Les extensions `doc` et `shortvrb`*

Frank Mittelbach†
Gutenberg Universität Mainz

30 avril 2008

Résumé

Cette extension contient les définitions nécessaires pour mettre en page la documentation des fichiers d'extension. Cette extension a été développée à Mainz en collaboration avec le Royal Military College of Science. Cette version est une mise à jour qui documente les divers changements et les nouvelles fonctionnalités de `doc` et intègre les fonctions de newdoc.

Table des matières

Préface à la version 1.7

Cette version de `doc.dtx` présente le changements qui sont apparus depuis la dernière version publiée [5] mais qui étaient présents dans les versions distribuées de `doc.sty` depuis quelque temps. Elle contient également les fonctions — non documentées — de la version distribuée de `newdoc.sty`.

Les changements et ajouts suivants ont modifié l'interface pour l'utilisateur depuis la version publiée [5]. Voir le §2 pour plus de détails.

*Traduction française par le TEXnicien de surface, version 1, du 2008-04-15, de la version v2.1d du 2006/02/02.

†Commentaire supplémentaire préparé au Royal Military College of Science par B. Hamilton Kelly ; traduction anglaise de parties des commentaires originaux en allemand par Andrew Mills ; additions très substantielles, particulièrement de `newdoc`, et documentation des caractéristiques post-v1.5q ajoutées à la version v1.7a par Dave Love (SERC Daresbury Lab). Extraction de l'extension `shortvrb` ajoutée par Joachim Schrod (TU Darmstadt).

Mécanismes du pilote on utilise désormais `\DocInput` dans le fichier pilote pour incorporer éventuellement plusieurs fichiers indépendants de `doc` et il n'est plus nécessaire que `doc` soit la dernière extension invoquée. On a remplacé `\IndexListing` par `\IndexInput` ;

L'indexation est controllée par `\PageIndex` et `\CodelineIndex` : on doit spécifier l'un des deux pour produire un index — il n'y a plus de `\makeindex` dans les paramètres par défaut fournis par `\DocstyleParms` ;

L'environnement `macro` prend maintenant comme argument le nom de la macro *avec* la contreoblique — barre oblique inverse — (*backslash*) ;

texte Verbatim Les sauts de ligne sont interdits désormais à l'intérieur de `\verb` et on fournit les commandes `\MakeShortVerb` et `\DeleteShortVerb` pour créer des raccourcis pour le verbatim ;

`\par` peut désormais être utilisé dans `\DoNotIndex` ;

La somme de contrôle (*checksum*) et la table de caractères ont été ajouté pour permettre le contrôle de l'intégrité d'une distribution ;

`\printindex` est devenu `\PrintIndex` ;

`multicol.sty` n'est plus nécessaire pour utiliser `doc` ou imprimer la documentation — bien que ce soit toujours recommandé ;

Les modules pour 'Docstrip' sont reconnus est mis en page de façon particulière.

Comme on ajoutait des trucs complètement nouveaux, on a profité de l'occasion pour ajouter quelques commentaires dans le code qui se trouvait auparavant dans `newdoc.sty` et dans celui ajouté après la version 1.5k de `doc.sty`. Puisque — comme on l'indique dans les sections concernées — ces commentaires n'ont pas été écrits par Frank Mittelbach mais que le code est de lui il n'est probablement *pas vrai* que dans le cas où « le code et les commentaires sont en désaccord, les deux sont probablement faux » !

Bogues

Cette version contient quelques bogues connus :

- La commande `\DoNotIndex` ne fonctionne pas avec les commandes d'un seul caractère parmi lesquelles la plus notable est `\%` ;
- L'entrée de glossaire « Changements généraux » devrait apparaitre après les noms de macros commençant par un `!` et peut-être un `"` ;
- Si vous avez une ancienne version de makeindex, les entrées longues de `\changes` seront rendues étrangement et vous pourrez obtenir une entête de section mélangée à la première entrées des changements. Essayez de vous procurez une version à jour (cf. p. 7) ;
- Comme le fichier de style de makeindex associé à cette extension gère des spécifications incohérentes d'attribut des ancienne et nouvelle versions de makeindex, ce dernier se plaint toujours de trouver trois *unknown specifier* — spécificateur inconnu — lorsqu'il trie les entrées d'index et de changements.
- Si on utilise `\MakeShortVerb` et `\DeleteShortVerb` avec un argument d'un seul caractère comme p. ex. `{|}` au lieu de `{\|}`, on peut déchainer le chaos.

— Quelques « fonctions » sont présentées ci-dessous. —

- des portegreffes (*hooks*) pour permettre à `\DescribeMacro` et `\DescribeEnv` d'écrire dans un fichier spécial des informations sur les définitions « exportées » de l'extension que ces commandes décrivent. Ces informations pourraient ensuite incluses dans le fichier `.sty` `docstrip`é dans une forme convenable pour être utilisées par un éditeur intelligent lors du complètement (*completion*) de commande, la vérification orthographique, etc. en se basant sur les extensions utilisées dans le document [1]. Il faudrait bien entendu un accord sur ce que serait une forme convenable ;
- Indexation des modules utilisés dans les directives `%<` de `docstrip`. Je n'ai pas d'idée arrêtée sur la façon d'indexer les directives contenant des combinaisons de modules ;
- Produire des information bibliographique sur l'extension ;
- Permettre de désactiver la police spéciale pour p. ex. le prochain bloc protégé.

1. Je pense à ce que fait déjà AucTeX dans Emacs. [Le TdS]

1 Introduction

Les macros TEX décrites ici permettent de placer dans un seul et même fichier définitions et documentations. Cela a pour avantage de permettre une meilleure compréhension d'instructions normalement très compliquées grâce à des commentaires placés au sein même de la définition. De plus, les mises à jour sont facilitées et il suffit de changer un unique fichier. D'un autre côté, de ce fait, les fichiers d'extension sont considérablement plus longs ce qui fait que TEX met plus de temps à les charger. Si cela est un problème, la solution est simple : il suffit de lancer le programme `docstrip.tex` qui supprime presque toutes les lignes commençant par un %.

