LaTeX 2_ε pour les auteurs de classes et packages *

12 mars 1999

Table des matières

*traduit par S. Le Callonnec (<sebastien.le-callonnec@eleve.emn.fr>) le 6 février 1999. Titre original : « LaTeX 2_ε for class and package writers »

1 Introduction

Ce document est une introduction à l'écriture de classes et packages pour LaTeX, avec une attention particulière donnée à la mise à jour des packages LaTeX 2.09 vers LaTeX 2_ε. Ce sujet est également traité par un article de Johannes Braams publié dans TUGboat 15.3.

1.1 Écriture de classes et packages pour LaTeX 2_ε

LaTeX est un système de préparation de document qui permet à l'auteur du document de se concentrer sur le contenu de son texte, sans avoir trop à se soucier de la mise en forme de celui-ci. Par exemple, les chapitres sont indiqués par `\chapter{`⟨*title*⟩`}` plutôt que par la sélection du 18 pt, gras.

Le fichier qui contient l'information concernant le passage de la structure logique (comme « `\chapter` ») à la composition (comme « 18 pt gras en drapeau à droite ») est une *classe de document*. De surcroît, certaines caractéristiques (telles que l'usage de la couleur ou des graphiques inclus) sont indépendantes de la classe de document, et sont contenues dans des *packages*.

Une des plus grandes différences entre LaTeX 2.09 et LaTeX 2_ε réside dans les commandes utilisées pour rédiger les packages et classes. En LaTeX 2.09, il existait peu d'appuis pour écrire les fichiers `.sty` et les rédacteurs devaient recourir à des commandes de bas niveau.

LaTeX 2_ε fournit des commandes de haut niveau afin de structurer les packages. Il est également plus facile de construire des classes et packages sur d'autres, pour écrire par exemple une classe pour un rapport technique local `cetechr` (pour le département du Génie Chimique[1]) basée sur `article`.

[1] NdT : « Chemical Engineering department »

1.2 Aperçu

Ce document contient un aperçu sur la façon d'écrire des classes et packages pour LaTeX. Il *n'*introduit *pas* toutes les commandes nécessaires pour écrire des packages : celles-ci peuvent être trouvées dans le *LaTeX: A Document Preparation System* ou *The LaTeX Companion*. Il décrit cependant les nouvelles commandes pour structurer les classes et packages.

La section 2.7 contient quelques conseils généraux concernant l'écriture de classes et packages. Elle décrit la différence entre classes et packages, les conventions de nommage des commandes, l'utilisation de `doc` et `docstrip`, comment les commandes de gestion des boîtes et des fichiers de TeX interagissent avec LaTeX. Elle contient également quelques conseils sur le style général de LaTeX.

La section 3 décrit la structure des classes et packages. Ceci inclut la construction de classes et packages sur d'autres classes et packages, la déclaration d'options et de commandes. Elle contient également des classes exemples.

La section 4 liste les nouvelles commandes de classe et package.

La section 6 donne des conseils détaillés sur la façon de mettre à jour les classes et packages LaTeX 2.09 vers LaTeX 2_ε.

1.3 Pour plus d'informations

Pour une introduction générale à LaTeX, y compris les nouvelles fonctionnalités de LaTeX 2_ε, vous devriez lire *LaTeX: A Document Preparation System* de Leslie Lamport [3].

Une description plus détaillée des nouvelles fontionnalités de LaTeX, et un aperçu de plus de 150 packages, peuvent être trouvés dans *The LaTeX Companion*, un livre de Michel Goossens, Frank Mittelbach et Alexander Samarin [1].

Le système LaTeX est basé sur TeX, qui est décrit dans *The TeXbook* de Donald E. Knuth [2].

Il y a nombre de fichiers de documentation qui accompagnent chaque copie de LaTeX. Une copie des *LaTeX News* sort avec chaque version de LaTeX éditée tous les 6 mois, et peut être trouvée dans les fichiers `ltnews*.tex`. Le guide de l'auteur *LaTeX 2_ε for Authors* décrit les nouvelles caractéristiques de document LaTeX ; il se trouve dans `usrguide.tex`. Le guide *LaTeX 2_ε Font Selection* décrit le procédé de sélection de fonte LaTeX pour les rédacteurs de classes et packages ; il se trouve dans `fntguide.tex`. La configuration de LaTeX est traitée par le guide *Configuration options for LaTeX 2_ε* dans le fichier `cfgguide.tex` tandis que la philosophie derrière notre politique de modification de LaTeX est décrite dans *Modifying LaTeX* dans le fichier `modguide.tex`.

Nous transcrivons progressivement le code source de LaTeX en un document LaTeX *LaTeX: the program*. Ce document inclut un index des commandes LaTeX et peut être composé à partir de `source2e.tex`.

Pour plus d'informations sur TeX et LaTeX, veuillez contacter votre Groupe local d'Utilisateurs de TeX, ou le Groupe international des Utilisateurs de TeX.

Voici quelques adresses utiles.

TeX Users Group, P.O. Box 1239, Three Rivers, CA 93271-1239, USA
Fax : +1 209 561 4584 Email : `tug@mail.tug.org`

UK TUG, 1 Eymore Close, Selly Oak, Birmingham B29 4LB, UK
Fax : +44 121 476 2159 Email : `uktug-enquiries@tex.ac.uk`

GUTenberg, c/o Irisa, Campus universitaire de Beaulieu
F-35042 Rennes Cedex, France
Fax : +33 (0)5 61 52 83 44 Email : `secretariat@gutenberg.eu.org`

1.4 Politique sur les classes standard

Beaucoup des problèmes signalés que nous recevons concernant les classes standard ne sont pas dûs à des bogues, mais suggèrent, plus ou moins poliment, que les décisions de design prises pour ces classes ne sont « pas optimales » et nous demandent de les modifier.

Il y a plusieurs raisons pour lesquelles nous ne devrions pas faire de tels changements à ces fichiers.

- Même s'il est malavisé, le comportement actuel est clairement ce qui était attendu lors du design de ces classes.
- Ce n'est pas une bonne pratique de changer des aspects des « classes standard » car beaucoup de gens comptent sur elles.

Nous avons par conséquent décidé de ne pas considérer l'éventualité de telles modifications ni même de perdre du temps à justifier cette décision. Ceci ne signifie pas que nous ne sommes pas d'accord qu'il y a beaucoup de déficiences dans le design de ces classes, mais nous avons des tâches de plus haute priorité que d'expliquer pourquoi les classes standard de LaTeX ne peuvent être changées.

Les productions de meilleures classes ou de packages améliorant ces classes seraient bien sûr les bienvenues. Quand vous réfléchirez à une telle déficience, votre première pensée sera, nous l'espérons, « Comment puis-je améliorer ceci ? »

Des considérations similaires peuvent être appliquées aux parties du noyau implémentant des décisions de design, dont beaucoup devraient être laissées au fichier de classe et ne le sont pas dans le système actuel. Nous réalisons que dans de tels cas, il est plus difficile pour vous de rectifier le problème vous-même mais faire de tels changements dans le noyau serait certainement un projet majeur pour nous. Par conséquent, de telles améliorations devront attendre LaTeX3.

2 Écriture de classes et packages

Cette section recouvre quelques points généraux quant à l'écriture de classes et packages LaTeX.

2.1 Anciennes versions

Si vous êtes en train de mettre à jour un fichier de style LaTeX 2.09 déjà existant, nous vous recommandons de geler la version 2.09 et de ne plus la maintenir. L'expérience acquise avec les différents dialectes de LaTeX qui existaient au début des années 90 montre que le maintien de packages pour différentes versions de LaTeX est presque impossible. Il sera bien sûr nécessaire pour certaines organisations de maintenir les deux versions en parallèle pendant quelques temps mais ceci n'est pas essentiel pour les packages et classes gérés par des particuliers : ils ne devraient gérer que le nouveau standard LaTeX 2_ε, et non les versions obsolètes de LaTeX.

2.2 Utilisation de « docstrip » et « doc »

Lorsque vous écrivez un gros package ou classe pour LaTeX, il vous faut alors considérer l'utilisation du programme `doc` livré avec LaTeX. Les classes et packages LaTeX écrits en l'utilisant peuvent être traités de deux façons : ils peuvent être exécutés par LaTeX, pour produire la documentation ; ils peuvent également être traités par `docstrip`, pour produire le fichier `.cls` ou `.sty`.

Le programme `doc` peut automatiquement produire les index de définitions, les index des commandes et les listes de transcription des changements. Ceci est particulièrement utile pour le maintien et la documentation de larges sources TeX.

Les sources documentées du noyau LaTeX lui-même, et des classes standard, etc., sont des documents `doc` ; ils se trouvent dans les fichiers `.dtx` de la distribution. Vous pouvez en fait composer le code source du noyau comme un long document, complété par un index, en exécutant LaTeX sur `source2e.tex`.

Pour plus d'informations concernant `doc` et `docstrip`, consultez les fichiers `docstrip.dtx`, `doc.dtx`, et *The LaTeX Companion*. Pour des exemples d'utilisation, regardez ces fichiers `.dtx`.

2.3 Est-ce une classe ou un package ?

La première chose à faire quand vous voulez mettre de nouvelles commandes LaTeX dans un fichier est de décider si ce fichier doit être une *classe de document* ou un *package*. En règle générale :

> Si les commandes peuvent être utilisées dans n'importe quelle classe de document, faites alors un package ; sinon, faites-en une classe.

