunglobber{\MGB@stdhooks\MGB@u}%
```

`\MGB@u` La macro `\MGB@u` a juste pour effet qu'elle gobe le délimiteur, et le définit de sorte qu'il se redéfinisse à sa valeur initiale. Donc ce qui suit entre les deux délimiteurs n'est pas gobé.

```
18 \def\MGB@u#1{\MGB@save@mark#1\def#1{\MGB@ungobblerhook#1}}%
```

`\MGB@g` La macro `\MGB@g` prend en argument une séquence de contrôle `\langle delimitueur \rangle`, et gobe tout ce qui suit jusqu'à rencontrer la string `\langle delimitueur \rangle` :

```
19 \def\MGB@g#1{%
20   \MGB@save@mark#1%
21   \begingroup
22   \expandafter\MGB@\expandafter
23   {\string#1}#1\MGB@set@catcoderegime#1}%
```

8.3 Gestion des crochets de fin de traitement

Notez que les deux macros `\MGBkeepelse` et `\MGBdropelse` ne se distinguent respectivement de `\MGBkeep` et `\MGBdrop` que par la macro `\MGB@elsehooks` au lieu de `\MGB@stdhooks`. En fait, le cœur du réacteur c'est les macros `\MGB@g` et `\MGB@u`. Chacune des ces macros développe en fin de traitement respectivement une macro crochet `\MGB@gobblerhook`, resp. `\MGB@ungobblerhook`, dont la valeur par défaut `\MGB@stdhook` consiste à finir le traitement.

`\MGB@stdhooks` La macro `\MGB@stdhooks` règle les définitions de ces deux macros crochets à leur valeur par défaut, après avoir empilé la valeur courante :

```
24 \def\MGB@stdhooks{%
```

Donc en premier lieu, avant de faire ce réglage, elle empile la valeur courante des macros crochets, ceci permettant de cascader plusieurs niveaux de gobage/dégobage avec leur `\langle delimitueur \rangle` respectifs :

```
25   \MGB@pushhooks
```

puis elle fait le réglage proprement dit aux valeurs par défaut des crochets en développant la macro `\MGB@resethooks` :

```
26   \MGB@resethooks
27 }
```

`\MGB@resethooks` La macro `\MGB@resethooks` règle les définitions des deux macros crochets à leur valeur par défaut, c'est-à-dire celle qui termine le traitement, ou en d'autres termes celle qui correspond au fait que le `\langle delimitueur \rangle` est le dernier :

```
28 \def\MGB@resethooks{%
29   \let\MGB@gobblerhook\MGB@stdhook
30   \let\MGB@ungobblerhook\MGB@stdhook
```

```

31 }
\MGB@elsehooks La macro \MGB@elsehooks ne fait que changer les définitions de ces macros cro-
chets pour qu'un dégobage soit fait à la suite d'un gobage, ou resp. un gobage à
la suite d'un dégobage, ceci permettant de traiter la branche else.
32 \def\MGB@elsehooks{%
Comme avec \MGB@stdhooks, la première chose qu'on fait c'est d'empiler la valeur
courante des macros crochets, de sorte à pouvoir les dépiler en fin de gobage.
33 \MGB@pushhooks
Donc on règle le crochet de gobage...
34 \def\MGB@gobblerhook##1{%
pour tout d'abord écraser les valeurs courantes des crochets de gobage/dégobage
par les valeurs par défaut, sans pour autant changer l'état de la pile...
35 \MGB@resethooks
et ensuite faire un dégobage :
36 \MGB@u##1}%
et pour le crochet de dégobage c'est l'inverse...
37 \def\MGB@ungobblerhook##1{%
comme précédemment on règle les macros crochets aux valeurs par défaut sans
changer l'état de la pile...
38 \MGB@resethooks
mais ensuite on enchaîne sur un gobage au lieu d'un dégobage :
39 \MGB@g##1}%
40 }
\MGB@stdhook La macro \MGB@stdhook est la valeur par défaut des macros crochets, elle finit le
traitement de gobage/dégobage : c'est-à-dire qu'elle restaure la macro \langle délimiteur \rangle
à sa valeur initiale 10 :
41 \def\MGB@stdhook#1{%
42 \MGB@pophooks
43 {\def\@tempa##1{\let#1##1\let##1\@undefined}%
44 \expandafter\expandafter\expandafter}\expandafter
45 \@tempa\cename MGB@save\string#1\endcsname
46 }%
\MGB@save@mark La restauration de \langle délimiteur \rangle se fait à partir d'une sauvegarde faite dans la
variable 11 \MGB@save\langle délimiteur \rangle. La macro \MGB@save@mark sert à effectuer la
sauvegarde en début de traitement :
47 \def\MGB@save@mark#1{%
48 \expandafter\let\cename MGB@save\string#1\endcsname#1}%

```

8.4 Gestion de la pile de macros crochets