L'idée d'une documentation intégrée est née avec le développement du programme TEX ; elle a pris forme en Pascal avec le système Web. Les avantages de cette méthode sont faciles à voir — on peut aisèment faire des comparaisons, cf. [2]. Depuis ce développement, des systèmes semblables à Web ont été développés pour d'autres langages de programmation. Mais pour l'un des plus compliqué d'entre eux — TEX — la documentation fut négligée. Le monde TEX parait divisé entre :

- d'une part, une paire de « mages » qui improvisent des lignes d'un code totalement illisible et
- d'autre part, de nombreux utilisateurs ébahis de voir que ce code marche juste comme il le souhaitait. Ou, plutôt, qui se désespère de voir que certaines macros refusent de faire ce qu'ils attendent d'elles.

Je ne pense pas que le système Web soit *la* référence, au contraire c'est un prototype qui suffit pour le développement de programme dans le monde TEX. Il est suffisant mais pas complètement suffisant [2]. Grâce à Web, on a présenté de nouvelles perspectives dans l'art de la programmation ; malheureusement, toutefois, on ne les a pas développées plus avant pour d'autres langages de programmation.

La méthode de documentation de macros TEX que j'introduis ici devrait également être considérée comme une première esquisse. Elle est prévue explicitement pour ne fonctionner qu'avec LATEX. Non parce que mon opinion était que c'était le meilleur point de départ mais parce que de ce point de départ le développement était le plus rapide [3]. Du fait de cette décision de conception, j'ai dû abandonner l'idée de modularité ; c'est certainement un pas en arrière.

Je serai heureux si cet article pouvait lancer une discussion sur la documentation à la TEX. Le seul conseil que je donnerais à ceux qui pensent qu'ils peuvent se passer de documentation serait d'arrêter le temps jusqu'à ce qu'ils aient complétement compris le code source d'$\mathcal{AMS}$-TEX.

1.1 Utiliser l'extension `doc`

De même que n'importe quelle autre extension, il suffit de l'invoquer à l'aide de la commande `\usepackage` placée dans le préambule. L'usage que fait `doc` de `\reversemarginpars` peut le rendre incompatible avec certaines classes de documents.

2 L'interface pour l'utilisateur

2.1 Le fichier pilote

Si on veut documenter un ensemble de macros avec l'extension `doc`, on doit préparer un fichier pilote spécial qui produira le document mis en page. Ce fichier pilote a les caractéristiques suivantes :

`\documentclass[`⟨*options*⟩`]{`⟨*classe-de-document*⟩`}`

`\usepackage{doc}`

⟨*préambule*⟩

`\begin{document}`

2. Je sais que certaines personnes ne voient pas comme moi mais ce produit ne devrait pas être considéré comme le produit fini, au moins en ce qui concerne les applications liées à TEX. Le débat déjà ancien à propos des « fichiers à changements multiples » le démontre assez clairement.
3. Cet argument est mauvais mais on le répète bien trop souvent.

⟨*commandes spéciales « input »*⟩

```
\end{document}
```

La ⟨*classe-de-document*⟩ peut être n'importe laquelle, j'utilise habituellement la classe `article`.

Dans le ⟨*préambule*⟩ on devrait placer des déclarations qui change le comportement de l'extension `doc` comme `\DisableCrossrefs` ou `\OnlyDescription`.

`\DocInput` `\IndexInput`

Enfin la partie ⟨*commandes spéciales « input »*⟩ devrait contenir une ou plusieurs commandes `\DocInput`⟨*nom de fichier*⟩ ou `\IndexInput`⟨*nom de fichier*⟩. On utilise la commande `\DocInput` pour des fichiers préparés avec l'exention `doc` alors qu'on peut utiliser `\IndexInput` avec toutes sortes de fichiers de macros — voir page 9 pour de plus amples détails sur `\IndexInput`. On peut se servir de plusieurs `\DocInput` avec des fichiers qui sont eux-même des fichiers d'extensions autonomes et documentés — par exemple, chacun peut contenir un `\maketitle`.

Par exemple, le fichier pilote de l'extension `doc` elle-même est le texte qui suit entouré par `%<*driver>` et `%</driver>`. Pour produire la documentation il suffit de compiler le fichier `.dtx` avec LaTeX et dans ce cas le code sera exécuté — il chargera la classe de document `ltxdoc`, etc. —, on peut, sinon, l'extraire pour en faire un fichier à part à l'aide du programme `docstrip`. Les numéros de ligne ci-dessous sont ajoutés lors de la composition du document par `doc`. Notons que la classe ltxdoc charge elle-même l'extension `doc`.

```
1 ⟨*driver⟩
2 \documentclass{ltxdoc}
3 \usepackage[papier]{TdSfrtrad}  % francisation
4
5 \EnableCrossrefs
6 %\DisableCrossrefs
7 % Say \DisableCrossrefs if index is ready
8 \CodelineIndex
9 \RecordChanges    % Gather update information
10 \OnlyDescription  % comment out for implementation details
11 \setlength\hfuzz{15pt}  % dont make so many
12 \hbadness=7000          % over and under full box warnings
13 \begin{document}
14    \DocInput{doc-fr.dtx}
15 \end{document}
16 ⟨/driver⟩
```

2.2 Conventions générales

Un fichier TeX prêt à être utilisé avec l'extension `doc` consiste en des « parties de documentation » mélangées de « parties de définition ».

Chaque ligne d'une « partie de documentation » commence par un signe pourcent « `%` » en première colonne. Elle peut contenir toute sorte de commandes TeX ou LaTeX mais le caractère « `%` » ne peut pas être utilisé pour écrire des commentaires. Pour permettre à l'utilisateur de placer des commentaires, on définit plus loin le caractère `^^A` comme caractère de commentaires. On peut introduire également de tels « métacommentaires » simplement en les entourant par `\iffalse` ... `\fi`.