Il y a deux principaux types de classes : celles, comme `article`, `report` ou `letter`, qui existent par elles-même, et celles qui ne sont que des extensions ou variations d'autres classes — la classe de document `proc`, par exemple, qui est construite sur la classe de document `article`.

Une entreprise peut avoir ainsi une classe locale `ownlet` pour imprimer des lettres avec leur propre entête. Une telle classe serait construite par-dessus la classe `letter` existante mais, ne pouvant être utilisée par aucune autre classe, serait appelée `ownlet.cls` plutôt que `ownlet.sty`.

Le package `graphics`, au contraire, fournit des commandes pour l'inclusion d'images dans un document LaTeX. Du fait que ces commandes peuvent être utilisées avec n'importe quelle classe de document, nous avons un fichier `graphics.sty` plutôt que `graphics.cls`.

2.4 Noms de commandes

LaTeX possède trois types de commandes.

Il y a les commandes de l'auteur, telles que `\section`, `\emph` et `\times` : la plupart d'entre elles possèdent un nom court, en bas de casse.

Il y a également les commandes du rédacteur de classes et packages : la plupart d'entre elles ont des noms mélangeant haut et bas de casse, comme celles qui suivent :

```
\InputIfFileExists  \RequirePackage  \PassOptionsToClass
```

Enfin, il y a les commandes internes utilisées pour l'implémentation LaTeX, telles que `\@tempcnta`, `\@ifnextchar` et `\@eha` : la plupart de ces commandes contiennent `@` dans leur nom, ce qui veut dire qu'elle ne peuvent être utilisées dans des documents, et seulement dans les fichiers de classes et packages.

Malheureusement, pour des raisons historiques, la distinction entre ces commandes est souvent floue. `\hbox` est, par exemple, une commande interne qui ne devrait être utilisée que dans le noyau LaTeX, alors que `\m@ne` est la constante -1 et aurait pu être appelée `\MinusOne`.

Ces indications restent toutefois utiles : si une commande possède `@` dans son nom, elle n'appartient alors pas au langage LaTeX géré — et son comportement peut changer dans de futures versions ! Si une commande contient des majuscules et minuscules ou est décrite dans *LaTeX: A Document Preparation System*, vous pouvez alors compter sur le fait que les futures versions de LaTeX 2_ε géreront cette commande.

2.5 Commandes de boîtes et couleur

Même si vous n'avez pas l'intention d'utiliser la couleur dans vos propres documents, vous pouvez vous assurer en prenant note des points développés dans cette section que votre classe ou package est compatible avec le package `color`. Ceci peut profiter aux gens utilisant votre classe ou package et ayant accès à une imprimante couleur.

Le moyen le plus facile pour assurer la « sécurité couleur » est de toujours utiliser les commandes de boîtes de LaTeX plutôt que les primitives TeX, ce qui veut dire utiliser `\sbox` plutôt que `\setbox`, `\mbox` plutôt que `\hbox` et `\parbox` ou l'environnement `minipage` plutôt que `\vbox`. Les commandes de boîtes LaTeX possèdent de nouvelles options, ce qui signifie qu'elles sont plus puissantes que les primitives TeX.

Comme exemple de ce qui peut poser problème, pensez que dans `{\ttfamily` ⟨*text*⟩`}`, la fonte est restaurée juste *avant* l'`}`, alors que dans la construction d'apparence similaire `{\color{green}` ⟨*text*⟩`}` la couleur est restaurée juste *après* l'`}` finale. Normalement, cette distinction n'a aucune importance ; mais considérez une assignation primitive TeX de boîte telle que :

```
\setbox0=\hbox{\color{green} ⟨text⟩}
```

La restauration de couleur intervient alors après l'} et donc *n'est pas* sauvée dans la boîte. Les mauvais effets que ceci peut avoir dépend exactement de la façon dont la couleur est implémentée : cela peut aller de la mauvaise couleur dans le reste du document à l'obtention d'erreurs dans le pilote dvi utilisé pour imprimer le document.

La commande `\normalcolor` a également son intérêt. Elle n'est en temps normal que `\relax` (i.e., ne fait rien), mais vous pouvez l'utiliser plutôt comme `\normalfont` pour passer des régions de la page telles que légendes ou entêtes de sections à la « couleur principale du document ».

2.6 Définition de caractères de texte et de maths

Parce que LaTeX 2_ε gère différents encodages, les définitions de commandes pour produire symboles, accents, glyphes composites, etc. doivent être faites en utilisant les commandes développées à cette fin et décrites dans *LaTeX 2_ε Font Selection*. Cette partie du système est encore en développement ; c'est pourquoi de telles tâches doivent être entreprises avec précaution.

De plus, `\DeclareRobustCommand` devrait être utilisé pour définir des commandes de ce type, indépendantes de l'encodage.

Notez qu'il n'est plus possible de faire référence aux jeux de caractères mathématiques en dehors du mode mathématique : par exemple ni `\textfont 1` ni `\scriptfont 2` n'ont la garantie d'être définis dans d'autres modes.

2.7 Style général

Le nouveau système fournit beaucoup de commandes étudiées pour vous aider à produire des fichiers de classes et packages bien structurés, à la fois robustes et portables. Cette section esquisse quelques façons de les utiliser intelligemment.

2.7.1 Charger d'autres fichiers

LaTeX fournit les commandes : New description 1995/12/01

```
\LoadClass        \LoadClassWithOptions
\RequirePackage   \RequirePackageWithOptions
```

pour utiliser des classes ou packages à l'intérieur d'autres classes ou packages. Nous vous recommandons expressément de les utiliser plutôt que la commande primitive `\input` pour un certain nombre de raisons.

Les fichiers chargés par `\input` ⟨*filename*⟩ ne seront pas répertoriés dans la liste `\listfiles`.

Si un package est toujours chargé avec `\RequirePackage...` ou `\usepackage` alors même si son chargement est demandé plusieurs fois, il ne sera toujours fait qu'une seule fois. Au contraire, s'il est chargé avec `\input`, il peut être alors chargé plus d'une fois. Un tel chargement supplémentaire peut faire perdre à la fois temps et mémoire et peut amener à de bien étranges résultats.

Si un package fournit un traitement d'option, là encore d'étranges résultats sont possibles si le package est chargé par `\input` plutôt que par `\usepackage` or `\RequirePackage`....

Si le package `foo.sty` charge le package `baz.sty` en utilisant `\input baz.sty`, alors l'utilisateur obtiendra l'avertissement :

```
LaTeX Warning: You have requested package 'foo',
               but the package provides 'baz'.
```

Ainsi, pour plusieurs raisons, l'utilisation de `\input` pour charger des packages n'est pas une bonne idée.

Malheureusement, si vous mettez à jour le fichier `myclass.sty` pour en faire une classe de document, il vous faut vous assurer que les anciens fichiers contenant `\input myclass.sty` marchent encore.

Ceci était également vrai pour les classes standard (`article`, `book` et `report`), car beaucoup des styles de document LaTeX 2.09 existants contiennent `\input article.sty`. L'approche que nous utilisons afin de résoudre ceci est de fournir des fichiers minimaux `article.sty`, `book.sty` et `report.sty`, qui chargent tout simplement les fichiers de classe appropriés.

Par exemple, `article.sty` ne contient juste que les lignes suivantes :

```
\NeedsTeXFormat{LaTeX2e}
\@obsoletefile{article.cls}{article.sty}
\LoadClass{article}
```

Vous pouvez désirer faire de même ou, si vous pensez que c'est sans danger, décider de simplement ôter `myclass.sty`.

2.7.2 Faire dans le robuste

Nous considérons comme un bon entraînement l'utilisation lors de l'écriture de packages et classes des commandes LaTeX autant que possible.

Ainsi, à la place de `\def...`, nous recommandons d'utiliser `\newcommand`, `\renewcommand` ou `\providecommand` ; `\CheckCommand` est également utile. Il y a moins de risque en faisant ceci que vous redéfinissiez par inadvertance une commande, donnant des résultats pour le moins inattendus.

Quand vous définissez un environnement, utilisez `\newenvironment` ou `\renewenvironment` au lieu de `\def\foo{...}` et `\def\endfoo{...}`.

Si vous avez besoin d'établir ou de changer la valeur d'un registre ⟨*dimen*⟩ ou ⟨*skip*⟩, utilisez `\setlength`.

Pour manipuler des boîtes, utilisez les commandes LaTeX telles que `\sbox`, `\mbox` et `\parbox` plutôt que `\setbox`, `\hbox` et `\vbox`.

Utilisez `\PackageError`, `\PackageWarning` ou `\PackageInfo` (ou les commandes de classes équivalentes) plutôt que `\@latexerr`, `\@warning` ou `\wlog`.

Il est encore possible de déclarer des options en définissant `\ds@`⟨*option*⟩ et en appelant `\@options` ; nous recommandons cependant d'employer les commandes `\DeclareOption` et `\ProcessOptions` à la place. Ces dernières sont en effet plus puissantes et utilisent moins de mémoire. Donc, plutôt que d'utiliser :

```
\def\ds@draft{\overfullrule 5pt}
\@options
```

vous devriez utiliser :

```
\DeclareOption{draft}{\setlength{\overfullrule}{5pt}}
\ProcessOptions\relax
```

L'avantage de ce genre de pratique est que votre code est plus lisible et, plus important, qu'il est moins exposé à d'éventuelles incompatibilités dans les prochaines versions de LaTeX.