```

\MGB@hook@sd La macro \MGB@hook@sd est un compteur qui donne la profondeur dans la pile
de macros crochets (sd = « stack depth ») :
49 \newcount\MGB@hook@sd
\MGB@pushhooks La macro \MGB@pushhooks sert à empiler la valeur courante des macros crochets :
50 \def\MGB@pushhooks{%
51 \expandafter\let\cename MGB@ghookstack\the\MGB@hook@sd\endcsname

```

10. Sa valeur avant gobage (\MGB@g) ou dégobage (\MGB@u).

11. La deuxième contr'oblique est une lettre.

```

52   \MGB@gobblerhook
53   \expandafter\let\csname MGB@uhookstack\the\MGB@hook@sd\endcsname
54   \MGB@ungobblerhook
55   \advance\MGB@hook@sd@one
56 }

```

`\MGB@pophooks` La macro `\MGB@pophooks` sert à dépiler les valeurs des macros crochets de sorte à restituer les valeurs avant le dernier empilement :

```

57 \def\MGB@pophooks{%
58   \advance\MGB@hook@sd@mone
59   \expandafter\let\expandafter\MGB@gobblerhook
60   \csname MGB@ghookstack\the\MGB@hook@sd\endcsname
61   \expandafter\let\expandafter\MGB@ungobblerhook
62   \csname MGB@uhookstack\the\MGB@hook@sd\endcsname
63 }

```

Touche finale de la gestion de la pile, les macros crochets de gobage/dégobage sont initialisées pour produire une erreur indiquant un épuisement de la pile¹². En effet, si on ne fait pas cela, et que par malheur survient une situation où se produit un dépilement de plus que d’empilements, alors on aura un message d’erreur abscons indiquant soit que `\MGB@gobblerhook` n’est pas définie, soit que c’est `\MGB@ungobblerhook` qui n’est pas définie. Avec les initialisations suivantes, dans une telle situation au lieu de cela on a un message d’erreur beaucoup plus explicite :

```

64 \def\MGB@gobblerhook#1{%
65   \PackageError{makegobbler}{\'Epuisement pile macros crochets}{%
66     Une erreur a produit un d\'epilage de plus que d\'empilage.%
67   }
68 }
69 \let\MGB@ungobblerhook\MGB@gobblerhook

```

8.5 Gobage/dégobage dans le cas d’une condition à la T_EX

`\MGBkeep@tex` On commence par le gros morceau, le cas des conditions à la T_EX. En fait on `\MGBdrop@tex` veut grosso modo que `\MGBkeep*\ifcondition` se développe en :

```

\ifcondition
<couic>
\expandafter\MGB@u
\else
\expandafter\MGB@g
\fi

```

Dans le cas de `\MGBdrop` c’est juste l’ordre de `\MGB@u` et `\MGB@g` qui est inversé. Comme c’est un peu coton de mettre en clair un `\else` ou un `\fi` dans un corps de macro sans qu’il n’y ait `\ifcondition` également en clair, on va passer le `\else` et le `\fi` via des registres toks pour qu’ils ne soient pas en clair. Pour faire ça on se place au sein d’un groupe, et le boulot sera fait via une macro `\@tempa` développée en after-group.