Toutes les autres parties du fichier sont des « parties de définition ». Elles contiennent des fragments des macros décrites dans les « parties de documentation ».

Si on utilise le fichier pour définir de nouvelles macros — c.-à-d. comme un fichier d'extension à utiliser dans la macro `\usepackage` — les « parties de documentation » sont ignorées et les définitions de macros sont recollées même si elles sont données en plusieurs fragments dans plusieurs « parties de définition ».

`macrocode`

Par ailleurs, si la documentation de ces macros doit être produite, les « parties de définition » seront composées verbatim. À cette fin, ces parties sont entourées d'un environnement macrocode. Plus exactement : avant une « partie de définition » doit se trouver une ligne contenant

`%␣␣␣␣\begin{macrocode}`

et, après cette partie, une ligne

`%␣␣␣␣\end{macrocode}`

Il doit y avoir *exactement* quatre espaces entre le % et le `\end{macrocode}` — TEX recherche cette chaine de caractères et pas la macro quand il compose la « partie de définition ».

Dans une « partie de définition » toutes les commandes TEX sont autorisées ; on peut même se servir du caractère pourcent pour supprimer les espaces inopportuns etc.

`macrocode*` Au lieu de l'environnement macrocode on peut aussi utiliser l'environnement macrocode∗ qui produit le même résultat si ce n'est que les espaces apparaissent comme des ␣.

2.3 Décrire l'utilisation de nouvelles macros

`\DescribeMacro` Lorsque l'on décrit une nouvelle macro, on peut utiliser `\DescribeMacro` pour signaler qu'à cet endroit on explique l'utilisation d'une macro particulière. Cette commande prend un argument qui sera imprimé dans la marge et produira une entrée spéciale dans l'index. Par exemple, j'utilise `\DescribeMacro{\DescribeMacro}` pour montrer clairement que c'est ici que j'explique l'utilisation de `\DescribeMacro`.

`\DescribeEnv` On devrait utiliser une macro analogue`\DescribeEnv` pour signaler que l'on explique un environnement LATEX. Cela produira une entrée d'index un peu différente. Ci-dessous j'ai utilisé `\DescribeEnv{verbatim}`.

`verbatim` C'est souvent une bonne idée d'inclure des exemples d'utilisation des nouvelles macros dans le texte. Comme chaque ligne commence par un % en première colonne, l'environnement

`verbatim*` verbatim est légèrement modifié pour supprimer ces caractères [4]. L'environnement verbatim∗

`\verb` est modifié de la même façon. La commande `\verb` est réimplémentée pour produire un rapport d'erreur si son argument contient un saut de ligne. Les environnements verbatim et verbatim∗ compose le texte dans le style défini par `\MacroFont` (§ 2.4).

2.4 Décrire la définition d'une nouvelle macro

`macro` Pour décrire la définition d'une nouvelle macro on utilise l'environnement macro. Il prend un argument : le nom de la nouvelle macro [5]. Cet argument est utilisé également pour imprimer le nom dans la marge et pour produire une entrée d'index. De fait, les entrées d'index pour l'utilisation et la définition sont distinctes, ce qui facilite la consultation de l'index. Cet environnement peut être imbriqué. Dans ce cas les étiquettes dans la marge sont placées l'une sous l'autre. On doit placer du texte — ne serait-ce qu'une `\mbox{}` vide — dans cet environnement avant le `\begin{macrocode}` sinon l'étiquette marginale ne sera pas placée au bon endroit.

`\MacrocodeTopsep` Il existe aussi quatre paramètres de style :

`\MacroTopsep` on utilise `\MacrocodeTopsep` et `\MacroTopsep` pour contrôler l'espace vertical surmontant

`\MacroIndent` les environnements macrocode et macro respectivement ; on utilise `\MacroIndent` pour fixer

`\MacroFont` le retrait des lignes de code ; `\MacroFont` contient une commande de changement de la police et éventuellement du corps utilisés pour composer les lignes de code, les environnements `verbatim`[`*`] et les noms des macros imprimés dans les marges. Si on veut changer leurs valeurs par défaut dans un fichier de classe — comme p. ex. `ltugboat.cls` — on utilisera la commande `\DocstyleParms` décrite ci-dessous. Depuis la version 2.0a, on peut désormais faire le changement directement tant que la redéfinition prend place avant le `\begin{document}`.

2.5 Formater les marges

`\PrintDescribeMacro` Comme signalé précédemment, quelques macros et l'environnement macro imprime

`\PrintDescribeEnv` leur argument dans la marge. Cet effet est réalisé par quatre macros que l'utilisa-

`\PrintMacroName` teur peut (re)-définir [6]. Elles ont pour noms `\PrintDescribeMacro`, `\PrintDescribeEnv`,

`\PrintEnvName` `\PrintMacroName` — appelée par l'environnement macro — et `\PrintEnvName` — appelée par l'environnement environment.

4. Ces macros ont été écrites par Rainer Schöpf [8]. Il a également fourni un nouvel environnementverbatim que l'on peut utiliser à l'intérieur d'autres macros.

5. C'est une nouveauté pour rapport à la conception du style que j'avais décrite dans *TUGboat* 10 n° 1 (jan 1989). Nous avons finalement décidé qu'il serait préférable d'utiliser en tant qu'argument le nom de la macro *avec* la controblique.

6. On peut placer ces définitions modifiées dans un fichier d'extension séparé ou au début du fichier de documentation. Par exemple, si l'on n'aime pas que les noms apparaissent dans les marges mais que l'on veuille un bel index on redéfinira simplement ces macros avec `\let` comme égales à `\@gobble`. L'extension `doc` ne modifiera aucune définitions préalables de ces macros.