2.7.3 Faire dans le portable

Il est également raisonnable de rendre vos fichiers le plus portable possible. Afin d'assurer cela, ils ne devraient contenir que du texte 7 bits visible et porter un nom d'au plus huit caractères (plus les trois lettres de l'extension). De plus, ils **ne doivent** bien sûr **pas** avoir le même nom qu'un fichier de la distribution standard LaTeX, même si son contenu peut être tout à fait similaire.

Il est également utile que les classes ou packages locaux aient un préfixe commun ; par exemple, ceux de l'Université de Nulle Part pourraient commencer par `unp`. Ceci aide à éviter que si chaque Université a sa propre classe de thèse, toutes soient appelées `thesis.cls`.

Si vous comptez sur certaines fonctions du noyau LaTeX, ou sur un package, veuillez spécifier la version-date dont vous avez besoin. Par exemple, les commandes d'erreur pour package ont été introduites dans la version de juin 1994 ; donc, si vous les employez, vous devriez spécifier :

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

2.7.4 Points d'entrée utiles

Certains packages et styles de document devaient redéfinir les commandes `\document` ou `\enddocument` pour parvenir à leurs fins. Ce n'est plus nécessaire. Vous pouvez désormais utiliser les « points d'entrée » `\AtBeginDocument` et `\AtEndDocument` (voir la section 4.6). Une fois encore, l'utilisation de ces points d'entrée rend improbable les ruptures avec les futures versions de LaTeX. Elle rend également plus probable le fonctionnement de votre package avec celui de quelqu'un d'autre.

Cependant, notez que le code dans le point d'entrée `\AtBeginDocument` appartient au préambule du document. Il y a ainsi des restrictions sur ce qui peut y être mis ; en particulier, aucune composition ne peut y être faite. New description 1996/12/01

3 La structure d'une classe ou package

Les classes et packages LaTeX 2ε ont plus de structure que les fichiers de style LaTeX 2.09 n'en avaient. L'esquisse d'une classe ou d'un package est la suivante :

Identification Le fichier dit qu'il est un package ou une classe LaTeX 2_ε, et donne une courte description de lui-même.

Déclarations préliminaires Ici, le fichier déclare quelques commandes et peut également charger d'autres fichiers. Habituellement, ces commandes seront juste celles requises pour le code utilisé dans les options déclarées.

Options Le fichier déclare et traite ses options.

D'autres déclarations C'est l'endroit où le fichier accomplit le gros de son travail : déclaration de nouvelles variables, commandes et fontes, et chargement d'autres fichiers.

3.1 Identification

La première chose qu'une classe ou qu'un package fait est de s'identifier lui-même. Les fichiers de package le font de la manière suivante :

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{⟨package⟩}[⟨date⟩ ⟨autre information⟩]
```

Par exemple :

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{latexsym}[1994/06/01 Standard LaTeX package]
```

Les fichiers de classes le font quant à eux ainsi :

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{⟨nom-classe⟩}[⟨date⟩ ⟨autre information⟩]
```

Par exemple :

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{article}[1994/06/01 Standard LaTeX class]
```

La ⟨*date*⟩ devrait être donnée sous la forme « YYYY/MM/DD » et doit être présente si l'argument optionnel est utilisé (ceci est également vrai pour la commande `\NeedsTeXFormat`). New description 1998/06/19

Tout écart par rapport à cette syntaxe résultera en des erreurs TeX de bas niveau — les commandes s'attendent à une syntaxe valide pour accélérer l'utilisation quotidienne du package ou de la classe et ne prévoit pas le cas où le développeur a commis une erreur !

La date est vérifiée chaque fois que l'utilisateur la spécifie dans la commande `\documentclass` ou `\usepackage`. Par exemple, si vous écriviez :

```
\documentclass{article}[1995/12/23]
```

alors les utilisateurs à différents endroits obtiendraient un avertissement leur signalant que leur copie d'`article` est obsolète.

La description d'une classe est affichée quand la classe est utilisée. La description d'un package est insérée dans le fichier log. Ces descriptions sont également affichées par la commande `\listfiles`. L'expression **`Standard LaTeX` ne doit pas** être utilisée dans la bannière d'identification d'un fichier autre que ceux de la distribution standard LaTeX.

3.2 Utilisation des classes et packages

La première différence importante entre les fichiers de style LaTeX 2.09 et les packages et classes de LaTeX 2_ε est que LaTeX 2_ε gère la *modularité*, au sens où il peut construire des fichiers à partir de petits blocs de construction plutôt qu'à partir d'un seul grand fichier.

Un package ou classe LaTeX peut charger un package comme suit :

```
\RequirePackage[⟨options⟩]{⟨package⟩}[⟨date⟩]
```

Par exemple :

```
\RequirePackage{ifthen}[1994/06/01]
```

Cette commande a la même syntaxe que la commande d'auteur `\usepackage`. Elle permet aux packages ou aux classes d'utiliser les fonctions d'autres packages. Par exemple, en chargeant le package `ifthen`, l'auteur d'un package peut utiliser les commandes « if...then...else... » fournies par ce package.

Une classe LaTeX peut charger une autre classe comme suit :

```
\LoadClass[⟨options⟩]{⟨om-classe⟩}[⟨date⟩]
```

Par exemple :

```
\LoadClass[twocolumn]{article}
```

Cette commande a la même syntaxe que la commande d'auteur `\documentclass`. Elle permet aux classes d'être basées sur la syntaxe et l'apparence d'une autre classe. Par exemple, en chargeant la classe `article`, l'auteur d'un package a seulement à changer les parties d'`article` qu'il n'aime pas, plutôt que de partir de zéro.

Les commandes suivantes peuvent être utilisées dans le cas classique où vous voulez simplement charger une classe ou un package avec exactement les mêmes options que celles utilisées par la classe courante. New feature 1995/12/01

```
\LoadClassWithOptions{⟨nom-classe⟩}[⟨date⟩]
\RequirePackageWithOptions{⟨package⟩}[⟨date⟩]
```

Par exemple :

```
\LoadClassWithOptions{article}
\RequirePackageWithOptions{graphics}[1995/12/01]
```

3.3 Déclaration d'options

L'autre importante différence entre les styles LaTeX 2.09 et les packages et classes LaTeX 2_ε est dans la manipulation des options. Les packages et classes peuvent à présent déclarer des options ; celles-ci peuvent être spécifiées par les auteurs. Par exemple, l'option `twocolumn` est déclarée par la classe `article`. Notez que le nom d'une option ne devrait contenir que les caractères autorisés dans un « nom LaTeX » ; en particulier, il ne doit pas contenir de séquences contrôle. New description 1998/12/01

Une option est déclarée comme suit :

```
\DeclareOption{⟨option⟩}{⟨code⟩}
```

Par exemple, l'option `dvips` (légèrement simplifiée) du package `graphics` est implémentée comme ceci :

```
\DeclareOption{dvips}{\input{dvips.def}}
```

Cela signifie que quand un auteur écrit `\usepackage[dvips]{graphics}`, le fichier `dvips.def` est chargé. Un autre exemple : l'option `a4paper` est déclarée dans la classe `article` afin de fixer les longueurs `\paperheight` et `\paperwidth` :

```
\DeclareOption{a4paper}{%
   \setlength{\paperheight}{297mm}%
   \setlength{\paperwidth}{210mm}%
}
```

Parfois, un utilisateur appellera une option que la classe ou le package n'a pas explicitement déclarée. Par défaut, ceci produira un avertissement (pour les classes) ou une erreur (pour les packages) ; ce comportement peut être changé comme suit :

```
\DeclareOption*{⟨code⟩}
```

Par exemple, pour que le package `fred` produise un avertissement plutôt qu'une erreur pour les options inconnues, vous pouvez spécifier :

```
\DeclareOption*{%
   \PackageWarning{fred}{Option inconnue '\CurrentOption'}%
}
```

Ainsi, si un auteur écrit `\usepackage[foo]{fred}`, il obtiendra l'avertissement `Package fred Warning : Option inconnue 'foo'`. Autre exemple : le package `fontenc` essaie de charger un fichier ⟨*ENC*⟩`enc.def` lorsque l'option ⟨*ENC*⟩ est utilisée. Ceci peut être fait en écrivant :

```
\DeclareOption*{%
   \input{\CurrentOption enc.def}%
}
```

Il est possible de passer des options à un autre package ou classe, en utilisant la commande `\PassOptionsToPackage` ou `\PassOptionsToClass` (notez que ce sont des opérations spécialisées qui ne marchent que pour les noms d'options). Par exemple, pour passer toutes les options inconnues à la classe `article`, vous pouvez utiliser : New description 1998/12/01

```
\DeclareOption*{%
   \PassOptionsToClass{\CurrentOption}{article}%
}
```

Si vous faites ceci, il faut alors vous assurer que vous chargez la classe à un moment ou à un autre, plus tard, sinon les options ne seront jamais traitées !