```

70 \beginingroup

```

Au fait, `\ifcondition` ça peut être n’importe quelle condition à la T_EX, par exemple `\ifnum<a>=`_□. Du coup pour éviter qu’au cas où l’utilisateur oublierait

¹². « stack underflow » en langue anglaise

l'espace¹³ juste après le $\langle b \rangle$ et où donc la branche vraie serait intempestivement développée, ce qui entrainerait une erreur de type `\else` sans `\if`, on insère en général un `\relax` après le `\ifcondition`, sauf pour les utilisateurs qui aiment vivre dangereusement et ont passé l'option `tense` au paquetage. Donc en général `\toks0` contient `\relax`, sauf si l'option `tense` est passée, auquel cas il est vide.

```
71 \edef\@tempa{\toks0{\ifMGB@tense\else\relax\fi}}\@tempa
\toks1 contient \else, et
72 \toks1\expandafter{\csname else\endcsname}%
\toks2 contient \fi.
```

```
73 \toks2\expandafter{\csname fi\endcsname}%
```

Et maintenant on définit `\MGBkeep@tex`, et `\MGBdrop@tex` dans une macro `\@tempa` développée en after-group.

```
74 \edef\@tempa{%
75   \def\noexpand\MGBkeep@tex##1{##1\the\toks0
76     \noexpand\expandafter\noexpand\MGB@u
77     \the\toks1
78     \noexpand\expandafter\noexpand\MGB@g
79     \the\toks2
80   }%
81   \def\noexpand\MGBdrop@tex##1{##1\the\toks0
82     \noexpand\expandafter\noexpand\MGB@g
83     \the\toks1
84     \noexpand\expandafter\noexpand\MGB@u
85     \the\toks2
86   }%
87 }%
88 \expandafter\endgroup\@tempa
```

8.6 Gobage/dégobage dans le cas d'une condition à la L^AT_EX

`\MGBkeep@latex` Le cas des conditions à la L^AT_EX est beaucoup plus simple à traiter. En fait on `\MGBdrop@latex` veut grosso modo que `\MGBkeep{\langle condition \rangle}` se développe en :

```
\ifthenelse{\langle condition \rangle}{%
\MGB@u}{%
\MGB@g}
```

Pour développer `\MGB@u` si $\langle condition \rangle$ est vraie, ou `\MGB@g` sinon. Et pour `\MGBdrop` c'est l'inverse, ce qu'on obtient en intervertissant `\MGB@u` et `\MGB@g`.

```
89 \def\MGBkeep@latex#1{\ifthenelse{#1}{\MGB@u}{\MGB@g}}
90 \def\MGBdrop@latex#1{\ifthenelse{#1}{\MGB@g}{\MGB@u}}
```

8.7 Macros auxiliaires de `\MGB@g`

`\MGB@set@catcoderegime`

La macro `\MGB@set@catcoderegime` sert à changer le régime de catcode pour que dans le cas du gobage, tout ce qui est gobé soit pris verbatim.

```
91 \def\MGB@set@catcoderegime{%
92   \catcode35=12
93   \catcode123=12
94   \catcode125=12
```

13. Cet espace fait partie du nombre et ne génère pas une espace dans le texte.

```

95 \catcode37=12
96 \catcode92=12
97 }
\MGB@otherize La macro \MGB@otherize change le catcode de chacun des caractères du délimiteur
en other. C'est pour que le délimiteur soit bien reconnu comme marque de fin.
98 \def\MGB@otherize#1{%
99 \def\@tempa{#1}%
100 \ifx\@tempa\@nnil\else
101 \catcode'#1=12
102 \expandafter\MGB@otherize
103 \fi
104 }
\MGB@ La macro \MGB@ est l'auxiliaire de \MGB@g, elle prend deux arguments :
#1 Le délimiteur en tant que string
#2 Le délimiteur en tant que séquence de contrôle
105 \def\MGB@#1#2{\MGB@otherize#1\@nil\long\def#2##1#1{\endgroup
106 \MGB@gobblerhook#2}}%
E finita la comedia !
107 \endinput

```