2.6 Utiliser un caractère d'échappement spécial

\SpecialEscapechar

Si l'on définit une macro complexe, on a parfois besoin d'introduire un nouveau caractère d'échappement parce que \ a reçu un \catcode spécial. Dans ce cas, on peut utiliser \SpecialEscapechar pour signaler quel caractère est, en fait, utilisé dans le rôle de « \ ». Un tel procédé est nécessaire car l'environnement macrocode et son *alter ego* macrocode∗ produisent des entrées d'index pour chaque occurence d'un nom de macro. Ils seraient très troublés si on ne leur indiquait pas que l'on a changé les \catcodes. L'argument de \SpecialEscapechar est une séquence de contrôle d'une seule lettre, c.-à-d. que l'on a besoin de \| p. ex. pour signaler que « | » est utiliser comme caractère d'échappement. \SpecialEscapechar ne modifie le comportement que du seul environnement macrocode ou macrocode∗ suivant.

Les entrées effectives de l'index seront toutes composées avec \ plutôt qu'avec | mais cela reflète probablement leur utilisation si ce n'est leur définition et, de toute façon, c'est préférable à ne pas avoir d'entrée du tout. Les entrées *peuvent* être formatées de façon appropriées mais le jeu n'en vaut pas la chandelle et l'index qui en résulterait risquerait d'être plus déroutant — et plus long !

2.7 Références croisées des macros utilisées

\DisableCrossrefs
\EnableCrossrefs

Comme mentionné ci-dessus, chaque nom de nouvelle macro utilisée dans un environnement macrocode ou macrocode∗ produira une entrée d'index. De cette façon, on peut aisément retrouver où l'on se sert d'une macro particulière. Comme TEX est considérablement plus lent lorsqu'il doit produire une telle masse d'entrées d'index, on peut désactiver cette fonction en plaçant \DisableCrossrefs dans le fichier pilote. Pour l'activer à nouveau on se sert de \EnableCrossrefs [7].

\DoNotIndex

On fournit toutefois le moyen d'un contrôle plus fin. La macro \DoNotIndex prend pour argument une liste de noms de macros séparés par des virgules. Ces noms n'apparaitront pas dans l'index. On peut utiliser plusieurs macros \DoNotIndex, leurs listes seront concaténées. Dans cet article j'utilise \DoNotIndex avec toutes les macros déjà définies par LATEX.

Les trois déclarations ci-dessus sont locales au groupe courant.

\PageIndex
\CodelineIndex
\theCodelineNo

La création — ou pas — d'un index, à l'aide de la commande \makeindex, est contrôlée en utilisant ou omettant, dans le préambule du fichier pilote, les déclarations suivantes ; si aucune n'est utilisée, l'index n'est pas créé. Avec \PageIndex, toutes les entrées font référence à leur numéro de page ; avec \CodelineIndex, les entrées produites par \DescribeMacro et \DescribeEnv indiquent le numéro de la page mais celles produites par l'environnement macro donne le numéro des lignes de code qui sont numérotées automatiquement [8]. Le style de la numérotation peut être contrôlé en définissant la macro \theCodelineNo. Sa définition par défaut entraine l'utilisation de chiffres arabes de taille scriptsize. Une définition donnée par l'utilisateur ne sera pas écrasée.

\CodelineNumbered

Quand on ne veut pas d'index mais que l'on souhaite toutefois la numérotation des lignes de code, on utilise \CodelineNumbered au lieu de \CodelineIndex. Cela empêche l'inutile production d'un fichier .idx.

2.8 Produire les entrées d'index effectives

Plusieurs des macros ci-dessus mentionnées produisent une sorte ou une autre d'entrée d'index. Ces entrées doivent être triées par un programmen externe — l'implémentation courante suppose que l'on se sert du programme makeindex de Chen [4]. [9]

Mais ce comportement n'est pas incorporé à l'extension : il suffit de redéfinir quelques unes des macros suivantes pour pouvoir utiliser n'importe quel autre programme d'indexation. Toutes les macros qui dépendent de l'installation sont définies de telle sorte qu'elles n'écraseront pas une définition précédente. On peut donc en toute sureté placer les versions modifiées dans un fichier d'extension qui devra être lu avant l'extension doc.

7. En fait, \EnableCrossrefs modifie les choses plus radicalement : tous les éventuels \DisableCrossrefs du source seront ignorés.

8. En fait, la ligne de code indiquée est la première du premier environnement macrocode de l'environnement macro.

9. On pourrait tout aussi bien utiliser le programme xindy pour ce faire. [Le TdS]

Pour permettre à l'utilisateur de changer les caractères spéciaux reconnus par son programme d'indexation, tous les caractères qui ont un sens spécial dans le programme makeindex ont reçu un nom symbolique [10]. Toutefois, les caractères utilisés doivent tous avoir un `\catcode` différent de « lettre (11) ».

`\actualchar` `\quotechar` `\encapchar` `\levelchar`

On utilise `\actualchar` pour séparer la « clé » de l'entrée véritable dans l'index. On se sert de `\quotechar` devant un caractère spécial du programme d'indexation pour lui ôter sa signification spéciale. Avec `\encapchar`, on sépare les informations d'indexation d'une chaine de caractères que makeindex utilise comme une macro TEX pour formater le numéro de page associé à une entrée particulière. On l'utilise dans cette extension pour appliquer les commandes `\main` et `\usage`. Un `\levelchar` supplémentaire s'utilise pour séparer les entrées de niveau `item`, `subitem` et `subsubitem`.

C'est une bonne idée que de rester fidèle à ces noms symboliques même si l'on sait quel programme d'indexation on va utiliser. Cela rendra vos fichiers portables.