Jusqu'à présent, nous avons expliqué comment déclarer des options, et non comment les exécuter. Pour traiter les options avec lesquelles le fichier a été appelé, vous devez utiliser :

```
\ProcessOptions\relax
```

Ceci exécute le ⟨*code*⟩ pour chacune des options qui ont été à la fois spécifiées et déclarées (voir la section 4.7 pour les détails sur comment ceci est fait).

Par exemple, si le fichier du package `jane` contient :

```
\DeclareOption{foo}{\typeout{Saw foo.}}
\DeclareOption{baz}{\typeout{Saw baz.}}
\DeclareOption*{\typeout{What's \CurrentOption?}}
\ProcessOptions\relax
```

et si un auteur écrit `\usepackage[foo,bar]{jane}`, il verra alors les messages `Saw foo.` et `What's baz?`

3.4 Un fichier de classe minimal

Le gros de la tâche d'une classe ou d'un package est de définir de nouvelles commandes, ou de changer l'apparence de documents. Ceci est fait dans le corps du package, en utilisant des commandes telles que `\newcommand` ou `\setlength`.

LaTeX 2_ε fournit plusieurs nouvelles commandes pour aider les auteurs de classes et de packages ; celles-ci sont décrites en détail dans la section 4.

Il y a trois choses que tout fichier de classe *doit* contenir : une définition de `\normalsize` et des valeurs de `\textwidth` et `\textheight`. Un fichier de classe minimal[2] ressemble donc à cela :

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{minimal}[1995/10/30 Standard LaTeX minimal class]
\renewcommand{\normalsize}{\fontsize{10pt}{12pt}\selectfont}
\setlength{\textwidth}{6.5in}
\setlength{\textheight}{8in}
```

Cependant, ce fichier de classe ne gèrera pas les notes de pied de page, les marges, les flottants, etc., et ne fournira pas non plus les commandes de fontes de 2 lettres telles que `\rm` ; ainsi, la plupart des classes contiendront un petit peu plus que ce minimum !

3.5 Exemple : une classe locale de lettre

Une entreprise peut avoir sa propre classe de lettre pour composer des lettres au style de l'entreprise. Cette section montre une simple implémentation d'une telle classe, bien qu'une vraie classe nécessiterait plus de structure.

Cette classe commence par s'annoncer comme étant le fichier `neplet.cls`.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{neplet}[1995/04/01 NonExistent Press letter class]
```

[2] Cette classe est maintenant dans la distribution standard, sous le nom `minimal.cls`.

Ensuite, la partie suivante passe n'importe quelle option à la classe `letter`, qui est chargée avec l'option `a4paper`.

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{letter}}
\ProcessOptions\relax
\LoadClass[a4paper]{letter}
```

Afin d'utiliser l'entête de l'entreprise, la classe redéfinit le style de la page `firstpage` : c'est le style de page utilisé sur la première page des lettres.

```
\renewcommand{\ps@firstpage}{%
   \renewcommand{\@oddhead}{⟨l'en-tête de la lettre va ici⟩}%
   \renewcommand{\@oddfoot}{⟨le pied de page de la lettre va ici⟩}%
}
```

Et voilà !

3.6 Exemple : une classe lettre d'information

Une simple lettre d'information peut être composée avec LaTeX, en utilisant une variante de la classe `article`. La classe commence par s'annoncer comme étant le fichier `smplnews.cls`.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{smplnews}[1995/04/01 The Simple News newsletter class]

\newcommand{\headlinecolor}{\normalcolor}
```

Elle passe la plupart des options spécifiées à la classe `article`, sauf l'option `onecolumn`, qui est désactivée et l'option `green` qui met le titre en vert.

```
\DeclareOption{onecolumn}{\OptionNotUsed}
\DeclareOption{green}{\renewcommand{\headlinecolor}{\color{green}}}

\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}

\ProcessOptions\relax
```

Elle charge alors la classe `article` avec l'option `twocolumn`.

```
\LoadClass[twocolumn]{article}
```

Étant donné que la lettre d'information sera imprimée en couleur, elle charge à présent le package `color`. La classe ne spécifie pas l'option du pilote car cela sera précisé par l'utilisateur de la classe `smplnews`.

```
\RequirePackage{color}
```

La classe redéfinit alors `\maketitle` pour produire un titre en 72 pt Helvetica gras oblique, dans la couleur appropriée.

```
\renewcommand{\maketitle}{%
   \twocolumn[%
      \fontsize{72}{80}\fontfamily{phv}\fontseries{b}%
      \fontshape{sl}\selectfont\headlinecolor
      \@title
   ]%
}
```

Elle redéfinit `\section` et désactive la numérotation des sections.

```
\renewcommand{\section}{%
   \@startsection
      {section}{1}{0pt}{-1.5ex plus -1ex minus -.2ex}%
      {1ex plus .2ex}{\large\sffamily\slshape\headlinecolor}%
}

\setcounter{secnumdepth}{0}
```

Elle fixe également trois choses importantes.

```
\renewcommand{\normalsize}{\fontsize{9}{10}\selectfont}
\setlength{\textwidth}{17.5cm}
\setlength{\textheight}{25cm}
```

En pratique, une classe nécessiterait plus que cela : elle fournirait des commandes pour le numéro de la lettre, les auteurs des articles, les styles de page, etc. Toutefois, ce squelette donne un début. Le fichier de la classe `ltnews` n'est pas beaucoup plus compliqué que celui-ci.

4 Commandes pour les auteurs de classes et packages

Cette section décrit brièvement chacune des nouvelles commandes pour les auteurs de classes et de packages. Pour vous renseigner sur les autres aspects du nouveau système, vous devriez lire également *LaTeX: A Document Preparation System*, *The LaTeX Companion* et *LaTeX 2ε for Authors*.

4.1 Identification

Le premier groupe de commandes discuté ici est celui utilisé pour identifier votre fichier de classe ou de package.

`\NeedsTeXFormat {`⟨*nom-format*⟩`} [`⟨*date-emission*⟩`]`

Cette commande indique à TeX que ce fichier doit être traité en utilisant un format du nom ⟨*nom-format*⟩. Vous pouvez utiliser l'argument optionnel ⟨*date-emission*⟩ pour indiquer la date d'émission du format le plus ancien requis. Quand la date de réalisation du format est postérieure à celle spécifiée, un avertissement est produit. Le ⟨*nom-format*⟩ standard est `LaTeX2e`. La date, si elle est présente, doit être mise sous la forme YYYY/MM/DD.

Exemple :

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

`\ProvidesClass` {⟨*nom-classe*⟩} [⟨*info-version*⟩]
`\ProvidesPackage` {⟨*nom-packge*⟩} [⟨*info-version*⟩]

Ceci déclare que le fichier actuel contient les définitions pour la classe de document ⟨*nom-classe*⟩ ou le package ⟨*nom-package*⟩.

L'optionnelle ⟨*info-version*⟩, si elle est utilisée, doit contenir :
- la date de réalisation de la version de ce fichier, sous la forme YYYY/MM/DD ;
- suivie de manière optionnelle par une espace et une courte description, qui peut inclure éventuellement un numéro de version.

La syntaxe ci-dessus doit être suivie scrupuleusement afin que cette information puisse être utilisée par `\LoadClass` ou `\documentclass` (pour les classes), ou `\RequirePackage` ou `\usepackage` (pour les packages) pour vérifier que la version n'est pas trop ancienne.

L'information ⟨*info-version*⟩ entière est affichée par `\listfiles` et ne doit pas, par conséquent, être trop longue.

exemple :

```
\ProvidesClass{article}[1994/06/01 v1.0 Standard LaTeX class]
\ProvidesPackage{ifthen}[1994/06/01 v1.0 Standard LaTeX package]
```

`\ProvidesFile` {⟨*nom-fichier*⟩} [⟨*info-version*⟩]

Ceci est similaire aux deux commandes précédentes, sauf qu'ici, le nom complet, y compris l'extension doit être donné. Cette commande est utilisée pour la déclaration de fichiers autres que les fichiers de classe et package principaux.

Exemple :

```
\ProvidesFile{T1enc.def}[1994/06/01 v1.0 Standard LaTeX file]
```

Notez que l'expression `Standard LaTeX` **ne doit pas** être utilisée dans la bannière d'identification d'un fichier autre que ceux de la distribution standard LaTeX.

4.2 Chargement de fichiers

Ce groupe de commandes peut être utilisé pour créer votre classe de document ou package en le construisant sur des classes ou packages déjà existants. New feature 1995/12/01

`\RequirePackage` [⟨*liste-options*⟩] {⟨*nom-package*⟩} [⟨*info-version*⟩]
`\RequirePackageWithOptions` {⟨*nom-package*⟩} [⟨*info-version*⟩]

Les packages et classes devraient utiliser ces commandes pour charger d'autres packages.

L'utilisation de `\RequirePackage` est la même que la commande d'auteur `\usepackage`.

Exemples :

```
\RequirePackage{ifthen}[1994/06/01]
\RequirePackageWithOptions{graphics}[1995/12/01]
```

`\LoadClass [`⟨*liste-options*⟩`] {`⟨*nom-classe*⟩`} [`⟨*info-version*⟩`]`
`\LoadClassWithOptions {`⟨*nom-classe*⟩`} [`⟨*info-version*⟩`]`

Ces commandes ne doivent être utilisées *que* dans des fichiers de classe, et ne peuvent figurer dans des fichiers de packages ; elles peuvent être utilisées au plus une fois dans un fichier de classe. New feature 1995/12/01

L'utilisation de `\LoadClass` est la même que celle de `\documentclass` pour charger un fichier de classe.

Exemples :

```
\LoadClass{article}[1994/06/01]
\LoadClassWithOptions{article}[1995/12/01]
```

Les deux versions `WithOptions` chargent simplement le fichier de la classe (ou du package) avec exactement les mêmes options que celles utilisées par le fichier courant (classe ou package). Voir plus loin, à la section 4.5, pour une discussion plus précise quant à leur utilisation. New feature 1995/12/01

4.3 Déclaration d'option

Les commandes suivantes se chargent de la déclaration et de la manipulation des options passées aux classes de document et aux packages. Chaque option doit porter un « nom LaTeX ». New description 1998/12/01

Il existe quelques commandes conçues spécialement pour être utilisées à l'intérieur de l'argument ⟨*code*⟩ de ces commandes (voir ci-dessous).

`\DeclareOption {`⟨*nom-option*⟩`} {`⟨*code*⟩`}`

Cette commande fait de ⟨*nom-option*⟩ une « option déclarée » de la classe ou du package dans laquelle elle est insérée.

L'argument ⟨*code*⟩ contient le code à exécuter si cette option est spécifiée pour la classe ou le package ; il peut contenir n'importe quelle construction valide LaTeX 2_ε.

Exemple :

```
\DeclareOption{twoside}{\@twosidetrue}
```

`\DeclareOption* {`⟨*code*⟩`}`

Ceci déclare le ⟨*code*⟩ à exécuter pour chaque option spécifiée, mais non explicitement déclarée par ailleurs par la classe ou le package ; ce code est appelé le « code d'option par défaut » et peut contenir n'importe quelle construction valide de LaTeX 2_ε.

Si le fichier d'une classe ne contient aucun `\DeclareOption*` alors, par défaut, toutes les options spécifiées mais non déclarées pour cette classe seront de manière silencieuse passées à tous les packages (comme le seront les options spécifiées et déclarées pour cette classe).

Si un fichier de package ne contient aucun `\DeclareOption*` alors, par défault, chaque option spécifiée mais non déclarée pour ce package produira une erreur.

4.4 Commandes à l'intérieur du code d'option

Ces deux commandes ne peuvent être utilisées qu'à l'intérieur de l'argument ⟨*code*⟩ de `\DeclareOption` ou `\DeclareOption*`. Les autres commandes communément utilisées à l'intérieur de ces arguments peuvent être trouvées dans les prochaines sous-sections.

`\CurrentOption`

Ceci sera remplacé le nom de l'option courante.

`\OptionNotUsed`

Ceci fait que l'option courante est ajoutée à la liste des « options inutilisées ».

Vous pouvez désormais inclure des dièses (`#`) à l'intérieur de ces arguments ⟨*code*⟩ sans traitement spécial (auparavant, il était nécessaire de les doubler). New feature 1995/06/01

4.5 Promener des options

Ces deux commandes sont également très utiles à l'intérieur de l'argument ⟨*code*⟩ de `\DeclareOption` ou `\DeclareOption*` :

`\PassOptionsToPackage {`⟨*liste-options*⟩`} {`⟨*nom-package*⟩`}`
`\PassOptionsToClass {`⟨*liste-options*⟩`} {`⟨*nom-classe*⟩`}`

La commande `\PassOptionsToPackage` passe les noms d'option dans la ⟨*liste-options*⟩ au package ⟨*nom-package*⟩. Cela signifie que la ⟨*liste-options*⟩ est ajoutée à la liste d'options utilisées par une éventuelle commande `\RequirePackage` or `\usepackage` pour le package ⟨*nom-package*⟩.

Exemple :

```
\PassOptionsToPackage{foo,bar}{fred}
\RequirePackage[baz]{fred}
```

est la même chose que :

```
\RequirePackage[foo,bar,baz]{fred}
```

De la même manière, `\PassOptionsToClass` peut être utilisé dans un fichier de classe pour passer des options à une autre classe chargée avec `\LoadClass`.

Les effets et l'utilisation de ces deux commandes doivent être mis en contraste avec ceux des deux suivantes (documentées ci-dessus, au 4.2) : New description 1995/12/01

```
\LoadClassWithOptions
\RequirePackageWithOptions
```

La commande `\RequirePackageWithOptions` est similaire à `\RequirePackage`, mais elle charge toujours le package requis avec exactement la même liste d'options que celle utilisée par la classe ou package courant plutôt qu'avec n'importe qu'elle option fournie ou passée par `\PassOptionsToPackage`.

Le but principal de `\LoadClassWithOptions` est de permettre à une classe d'être construite sur une autre. Par exemple :

```
\LoadClassWithOptions{article}
```

À comparer avec la construction légèrement différente :

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\ProcessOptions\relax
\LoadClass{article}
```

Utilisée comme ci-dessus, les effets sont plus ou moins les mêmes, mais la première construction est tout de même moins longue à taper ; de plus, la méthode utilisant `\LoadClassWithOptions` s'exécute légèrement plus vite.

Si, cependant, la classe déclare ses propres options, les deux constructions sont alors différentes. Comparez, par exemple :

```
\DeclareOption{landscape}{\@landscapetrue}
\ProcessOptions\relax
\LoadClassWithOptions{article}
```

avec :

```
\DeclareOption{landscape}{\@landscapetrue}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\ProcessOptions\relax
\LoadClass{article}
```

Dans le premier exemple, la classe article sera chargée avec l'option `landscape` précisément quand la classe courante sera appelée avec cette option. À l'opposé, dans le second exemple, elle ne sera jamais appelée avec l'option `landscape`, étant donné que dans ce cas, les options sont passées à article seulement par le manipulateur d'option par défaut. Or, ce manipulateur n'est pas utilisé pour `landscape` car cette option est explicitement déclarée.

4.6 Différer

Ces deux premières commandes sont également essentiellement destinées à être utilisées à l'intérieur de l'argument ⟨*code*⟩ de `\DeclareOption` ou `\DeclareOption*`.

`\AtEndOfClass {⟨code⟩}`
`\AtEndOfPackage {⟨code⟩}`

Ces commandes déclarent ⟨*code*⟩ qui est mémorisé de manière interne et ensuite exécuté après le traitement du fichier de la classe ou package entier.

L'utilisation réitérée de ces commandes est autorisée : le code de leurs arguments est sauvé (et exécuté plus tard) dans l'ordre de leurs déclarations.

`\AtBeginDocument {⟨code⟩}`
`\AtEndDocument {⟨code⟩}`

Ces commandes déclarent ⟨*code*⟩ qui est mémorisé de manière interne et ensuite exécuté lorsque LaTeX exécute `\begin{document}` ou `\end{document}`.

Le ⟨*code*⟩ spécifié dans l'argument de `\AtBeginDocument` est exécuté proche de la fin du code de `\begin{document}`, *après* l'installation des tables de sélection de fontes. Cela reste par conséquent un endroit intéressant pour mettre du code qui doit être exécuté après que tout a été préparé pour la composition et lorsque le fonte courante est la fonte normale appliquée à tout le document.

Le point d'entrée `\AtBeginDocument` ne doit pas être utilisé pour du code réalisant une composition, car le résultat est assez imprévisible. New description 1995/12/01

Le ⟨*code*⟩ spécifié dans l'argument de `\AtEndDocument` est exécuté au début du code de `\end{document}`, *avant* que la page finale ne soit terminée et avant que les environnements flottants restants ne soient traités. Si une partie du ⟨*code*⟩ doit être exécutée après ces deux traitements, vous devriez inclure un `\clearpage` à un endroit approprié de ⟨*code*⟩.

L'utilisation réitérée de ces commandes est autorisée : le code de leurs arguments est sauvé (et exécuté plus tard) dans l'ordre de leurs déclarations.

`\AtBeginDvi {⟨specials⟩}` New feature 1994/12/01

Cette commande sauve dans un registre de boîte des choses à écrire dans le fichier `.dvi` au début du « shipout » de la première page du document.

Elle ne devrait pas être utilisée pour quelque chose qui ajouterait des éléments de composition dans le fichier `.dvi`.

L'utilisation réitérée de cette commande est autorisée.

4.7 Traitement d'options

`\ProcessOptions`

Cette commande exécute ⟨*code*⟩ pour chaque option sélectionnée.

Nous allons d'abord décrire comment `\ProcessOptions` marche dans le fichier d'un package, et ensuite expliquer en quoi cela diffère dans un fichier de classe.

Pour comprendre dans le détail ce que `\ProcessOptions` fait dans le fichier d'un package, vous devez connaître la différence entre les options *locales* et *globales*.

- **Les options locales** sont celles qui ont été expicitement spécifiées pour ce package particulier dans l'argument ⟨*options*⟩ de :

  `\PassOptionsToPackage{`⟨*options*⟩`}` `\usepackage[`⟨*options*⟩`]`
  `\RequirePackage[`⟨*options*⟩`]`
- **Les options globales** sont toutes les autres options déclarées par l'auteur dans l'argument ⟨*options*⟩ de `\documentclass[`⟨*options*⟩`]`.

Par exemple, supposons qu'un document commence par :