`\SpecialMainIndex` `\SpecialMainEnvIndex` `\SpecialIndex` `\SpecialUsageIndex` `\SpecialEnvIndex` `\SortIndex`

Pour produire une entrée principale d'index pour une macro, on peut utiliser la commande `\SpecialMainIndex` [11] Une macro du même genre, nommée `\SpecialMainEnvIndex` est utilisée pour indexer le point de la définition principale d'un environnement [12]. On peut se servir de `\SpecialIndex` pour obtenir une entrée d'indes normal pour un nom de macro [13]. On peut utiliser `\SpecialUsageIndex` et `\SpecialEnvIndex` pour indexer l'utilisation d'une macro ou d'un environnement. On fournit de plus une commande `\SortIndex`. Elle prend deux arguments : la clé de tri et l'entrée effective d'index.

Normalement, toutes ces macros sont utilisées par d'autres ; on en n'aura besoin qu'en cas d'urgence.

`\verbatimchar`

Cela vaut la peine de mentionner une caractéristique : tous les noms de macros de l'index sont composés avec la commande `\verb*`. Aussi on a besoin d'un caractère spécial pour server de délimiteur pour cette commande. Pour permettre de changer de délimiteur on appelle ce caractère de manière indirecte avec la macro `\verbatimchar`. Cette dernière est développée par défaut en `+` mais si son code contient des macros dont les noms comportent des caractères « `+` » — p. ex. quand on utilise `\+` — on obtiendra au final une entrée d'index contenant `\verb+\++` qui sera imprimée « \+ » et non comme « `\+` ». Dans ce cas, on devrait redéfinir `\verbatimchar`, globalement ou localement, pour résoudre ce problème.

`\*`

Nous fournissons aussi une macro `\*`. Elle est destinée aux entrées d'index comme

entrée d'index
    Macros spéciales pour ~

On peut produire une telle entrée avec la ligne :

```
\index{entrée d'index\levelchar Macros spéciales pour \*}
```

`\OldMakeindex`

Les versions de makeindex antérieures à la 2.9 contiennent quelques bogues qui affectent `doc`. L'un d'eux, lié au caractère `%`, ne peut être contourné d'une manière convenant autant aux versions boguées qu'aux autres. Si l'on possède une ancienne version, il faut invoquer `\OldMakeindex` dans le fichier de l'extension ou le fichier pilote pour s'épargner les ennuis avec les entrées d'index comme `\%`, quoique l'on souhaite en général ne pas indexer `\%`. Le mieux est de se procurer une version à jour de makeindex dans l'un des dépots TEX.

2.9 Préparer les entrées d'index

Après la première passe de composition à travers le fichier `.dtx`, on doit trier les entrées d'index écrites dans le fichier `.idx` à l'aide de makeindex ou autre. Il faut utiliser un style convenable pour makeindex, style précisé grace à l'option `-s`. `doc` est distribué avec un tel style, `gind.ist`.

`\PrintIndex`

Pour lire l'index trié et l'imprimé, il suffit de placer la commande `\PrintIndex`, précédée de `%` — ce qui assure qu'elle sera exécutée lors de la composition de la documentation —, comme dernière commande du fichier de l'extension. On peut la faire précéder par toute

10. Je ne sais pas s'il existe un programme qui a besoin de plus de caractères de commande mais j'espère que non.
11. Cette macro est appelée par l'environnement macro.
12. Cette macro est appelée par l'environnement environment.
13. Cette macro est appelée de l'intérieur de l'environnement macrocode lorsque un nom de macro se présente.

commande de bibliographie utile pour les références. Sinon, on peut placer de tels appels parmi les arguments de la macro `\StopEventually`, auquel cas on placera une commande `\Finale` à la fin du fichier.

`theindex` Contrairement à ce qui se passe normalement en LaTeX, l'index est composé sur trois colonnes. Ce comportement est contrôlé par le compteur LaTeX IndexColumns et l'on peut en changer avec une déclaration `\setcounter`. De plus, pour ne pas commencer inutilement sur `\IndexMin` une nouvelle page, l'environnement theindex est redéfini. Lorsque l'environnement theindex commence, il mesure l'espace restant sur la page courante. Si cet espace est plus grand que `\IndexMin`, l'index commence sur la page, sinon on appelle `\newpage`.

Alors apparait une courte introduction sur la signification de plusieurs sortes d'entrées d'index — toujours sur une colonne. Ensuite les entrées effectives de l'index sont composées `\IndexPrologue` sur plusieurs colonnes. On peut changer le prologue à l'aide de la macro `\IndexPrologue`. En fait l'entête de la section est également produit de cette façon, on aura donc intérêt à écrire quelque chose comme :

```
\IndexPrologue{\section*{Index}
               Les entrées d'index soulignées ...}
```

Quand l'environnement theindex est fini la dernière page est reformatée pour équilibrer les colonnes. Cela améliore la mise en page et permet au prochain article de commencer sur `\IndexParms` cette même page. La présentation des colonnes de l'index — valeurs de `\columnssep` etc. — est contrôlée par la macro `\IndexParms`. Elle fixe les valeurs suivantes :

`\parindent`	= 0.0pt	`\columnsep`	= 15.0pt
`\parskip`	= 0.0pt plus 1.0pt	`\rightskip`	= 15.0pt
`\mathsurround`	= 0.0pt	`\parfillskip`	= −15.0pt

`\@idxitem` Elle définit de plus `\@idxitem` — qui sera utilisée quand une commande `\item` est rencontrée — et sélectionne la taille `\small`. Si on veut changer une quelconque de ces valeurs il faut les rédefinir toutes.

`\main` Les numéros de page des entrées principales sont encapsulés dans la macro `\main`, qui `\usage` souligne son argument, et les nombres renvoyant à une description de l'utilisation sont encapsulés dans la macro `\usage`, qui les compose en *italique*. Comme d'habitude l'utilisateur peut redéfinir ces commandes.

2.10 Changer les valeurs par défaut des paramètres de style

`\DocstyleParms` Si l'on veut remplacer les réglages par défaut de `doc`, on peut, pour ce faire, soit placer ses propres déclarations dans un fichier pilote — c.-à-d. après que `doc.sty` est lu — soit utiliser un fichier d'extension séparée. Dans le premier cas, on peut définir la macro `\DocstyleParms` pour qu'elle contienne tous les réglages et assignations. Cette approche indirecte est nécessaire si l'extension personnelle doit être lue avant `doc.sty`, alors que certains registres n'ont pas encore été alloués. Sa définition par défaut est nulle.