```
\documentclass[german,twocolumn]{article}
\usepackage{gerhardt}
```

alors que le package `gerhardt` appelle le package `fred` avec :

```
\PassOptionsToPackage{german,dvips,a4paper}{fred}
\RequirePackage[errorshow]{fred}
```

alors :
- les options locales de `fred` sont `german`, `dvips`, `a4paper` et `errorshow` ;
- la seule option globale de `fred` est `twocolumn`.

Quand `\ProcessOptions` est appelé, il se produit les choses suivantes.
- *D'abord*, pour chaque option déclarée jusqu'alors dans `fred.sty` par `\DeclareOption`, il faut déterminer si cette option est une option globale ou locale pour `fred` : le code est alors exécuté selon le cas.
  Ceci se fait dans l'ordre de déclaration de ces options dans `fred.sty`.
- *Ensuite*, pour chaque option *locale* restante, la commande `\ds@`⟨*option*⟩ est exécutée si elle a été définie quelque part (ailleurs que dans un `\DeclareOption`) ; sinon, le « code de l'option par défaut » est exécuté. Si aucun code d'option par défaut n'a été déclaré, un message d'erreur est retourné.
  Ceci se fait dans l'ordre dans lequel les options ont été spécifiées.

Durant tout ce processus, le système s'assure que le code déclaré pour une option est exécuté au plus une fois.

Revenons à notre exemple. Si `fred.sty` contient :

```
\DeclareOption{dvips}{\typeout{DVIPS}}
\DeclareOption{german}{\typeout{GERMAN}}
\DeclareOption{french}{\typeout{FRENCH}}
\DeclareOption*{\PackageWarning{fred}{Unknown `\CurrentOption'}}
\ProcessOptions\relax
```

alors le résultat du traitement de ce document sera :

```
DVIPS
GERMAN
Package fred Warning: Unknown `a4paper'.
Package fred Warning: Unknown `errorshow'.
```

Notez que :

– le code pour l'option `dvips` est exécuté avant celui de l'option `german`, car c'est l'ordre dans lequel elles sont déclarées dans `fred.sty` ;
– le code pour l'option `german` est exécuté seulement une fois, quand les options déclarées sont traitées ;
– les options `a4paper` et `errorshow` produisent l'avertissement du code déclaré par `\DeclareOption*` (dans l'ordre dans lequel elles ont été spécifiées), alors que ce n'est pas le cas pour l'option `twocolumn`, ceci parce que `twocolumn` est une option globale.

Dans une classe, `\ProcessOptions` fonctionne de la même façon, sauf que *toutes* les options sont locales ; la valeur par défaut de `\DeclareOption*` est alors `\OptionNotUsed` plutôt qu'une erreur.

Notez que, du fait que `\ProcessOptions` ait une forme `*`, il est sage de faire suivre la forme non-étoilée par un `\relax`, comme dans les exemples précédents, afin d'empêcher les inutiles poursuites de lecture de code et les potentiels messages induisant en erreur. New description 1995/12/01

`\ProcessOptions* \@options`

Cette commande ressemble à `\ProcessOptions` mais elle exécute les options dans l'ordre spécifié par les commandes appelantes, plutôt que dans l'ordre de leur déclaration dans la classe ou package. Pour un package, cela veut dire que les options globales sont traitées d'abord.

La macro `\@options` de LaTeX 2.09 a été rendue équivalente à cette commande afin de faciliter les remises à jour des anciens styles de document en fichier de classe LaTeX 2_ε.

`\ExecuteOptions` {⟨*liste-options*⟩}

Pour chaque option de ⟨*liste-options*⟩, cette commande exécute simplement, dans l'ordre, la commande `\ds@`⟨*option*⟩ (si cette commande n'est pas définie, alors cette option est ignorée silencieusement).

Elle peut être utilisée pour fournir une « liste d'options par défaut » juste avant `\ProcessOptions`. Supposons, par exemple, que dans un fichier de classe, vous souhaitez que la composition par défaut soit : impression recto-verso ; fontes 11 pt ; sur deux colonnes. Alors cette commande pourra spécifier :

```
\ExecuteOptions{11pt,twoside,twocolumn}
```

4.8 Commandes sûres pour les fichiers

Ces commandes s'occupent de l'insertion de fichiers ; elles assurent que la non-existence d'un fichier appelé sera traité de manière conviviale...

`\IfFileExists` {⟨*nom-fichier*⟩} {⟨*vrai*⟩} {⟨*faux*⟩}

Si le fichier existe, alors le code précisé dans ⟨*vrai*⟩ est exécuté.

S'il n'existe pas, alors le code dans ⟨*faux*⟩ est exécuté.

Cette commande *n'insère pas* le fichier.

`\InputIfFileExists {⟨nom-fichier⟩} {⟨vrai⟩} {⟨faux⟩}`

Cette commande insère le fichier ⟨*nom-fichier*⟩ s'il existe et immédiatement avant l'insertion, le code dans ⟨*vrai*⟩ est exécuté.

Si le fichier n'existe pas, alors le code dans ⟨*faux*⟩ est exécuté.

Cette commande est implémentée en utilisant `\IfFileExists`.

4.9 Rapport d'erreurs

Ces commandes doivent être utilisées dans la troisième partie des classes ou packages pour rapporter des erreurs ou fournir des informations aux auteurs.

`\ClassError {⟨nom-classe⟩} {⟨texte-erreur⟩} {⟨texte-aide⟩}`
`\PackageError {⟨nom-package⟩} {⟨texte-erreur⟩} {⟨texte-aide⟩}`

Ceci produit un message d'erreur. Le ⟨*texte-erreur*⟩ est affiché et le message d'erreur `?` est affiché. si l'utilisateur tape `h`, ⟨*texte-aide*⟩ sera affiché.

À l'intérieur de ⟨*texte-erreur*⟩ et ⟨*texte-aide*⟩, `\protect` peut être utilisé afin d'arrêter l'expansion d'une commande ; `\MessageBreak` provoque un retour à la ligne et `\space` affiche un espace.

Notez que le ⟨*texte-erreur*⟩ se verra ajouté un point final. N'en mettez donc pas dans l'argument.

Par exemple :

```
\newcommand{\foo}{FOO}
\PackageError{ethel}{%
   Votre aeroglisseur est plein d'anguilles,\MessageBreak
   et \protect\foo\space est \foo
}{%
   Oh mon Dieu! Quelque chose ne va pas.\MessageBreak
   \space \space Essayez de taper \space <return>
   \space pour poursuivre, et ignorer \protect\foo.
}
```

produit l'affichage :

```
! Package ethel Error:  Votre aeroglisseur est plein d'anguilles,
(ethel)                et \foo est FOO.

See the ethel package documentation for explanation.
```

Si l'utilisateur tape `h`, le message suivant apparaîtra :

```
Oh mon Dieu! Quelque chose ne va pas.
  Essayez de taper  <return>  pour poursuivre, et ignorer \foo.
```

```
\ClassWarning {⟨nom-classe⟩} {⟨texte-avertissement⟩}
\PackageWarning {⟨nom-package⟩} {⟨texte-avertissement⟩}
\ClassWarningNoLine {⟨nom-classe⟩} {⟨texte-avertissement⟩}
\PackageWarningNoLine {⟨nom-package⟩} {⟨texte-avertissement⟩}
\ClassInfo {⟨nom-classe⟩} {⟨texte-info⟩}
\PackageInfo {⟨nom-package⟩} {⟨texte-info⟩}
```

Les quatre commandes `Warning` sont identiques aux commandes d'erreur, à ceci près qu'elles ne produisent que des avertissements à l'écran, sans prompt d'erreur.

Les deux premières, les versions `Warning`, indiquent également le numéro de la ligne où l'avertissement a été produit tandis que les deux suivantes, les versions `WarningNoLine` ne le font pas.

Les deux commandes `Info` sont identiques, à ceci près qu'elles n'enregistrent les informations que dans le fichier de transcription, y compris le numéro de ligne. Il n'existe aucune version `NoLine` pour ces deux commandes.

À l'intérieur du ⟨*texte-averstissement*⟩ et ⟨*texte-info*⟩, `\protect` peut être utilisé afin d'arrêter l'expansion d'une commande ; `\MessageBreak` provoque un retour à la ligne et `\space` affiche un espace. De plus, aucun point final ne devrait être inséré car il y en a un inséré automatiquement.

4.10 Définition de commandes

LaTeX 2_ε fournit des méthodes supplémentaires de (re)définition de commandes destinées à être utilisées dans les fichiers de classe et package.

Les formes * de ces commandes doivent être utilisées pour définir des commandes qui ne sont pas, au sens TeX, longues. Ceci peut s'avérer utile pour les gestions d'erreur avec des commandes dont les arguments ne sont pas destinés à contenir des paragraphes entiers de texte. New feature 1994/12/01

```
\DeclareRobustCommand {⟨cmd⟩} [⟨num⟩] [⟨defaut⟩] {⟨définition⟩}
\DeclareRobustCommand* {⟨cmd⟩} [⟨num⟩] [⟨defaut⟩] {⟨définition⟩}
```

Cette commande prend les mêmes arguments que `\newcommand` mais elle déclare une commande robuste, même si le code à l'intérieur de ⟨*définition*⟩ est fragile. Vous pouvez utiliser cette commande pour définir de nouvelles commandes robustes, ou pour redéfinir des commandes existantes et les rendre robustes. Un enregistrement est inséré dans le fichier de transcription si la commande est redéfinie.

Par exemple, si `\seq` est définie comme suit :