L'extension `doc` affecte actuellement les registres suivants :

`\IndexMin`	= 80.0pt	`\MacroTopsep`	= 7.0pt plus 2.0pt minus 2.0pt
`\marginparwidth`	= 126.0pt	`\MacroIndent`	= 10.66672pt
`\marginparpush`	= 0.0pt	`\MacrocodeTopsep`	= 3.0pt plus 1.2pt minus 1.0pt
`\tolerance`	= 1000		

2.11 Insérer de courtes citations verbatim

`\MakeShortVerb` Taper continuellement `\verb|...|` lorsque l'on cite un morceau verbatim — comme un `\MakeShortVerb*` nom de macro — dans le texte est fastidieux, c'est pourquoi on fournit un mécanisme abrégé. `\DeleteShortVerb` On choisit un caractère ⟨*c*⟩ — normalement un de ceux dont le catcode est « autre » à moins d'avoir de très sérieuses raisons de faire autrement — dont on pense qu'on ne l'emploiera pas dans le texte, ou, du moins, pas souvent. (J'aime bien `"`, mais on peut préférer `|` si le caractère `"` est activé pour créer les *umlauts* p. ex.) Alors après avoir écrit `\MakeShortVerb{\`⟨*c*⟩`}`, on peut utiliser ⟨*c*⟩⟨*text*⟩⟨*c*⟩ comme équivalent de `\verb`⟨*c*⟩⟨*text*⟩⟨*c*⟩ ; de manière analogue la forme étoilée `\MakeShortVerb*{\`⟨*c*⟩`}` donne l'équivalent de `\verb*`⟨*c*⟩⟨*text*⟩⟨*c*⟩. On utilisera

`\DeleteShortVerb{\⟨c⟩}` si l'on veut, par la suite, redonner à ⟨*c*⟩ sa signification première — on peut le redéfinir comme abréviation de verbatim plus tard. La commande « verbatim court » réalise des changements globaux. Le `\verb` abrégé ne peut pas apparaitre dans l'argument d'une autre commande exactement comme `\verb`. Toutefois, on peut utiliser librement le caractère de « verbatim court » dans un environnement verbatim ou macrocode sans effet néfaste. Une commande `\DeleteShortVerb` est ignorée silencieusement si son argument ne représente pas un caractère de verbatim court. Les deux commandes émettent un message pour signaler le changement de signification du caractère.

Nous vous prions de vous souvenir que la commande `\verb` ne peut pas être utilisée dans l'argument d'une autre commande. Aussi les caractères d'abréviation de la commande `\verb` ne peuvent pas non plus y être utilisés.

Cette fonction est disponible, toute seule, dans l'extension `shortvrb`.

2.12 Bricoles supplémentaires

Nous fournissons des macros pour les logos WEB, $\mathcal{AMS}$-TEX, BIBTEX,SLITEX et PLAIN TEX. Il suffit de taper `\Web`, `\AmSTeX`, `\BibTeX`, `\SliTeX` ou `\PlainTeX`, respectivement. LATEX et TEX sont déjà définis dans `latex.tex`.

`\meta` Une autre macro utile est `\meta` qui prend un argument et produit quelque comme ⟨*paramètre de dimension*⟩.

`\OnlyDescription` `\StopEventually` On peut utiliser la déclaration `\OnlyDescription` dans le fichier pilote pour supprimer la dernière partie du document — qui montre vraisemblablement le code. Afin que cela fonctionne, on placera la commande `\StopEventually` à l'endroit convenable dans le fichier. Cette macro prend un argument dans lequel on place tout ce que l'on veut voir imprimer en fin de document s'il s'arrête là — par exemple, une bibliographie, que l'on imprime généralement à la toute fin. Lorsque la déclaration `\OnlyDescription` manque, la macro `\Finale` `\StopEventually` enregistre son argument dans une macro appelée `\Finale` que l'on peut utiliser ensuite pour récupérer des choses — généralement à la toute fin. Une telle combine rend inutile les changements faits à deux endroits.

Ainsi on peut utiliser cette fonction pour produire un guide local pour les utilisateurs de TEX qui ne décrit que l'utilisation des macros — la plupart des utilisateurs ne seront pas intéressés par la définition des macros de toute façon. Pour la même raison la commande `\maketitle` `\maketitle` est changée quelque peu pour permettre plusieurs titres dans le même document. `\ps@titlepage` Ainsi on peut créer un fichier pilote qui lit plusieurs articles en une seule fois. Pour éviter les pagestyle (style de page) indésirables sur la page de titre, la commande `\maketitle` émet une déclaration `\thispagestyle{titlepage}` qui produit une page de style banal (*plain*) si le style de page de titre, titlepage, n'est pas défini. Cela permet à des fichiers de classes comme ltugboat.cls de définir leur propre style de page de titre.

`\AlsoImplementation` Par défaut toute la documentation est composée. Toutefois, on peut choisir explicitement cette action par défaut à l'aide de la déclaration `\AlsoImplementation`. Elle annule toute déclaration `\OnlyDescription` préalable. La distribution de LATEX 2$_\varepsilon$, par exemple, est documentée avec la classe `ltxdoc` qui permet l'utilisation du fichier de configuration `ltxdoc.cfg`. Dans ce fichier on peut donc ajouter :

```
\AtBeginDocument{\AlsoImplementation}
```

afin de s'assurer que tous les documents montreront la partie de code.

`\IndexInput` Enfin, j'ai défini une macro `\IndexInput` qui prend un nom de fichier en argument et produit le listage verbatim de ce fichier en indexant les commandes au passage. Cela peut être utile si l'on veut comprendre quelque chose à des macros sans documentation suffisante. J'ai utilisé cette fonction pour faire des références croisées dans `latex.tex` et j'ai obtenu une copie verbatim contenant quelques 15 pages d'index [14].