```
\DeclareRobustCommand{\seq}[2][n]{%
  \ifmmode
    #1_{1}\ldots#1_{#2}%
  \else
    \PackageWarning{fred}{Vous ne pouvez utiliser \protect\seq\space dans le texte}%
  \fi
}
```

alors la commande `\seq` peut être utilisée avec des arguments changeants, même si `\ifmmode` ne le peut pas ; par exemple :

```
\section{Truc concernant la s\'equence $\seq{x}$}
```

Notez également qu'il n'y a pas besoin de mettre un `\relax` avant le `\ifmmode` au début de la définition : ceci est dû au fait que la protection donnée par ce `\relax` contre l'expansion au mauvais moment est fournie de manière interne.

`\CheckCommand {`⟨*cmd*⟩`} [`⟨*num*⟩`] [`⟨*defaut*⟩`] {`⟨*definition*⟩`}`
`\CheckCommand* {`⟨*cmd*⟩`} [`⟨*num*⟩`] [`⟨*defaut*⟩`] {`⟨*definition*⟩`}`

Cette commande prend les mêmes arguments que `\newcommand` mais, plutôt que de redéfinir ⟨*cmd*⟩, elle vérifie juste que la définition courante de ⟨*cmd*⟩ est exactement celle donnée par ⟨*definition*⟩. Une erreur est soulevée si ces définitions diffèrent.

Cette commande est utile pour vérifier l'état du système avant que votre package ne commence à modifier les définitions des commandes. Elle vous permet de vérifier qu'en particulier aucun autre package n'a redéfini la même commande.

4.11 Arguments changeants

L'établissement d'une protection pendant le traitement (i.e. changement) d'arguments changeants a été réimplémenté, comme l'a été la méthode d'écriture d'informations du fichier `.aux` vers d'autres fichiers tels que le fichier `.toc`. Des détails peuvent être trouvés dans le fichier `ltdefns.dtx`. New description 1994/12/01

Nous espérons que ces changements n'affecteront pas beaucoup de packages.

5 Commandes diverses, etc

5.1 Paramètres de mise en page

`\paperheight`
`\paperwidth`

Ces deux paramètres sont habituellement initialisés par la classe comme étant les dimensions du papier utilisé. Ils devraient être les dimensions réelles du papier, au contraire de `\textwidth` et `\textheight` qui correspondent à la taille du corps de texte principal, entre les marges.

5.2 Changement de casse

`\MakeUppercase {`⟨*text*⟩`}`
`\MakeLowercase {`⟨*text*⟩`}`

TEX fournit deux primitives `\uppercase` et `\lowercase` pour changer la casse du New feature 1995/06/01 texte. Celles-ci sont parfois utilisées dans des classes de document pour mettre les informations dans les entêtes en majuscules.

Malheureusement, ces primitives TEX ne changent pas la casse de caractères obtenus par des commandes comme `\ae` ou `\aa`. Pour résoudre ce problème, LATEX fournit deux nouvelles commandes `\MakeUppercase` et `\MakeLowercase`, qui font cela.

Par exemple :

`\uppercase{aBcD\ae\AA\ss\OE}`	ABCDæÅßŒ
`\lowercase{aBcD\ae\AA\ss\OE}`	abcdæÅßŒ
`\MakeUppercase{aBcD\ae\AA\ss\OE}`	ABCDÆÅSSŒ
`\MakeLowercase{aBcD\ae\AA\ss\OE}`	abcdæåßœ

Les commandes `\MakeUppercase` et `\MakeLowercase` sont elles-mêmes robustes, bien qu'elles aient des arguments changeants.

Ces commandes utilisent les primitives TEX `\uppercase` et `\lowercase`, et donc ont un certain nombre de « fonctions » inattendues. En particulier, elles changent la casse de tout (sauf les caractéres des noms de commandes) ce qui se trouve dans le texte de leur argument : ceci inclut les mathematiques, les noms d'environnement et les noms de label.

Par exemple :

```
\MakeUppercase{$x+y$ en \ref{foo}}
```

produit $X + Y$ et l'avertissement :

```
LaTeX Warning: Reference `FOO' on page ... undefined on ...
```

À long terme, nous aimerions utiliser les fontes toutes en majuscules plutôt qu'une commande comme `\MakeUppercase`, mais ceci n'est pas possible pour le moment car de telles fontes n'existent pas.

Afin que ce changement de casse marche raisonnablement bien et que les coupures de mots soient correctes, LATEX 2_ε *doit* utiliser, tout au long d'un document, la même table fixée de changement de casse. La table utilisée est établie pour l'encodage de fonte `T1` ; cela marche bien avec les fontes standard TEX pour tous les alphabets latins mais causera des problèmes lors de l'utilisation d'autres alphabets. New description 1995/12/01

5.3 L'option 'openany' dans la classe 'book'

L'option `openany` permet aux chapitres et autres débuts semblables de survenir éventuellement sur les pages de gauche. Auparavant, cette option affectait seulement `\chapter` et `\backmatter`. Elle a désormais également un influence sur `\part`, `\frontmatter` et `\mainmatter`. New description 1996/06/01

5.4 Environnements d'affichage de maths mieux définis

`\ignorespacesafterend`

Supposez que vous voulez définir un environnement pour afficher du texte numéroté comme une équation. Une façon simple de faire cela est d'écrire : New feature 1996/12/01

```
\newenvironment{texteqn}
  {\begin{equation}
     \begin{minipage}{0.9\linewidth}}
    {\end{minipage}
   \end{equation}}
```

Toutefois, si vous avez déjà essayé cela, vous avez probablement remarqué que cela ne marche pas parfaitement au milieu d'un paragraphe car un espace de séparation de mots apparaît au début de la première ligne après l'environnement.

Il existe désormais une commande supplémentaire (avec un nom très long) disponible que vous pouvez utiliser afin de remédier à ce problème ; elle doit être insérée de la manière suivante :

```
\newenvironment{texteqn}
  {\begin{equation}
     \begin{minipage}{0.9\linewidth}}
    {\end{minipage}
   \end{equation}
   \ignorespacesafterend}
```

5.5 Normalisation de l'espacement

`\normalsfcodes`

Cette commande devrait être utilisée pour restaurer les valeurs normales des paramètres influençant l'espacement entre les mots, les phrases, etc. New feature 1997/06/01

Une utilisation importante de cette fonction est de corriger un problème signalé par Donald Arseneau : la ponctuation dans les entêtes a toujours (dans tous les formats TEX connus) été potentiellement incorrecte lorsqu'un changement de page survenait pendant qu'une valeur locale des codes d'espacement est en vigueur. Ces codes d'espacement sont changés, entre autres, par la commande `\frenchspacing`) et l'environnement verbatim.

La définition correcte de ces codes est normalement automatiquement donnée dans le `\begin{document}` et n'a donc pas besoin d'être explicitement faite ; cependant, s'ils sont explicitement rendus non-vides dans le fichier d'une classe, alors l'initialisation automatique par défaut sera annulée.

6 Remise à jour des classes et packages LaTeX 2.09

Cette section décrit les changements que vous pouvez avoir besoin de faire lorsque vous remettez à jour un style LaTeX existant vers un package ou une classe. Le mieux est de toutefois partir sur une note optimiste.

Beaucoup des fichiers de style existants s'exécuteront avec LaTeX 2_ε sans la moindre modification au fichier lui-même. Si tout marche correctement, insérez dans la classe ou package nouvellement créé une note afin de préciser qu'il marche avec le nouveau standard LaTeX ; ensuite, vous pouvez le distribuer aux utilisateurs.

6.1 Essayez d'abord !

La première chose à faire est de tester votre style en mode compatibilité. Le seul changement nécessaire pour faire cela est de changer l'extension du fichier en `.cls` : ce changement n'est à faire que si votre fichier était utilisé en tant que style de document principal. Alors, sans aucune autre modification, exécutez LaTeX 2_ε sur un document qui utilise votre fichier. Ceci suppose que vous ayez un collection appropriée de fichiers qui testent toutes les fonctionnalités fournies par votre fichier de style. (Si ce n'est pas le cas, c'est le moment d'en créer un !)

Il vous faut alors modifier les documents de test de telle sorte qu'ils deviennent des documents LaTeX 2_ε : voir *LaTeX 2_ε for Authors* pour obtenir des détails sur comment faire cela. Puis réessayez-les. Vous avez désormais essayé les documents test dans le mode natif LaTeX 2_ε et le mode compatiblité LaTeX 2.09.

6.2 Résolution de problèmes

Si votre fichier ne marche pas avec LaTeX 2_ε, il y a deux raisons possibles.
- LaTeX possède à présent une interface robuste et bien définie pour la sélection de fontes, différant énormément de LaTeX 2.09.
- Votre fichier de style peut avoir utlisé quelques-unes des commandes internes LaTeX 2.09 qui ont changé ou ont été supprimées.

Quand vous corrigerez votre fichier, vous aurez probablement besoin de plus d'informations que celles affichées normalement par LaTeX 2_ε. Ceci peut être fait en initialisant le compteur `errorcontextlines` de sa valeur de défaut de -1 à une valeur bien plus grande, par ex. 999.

6.3 Satisfaire le mode compatibilité

Parfois, une collection existante de documents LaTeX 2.09 rendent inopportunes ou impossibles l'abandon entier des anciennes commandes. Si tel est le cas, il est alors possible de satisfaire les deux conventions en faisant des traitements spéciaux pour les documents traités en mode compatibilité.

`\if@compatibility`

Ce test est initialisé lorsqu'un document débute par `\documentstyle` au lieu de `\documentclass`. Un code approprié peut alors être fourni pour l'une ou l'autre des conditions, comme suit :

```
\if@compatibility
  ⟨code emulant le comportement LaTeX 2.09⟩
```

```
\else
  ⟨code convenant a LaTeX2e⟩
\fi
```

6.4 Commandes de fontes

Des commandes de fontes et de tailles sont maintenant définies par la classe de document à la place du noyau LaTeX. Si vous remettez à jour un style de document LaTeX 2.09 vers une classe qui ne charge pas une des classes standard, alors il vous faudra probablement ajouter les définitions de ces commandes.

`\rm \sf \tt \bf \it \sl \sc`

Aucune de ces formes courtes de commandes de sélections ne sont définies dans le noyau LaTeX 2_ε. Elles sont définies par tous les fichiers de classe standard.

Si vous voulez les définir dans le fichier de votre classe, il y a plusieurs façons de le faire.

Une définition possible est :

```
\newcommand{\rm}{\rmfamily}
...
\newcommand{\sc}{\scshape}
```

Ceci rendrait les commandes de fontes othogonales ; par exemple `{\bf\it texte}` produirait de l'italique gras, ainsi : ***texte***. Elles provoqueront également une erreur si elles sont utilisées en mode math.

Une autre définition possible est :

```
\DeclareOldFontCommand{\rm}{\rmfamily}{\mathrm}
...
\DeclareOldFontCommand{\sc}{\scshape}{\mathsc}
```

Ceci fera agir `\rm` comme `\rmfamily` en mode texte (voir ci-dessus) et lui fera sélectionner l'alphabet math `\mathrm` en mode math.

Ainsi `${\rm math} = X + 1$` produira « $\mathrm{math} = X + 1$ ».

Si vous ne voulez pas que la sélection de fonte soit orthogonale, vous pouvez alors suivre les classes standard et définir :

```
\DeclareOldFontCommand{\rm}{\normalfont\rmfamily}{\mathrm}
...
\DeclareOldFontCommand{\sc}{\normalfont\scshape}{\mathsc}
```

Cela signifie, par exemple, que `{\bf\it texte}` produira une graisse moyenne (plutôt que grasse) italique, comme ceci : *texte*.

`\normalsize`
`\@normalsize`

La commande `\@normalsize` est conservée pour la compatibilité avec les packages LaTeX 2.09 qui peuvent avoir utilisé sa valeur ; toutefois, la redéfinir dans un fichier de classe n'aura aucun effet car elle est toujours réinitialisée pour avoir la même signification que `\normalsize`.

Cela veut dire que les classes *doivent* désormais redéfinir `\normalsize` plutôt que de redéfinir `\@normalsize` ; par exemple (exemple plutôt incomplet) :

```
\renewcommand{\normalsize}{\fontsize{10}{12}\selectfont}
```

Notez que `\normalsize` est défini par le noyau LaTeX pour être un message d'erreur.

`\tiny` `\footnotesize` `\small` `\large`
`\Large` `\LARGE` `\huge` `\Huge`

Aucune de ces autres commandes de changement de taille « standard » ne sont définies dans le noyau : chacune d'entre elles doit être définie dans un fichier de classe si vous en avez besoin. Elles sont toutes définies par les classes standard.

Cela signifie que vous devriez utiliser `\renewcommand` pour `\normalsize` et `\newcommand` pour les autres commandes de changement de taille.

6.5 Commandes obsolètes

Certains packages ne marcheront pas avec LaTeX 2_ε, normalement parce qu'ils comptaient sur une commande interne LaTeX qui n'était pas garantie ou documentée et a été depuis changée ou retirée.

Dans beaucoup de cas, il y aura désormais un moyen robuste de haut niveau pour réaliser ce qui avant nécessitait des commandes bas-niveau. Veuillez consulter la section 4 pour voir si vous pouvez utiliser les commandes LaTeX 2_ε d'auteur de classe et package.

De plus, si votre package ou classe redéfinit une des commandes du noyau (i.e. celles définies dans les fichiers `latex.tex`, `slitex.tex`, `lfonts.tex`, `sfonts.tex`), alors il vous faudra bien sûr vérifier attentivement si l'implémentation n'a pas changé ou si la commande existe encore dans le noyau LaTeX 2_ε.

Trop des commandes internes de LaTeX 2.09 ont été réimplémentées ou retirées pour les lister toutes ici. Il vous faut vérifier celles que vous avez utilisées ou changées.

Nous allons, toutefois, lister quelques-unes des commandes les plus importantes qui ne sont plus gérées.

`\tenrm` `\elvrm` `\twlrm` ...
`\tenbf` `\elvbf` `\twlbf` ...
`\tensf` `\elvsf` `\twlsf` ...
⋮

Les soixante-dix (approximativement) commandes internes de cette forme ne sont plus définies. Si votre classe ou package les utilise, alors *s'il vous plaît*, veuillez les remplacer par les nouvelles commandes de fontes décrites dans *LaTeX 2ε Font Selection*.

Par exemple, la commande `\twlsf` devrait être remplacée par :

```
\fontsize{12}{14}\normalfont\sffamily\selectfont
```

Une autre possibilité est d'utiliser le package `rawfonts`, décrit dans *LaTeX 2ε for Authors*.

Souvenez-vous en outre que beaucoup des fontes pré-chargées par LaTeX 2.09 ne le sont plus.

`\vpt \vipt \viipt ...`

Ces commandes étaient des commandes internes de sélection de taille de LaTeX 2.09. (Elles peuvent être encore utilisées en mode compatibilité avec LaTeX 2.09.) Veuillez utiliser la commande `\fontsize` à la place : voir *LaTeX 2ε Font Selection* pour plus de détails.

Par exemple, `\vpt` devrait être remplacé par :

```
\fontsize{5}{6}\normalfont\selectfont
```

`\prm, \pbf, \ppounds, \pLaTeX ...`

LaTeX 2.09 utilisait plusieurs commandes commençant par `\p` afin de fournir des commandes « protégées » . Par exemple, `\LaTeX` était défini par `\protect\pLaTeX`, et `\pLaTeX` était défini pour produire le logo LaTeX. Ceci faisait de `\LaTeX` une commande robuste, même si `\pLaTeX` ne l'était pas.

Ces commandes ont été désormais réimplementées en utilisant `\DeclareRobustCommand` (décrite dans la Section 4.10). Si votre package redéfinit une des commandes `\p`, il vous faut alors retirer la redéfinition et utiliser `\DeclareRobustCommand` pour redéfinir la commande non-`\p`.

```
\footheight
\@maxsep
\@dblmaxsep
```

Ces paramètres ne sont pas utilisés par LaTeX 2ε ; ils ont donc été retirés, sauf en mode compatibilité LaTeX 2.09. Les classes ne doivent plus les utiliser.

Références

[1] Michel Goossens, Frank Mittelbach, et Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[2] Donald E. Knuth. *The TeXbook*. Addison-Wesley, Reading, Massachusetts, 1986. Révisé pour couvrir TeX3, 1991.

[3] Leslie Lamport. *LaTeX : A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, seconde édition, 1994.

Page de résumé LaTeX 2_ε : remettre à jour d'anciens styles

Les sections de références ci-dessous sont celles du *LaTeX 2_ε for Class and Package Writers*[3].

1. Ce style doit-il devenir une classe ou un package ? Voir la section 2.3 pour savoir comment répondre à cette question.
2. S'il utilise un autre fichier de style, alors il vous faudra obtenir une version remise à jour de cet autre fichier. Voir à la section 2.7.1 pour informations sur comment charger d'autres fichiers de classe et de package.
3. Essayez-le : voir la Section 6.1.
4. Ça a marché ? Excellent, mais il vous reste probablement des choses à changer afin de faire de votre fichier un fichier LaTeX 2_ε bien structuré, et à la fois robuste et portable. Il vous faut donc à présent lire la Section 2, et plus particulièrement 2.7. Vous trouverez également des exemples utiles dans la Section 3.

   Si votre fichier prépare de nouvelles fontes, des commandes de changement de fontes ou des symboles, vous devriez également lire *LaTeX 2_ε Font Selection*.
5. Ça n'a pas marché ? Il y alors trois possibilités :
   - des messages d'erreur sont produits pendant la lecture de votre fichier ;
   - des messages d'erreur sont produits pendant le traitement des documents tests ;
   - il n'y a pas d'erreur, mais la sortie n'est pas ce qu'elle devrait être.

   N'oubliez pas de vérifier attentivement cette dernière éventualité.

   Si vous en êtes à cette étape, il vous faut alors lire la Section 6 pour trouver les solutions qui feront marcher votre fichier.

[3] *NdT :* « LaTeX 2_ε pour les auteurs de classes et packages », le présent document