`\changes` Pour maintenir un historique des changements d'un fichier, dans le fichier lui-même, on peut placer la commande `\changes` dans la partie descriptive du code modifié. Elle prend trois arguments, ainsi :

`\changes{`⟨*version*⟩`}{`⟨*date*⟩`}{`⟨*texte*⟩`}`

14. Cela prit pas mal de temps et le fichier `.idx` obtenu était plus long que le fichier `.dvi`. En fait, trop long pour être manipulé directement par le programme makeindex — sur notre MicroVAX — mais le résultat en valait la peine.

On peut utiliser ces changements pour produire une fichier auxiliaire — on utilise le mécanisme de `\glossary` de LaTeX dans ce but — qui peut être imprimé après avoir été formaté de manière idoine. La macro `\changes` crée l'entrée imprimée d'un tel historique des changements. Comme les anciennes versions [15] de makeindex limitent la taille de tels champs à 64 caractères, on prendra soit de ne pas dépasser cette limite quand on décrit un changement. L'entrée effective est composée de la ⟨*version*⟩, le caractère `\actualchar`, le nom de la macro concernée par le changement, un deux-points, le caractère `\levelchar` et, enfin, le ⟨*texte*⟩. Le résultat est une entrée de glossaire pour la ⟨*version*⟩, avec le nom de la macro concernée comme sous-entrée. En dehors de l'environnement `macro`, le texte `\generalname` est utilisé au lieu du nom de macro. Quand on fait référence à des macros dans la description d'une modification, on utilise conventionnellement `\cs{`⟨*macroname*⟩`}` plutôt que de chercher à les formater correctement et d'utiliser des caractères précieux dans l'entrée avec une version ancienne de makeindex.

`\RecordChanges` `\PrintChanges`

Pour faire écrire les informations sur les modifications, il suffit d'inclure `\RecordChanges` dans le fichier pilote. Pour lire et imprimer l'historique des changements — sur deux colonnes —, il suffit de placer la commande `\PrintChanges` — précédée d'un `%` et donc exécutée lors de la passe de documentation du fichier — dans le fichier de l'extension. Sinon, cette commande peut être un des arguments de `\StopEventually` quoique un historique des changements ne soit pas nécessaire si on n'imprime que la description. La commande suppose que le fichier `.glo` a été compilé par makeindex ou un autre programme pour créer un fichier `.gls` convenablement trié. On a besoin d'un fichier de style spécial pour makeindex ; on ne fournit un convenable avec `doc`, appelé `gglo.ist`. Les macros `\GlossaryMin`, `\GlossaryPrologue` et `\GlossaryParms` sont analogues aux macros `\Index`... respectives. *Le mécanisme de « glossaire » à la LaTeX est utilisé pour écrire les entrées de l'historique des changements.*

`\GlossaryMin` `\GlossaryPrologue` `\GlossaryParms`

`\CharacterTable` `\CheckSum`

Pour pallier certaines difficultés qui peuvent apparaitre lorsque l'on envoie des fichiers sur un réseau, nous avons développés deux macros qui devrait détecter les fichiers corrompus. Si l'on place les lignes suivantes

```
%%\CharacterTable
%% {Upper-case    \A\B\C\D\E\F\G\H\I\J\K\L\M\N\O\P\Q\R\S\T\U\V\W\X\Y\Z
%%  Lower-case    \a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z
%%  Digits        \0\1\2\3\4\5\6\7\8\9
%%  Exclamation   \!     Double quote  \"     Hash (number) \#
%%  Dollar        \$     Percent       \%     Ampersand     \&
%%  Acute accent  \'     Left paren    \(     Right paren   \)
%%  Asterisk      \*     Plus          \+     Comma         \,
%%  Minus         \-     Point         \.     Solidus       \/
%%  Colon         \:     Semicolon     \;     Less than     \<
%%  Equals        \=     Greater than  \>     Question mark \?
%%  Commercial at \@     Left bracket  \[     Backslash     \\
%%  Right bracket \]     Circumflex    \^     Underscore    \_
%%  Grave accent  \`     Left brace    \{     Vertical bar  \|
%%  Right brace   \}     Tilde         \~}
%%
```

au début du fichier alors les défauts de recodage des caractères seront détectés, si tant est, bien entendu, qu'on utilise l'extension `doc` avec une table correcte. Le signe de pourcentage [16] qui commence les lignes devrait être utilisé car seul l'extension `doc` devrait regarder cette commande.

L'envoi de fichier par courriel peut créer un autre problème : une troncation intempestive. Pour détecter ce type d'erreurs, nous fournissons la macro `\CheckSum`. La somme de contrôle du fichier est simplement le nombre de controbliques présentes dans le code c.-à-d. toutes les lignes contenues dans un environnement macrocode. Mais il ne faut pas avoir peur ! On n'a pas besoin de compter les lignes de code soi-même, l'extension `doc` le fait. Il suffit d'utiliser les commandes `\StopEventually` — qui déclenche la recherche des controbliques — et `\Finale`. Cette dernière dit alors que le fichier n'a pas de somme de contrôle — en précisant le nombre correct — ou qu'il a une somme de contrôle incorrecte — et cette fois donne et la somme

15. Celles qui précèdent la 2.6.
16. Il y a deux signes de pourcentage par ligne. Cela entraine que ces lignes ne sont pas retirées par le programme `docstrip.tex`.

correcte et la somme incorrecte. Il suffit alors de retourner au début du fichier et d'insérer la ligne

```
%% \CheckSum{⟨nombre⟩}
```

et c'est fini ! Si l'on fait précéder cette commande d'un seul signe %, cette ligne sera absente des versions docstrip-ées du fichier. C'est ce que l'on devrait faire toutes les fois que l'on utilise du code conditionnel — voir la documentation de docstrip — puisque, alors, la somme de contrôle ne reflétera pas le nombre de controbliques des versions décapées [17].

\bslash

De temps à autres, il est nécessaire d'imprimer un \ sans pouvoir utiliser la commande \verb car les \catcodes des symboles sont déjà fermement établis. Dans ce cas, on peut utiliser la commande \bslash, en supposant, bien entendu, que la fonte courante contient bien une controblique comme glyphe. On notera que cette définition de \bslash est développable ; elle introduit un $\backslash_{12}$. Cela signifie que l'on doit la protéger — par \protect — si on l'utilise dans un « argument mouvant ».

\MakePrivateLetters

Si l'on change le catcode d'autre chose que @ en celui d'une « lettre », l'on devrait redéfinir \MakePrivateLetters afin que les caractères « lettres » appropriés bénéficient de l'indexation. La définition par défaut est simplement \makeatletter.

\DontCheckModules
\CheckModules
\Module
\AltMacroFont

Les directives de « module » du système de docstrip [6] sont, normalement reconnues et appellent un formatage spécial. On peut activer et désactiver ce mécanisme dans le fichier .dtx ou dans le fichier pilote à l'aide de \CheckModules et \DontCheckModules. Si le contrôle des modules est activé — c'est l'état par défaut — alors le code contenu dans la portée des directives est composé comme le définit le porte-greffe \AltMacroFont, lequel donne une *petite mécane*[18] *italique* par défaut dans le *nouveau* schéma de sélection des fontes *NFSS* mais simplement une petite mécane dans l'ancien dans lequel une fonte comme la mécane italique n'est pas portable ; il faudra redéfinir ce mécanisme si l'on ne dispose pas d'une mécane italique. Du code est dans une portée de ce type s'il est situé sur une ligne commençant par %< ou entre deux lignes commençant, l'une par %<*⟨*liste de noms*⟩> et l'autre par %</⟨*liste de noms*⟩>. La directive est formatée par la macro \Module dont l'unique argument est le texte de la directive, texte situé entre les parenthèses angulaires mais sans les inclure ; on peut redéfinir cette macro dans le fichier pilote ou le fichier d'extension et, par défaut, elle produit quelque chose comme ⟨+foo | bar⟩ sans espace derrière.

StandardModuleDepth

Parfois — comme dans ce fichier — tout le code est entouré par des modules pour produire plusieurs fichiers à partir d'une source unique. Dans ce cas, il est clairement inutile de formater toutes les lignes de code dans une \AltMacroFont spéciale. C'est pour cette raison que l'on fournit un compteur StandardModuleDepth qui définit le niveau d'imbrication des modules qui doivent encore être composé dans la \MacroFont plutôt que dans la \AltMacroFont. La valeur par défaut est 0, pour cette documentation on l'a fixé à

```
\setcounter{StandardModuleDepth}{1}
```

au début du fichier.

2.13 Résumé de l'utilisation élémentaire

En résumé, la structure élémentaire d'un fichier .dtx sans raffinements ressemble à ceci :

```
% ⟨gauffre⟩...
  ...
% \DescribeMacro{\fred}
% ⟨description de l'utilisation de fred⟩
  ...
% \StopEventually{⟨code final⟩}
  ...
% \begin{macro}{\fred}
% ⟨commentaire sur la macro fred⟩
%␣␣␣␣\begin{macrocode}
⟨code de la macro fred⟩
%␣␣␣␣\end{macrocode}
```

17. Je prends « décapé » pour traduire *stripped* ce qui me semble un peu plus clair que le simple « nu ». [Le TdS]

18. Une mécane est une fonte appelée *type-writer*. [Le TdS]

```
% \end{macro}
  ...
% \Finale \PrintIndex \PrintChanges
```

Pour des exemples de l'utilisation de la plupart — si ce n'est toutes — les fonctions décrites ci-dessus, on consultera le source `doc.dtx` lui-même.

2.14 Remerciements

Je voudrais remercier tout le monde à Mayence et au *Royal Military College of Science* pour l'aide que j'ai reçue en menant à terme ce projet ; spécialement Brian et Rainer qui ont fait progresser le tout grâce à leurs suggestions, réparations de bogues, etc.

Un très grand merci à David Love qui a mis à jour, une fois encore, la documentation, après que j'ai négligé ce fichier pendant plus de deux ans. Ce fut très certainement un travail pénible car de nombreuses fonctions ajoutées à `doc.dtx` après sa publication dans *TUGboat* n'avaient jamais été décrites correctement. Outre ce magnifique travail, il a gentiment fourni du code supplémentaire — comme, par exemple, le formatage des modules « docstrip » — dont je pense que tous les utilisateurs de `doc.dtx` lui seront reconnaissants.

Références

[1] G. A. Bürger. Wunderbare Reisen zu Wasser und zu Lande, Feldzüge und lustige Abenteuer des Freyherrn v. Münchhausen. London, 1786 & 1788.

[2] D. E. Knuth. Literate Programming. Computer Journal, Vol. 27, *pp.* 97–111, May 1984.

[3] D. E. Knuth. Computers & Typesetting (The TEXbook). Addison-Wesley, Vol. A, 1986.

[4] L. Lamport. MakeIndex : An Index Processor for LATEX. 17 February 1987. (Taken from the file `makeindex.tex` provided with the program source code.)

[5] Frank Mittelbach. The `doc`-option. *TUGboat*, Vol. 10(2), *pp.* 245–273, July 1989.

[6] Frank Mittelbach, Denys Duchier and Johannes Braams. `docstrip.dtx` (to appear). The file is part of the DOC package.

[7] R. E. Raspe (*1737, †1797). Baron Münchhausens narrative of his marvellous travels and campaigns in Russia. Oxford, 1785.

[8] Rainer Schöpf. A New Implementation of LATEX's `verbatim` and `verbatim*` Environments. File `verbatim.doc`, version 1.4i.