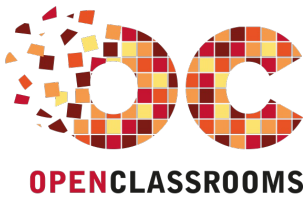
Créez votre propre package

Par sp.conductos



www.openclassrooms.com

Dernière mise à jour le 14/12/2009

Sommaire

Créez votre propre package

Par  sp.conductos

Mise à jour : 14/12/2009
Difficulté : Intermédiaire

Bonjour chers amis ZérOs !

Utilisateurs du système de composition typographique LaTeX, vous en avez assez de saisir toujours les mêmes lignes dans le préambule de votre document ? Sachez qu'il est possible de créer votre propre package (ou dans un langage plus français : extension) dans lequel vous chargerez vos extensions, commandes et environnements favoris, afin de vous rendre la vie beaucoup plus facile !

Prêts à franchir un nouveau pas dans votre utilisation du merveilleux système qu'est TeX (et sa *surcouche* LaTeX) ?

Suivez le guide...

Dans la suite du tutoriel, par pure fainéantise, j'écrirai TeX et LaTeX, respectivement à la place de TeX et LaTeX.

Sommaire du tutoriel :

Structure de base

Le code minimal

Tout d'abord, commençons par la structure minimale de toute extension LaTeX 2ε (pour les novices, prononcez la tek 2 epsilon) :

Code : TeX

```
\NeedsTeXFormat{LaTeX2e}[1999/01/01]
\ProvidesPackage{nomPackage}[2007/11/26]

%chargement des extensions requises au bon fonctionnement de l'extension et des

%déclaration des options de l'extension
\ProcessOptions

%commandes et/ou environnements personnalisés

\endinput
```

Analyse du code minimal

Afin de mieux comprendre, je vous propose une étude ligne par ligne du code.

Spécifications de base

Code : TeX

```
\NeedsTeXFormat{LaTeX2e}[1999/01/01]
```

On spécifie tout d'abord la version de LaTeX nécessaire au fonctionnement de l'extension. Dans ce cas, on demande la version 2e soit 2_ε. L'argument optionnel (entre crochets) précise la date minimale de sortie du moteur TeX (ou des macros LaTeX permettant l'utilisation de l'extension créée). Ceci n'est nécessaire que si vous souhaitez utiliser des commandes seulement présentes dans des versions bien précises des moteurs TeX ou LaTeX.

Aux utilisateurs de $\LaTeX 2.09$:
il est actuellement fortement déconseillé d'utiliser cette version du système de macros LaTeX, car de plus en plus d'extensions sont inaccessibles et l'utilisation est bien moins confortable pour de nombreuses actions.

Code : TeX

```
\ProvidesPackage{monPackage}[2007/11/26]
```

Ceci est une ligne essentielle. En effet, c'est elle qui détermine le nom de l'extension que vous créez. Dans ce cas, le nom est très original : *monPackage*. Une fois de plus, l'argument optionnel permet de préciser la date de création de l'extension.
Ainsi, pour utiliser votre extension dans votre document, vous écrirez :

Code : TeX

```
\usepackage{monPackage}
```

Chargement des extensions

Code : TeX

```
%chargement des extensions requises au bon fonctionnement de l'extension et des c
```

Eh ! C'est quoi cette ligne en commentaire ? Tu crois vraiment que le compilateur va la lire ?

Bien sûr que non ! Les marqueurs de commentaires ne varient pas entre les fichiers sources et les fichiers d'extensions. Je ne voulais simplement pas vous donner d'exemple et donner directement la commande sans argument n'aurait pas été très explicite. Je vous propose maintenant la syntaxe correcte à utiliser :

Code : TeX

```
\RequirePackage[<options]{<package>}
```

Comme on peut aisément le remarquer, *\RequirePackage[]{}* possède la même syntaxe que *\usepackage[]{}* que l'on utilise dans les fichiers sources "classiques".

Veillez à bien charger **toutes** les extensions nécessaires au fonctionnement de votre extension. Je pense notamment aux

extensions du type *amsmath* si vous définissez dans votre extension une commande utilisant la famille *\mathbb{}*.

Options

Code : TeX

```
%déclaration des options de l'extension
```

Afin de déclarer les options que vous comptez utiliser, on procède comme suit.

1. **Options définies**

Code : TeX

```
\DeclareOption{<nom de l'option>}{<action>}
```

Dans *<nom de l'option>*, on met le mot clef à indiquer dans les arguments optionnels de la commande *\usepackage[]{}*.
Dans *<action>*, on indique l'action à réaliser si l'option est demandée.

C'est bien beau tout ça, mais cela ne m'indique pas ce qu'il faut mettre dans *<action>*.

En fait, on met bien souvent (voire toujours) un changement de valeur de booléen. Je m'explique : imaginons que je veuille rendre l'utilisation de l'extension *hyperref* (pour créer des liens hypertextes et bien d'autres choses dans mon document) optionnelle. Je crée alors un booléen auquel j'affecte la valeur *false* :

Code : TeX

```
\RequirePackage{ifthen}%pour utiliser les booléens, à mettre avec les autres Requ

\newboolean{booleenH}%création du booléen
\setboolean{booleenH}{false}%affectation de la valeur false
```

Ensuite, lorsque je déclare mon option, j'écris :

Code : TeX

```
\DeclareOption{optionH}{\setboolean{booleenH}{true}}
```

Donc, si l'option est demandée, on affecte à mon booleenH la valeur *true*.

Enfin, après le *\ProcessOptions*, je charge l'extension *hyperref*, uniquement si l'option *optionH* est demandée :

Code : TeX

```
\ifthenelse{\boolean{booleenH}%
}{\RequirePackage{hyperref}%
}{}
```

Remarque : une autre solution est présentée dans la suite du tutoriel.

2. **Options indéfinies**

Imaginons à présent que vous êtes polyglottes. Vous aurez donc besoin de charger des options de *babel* différentes (eh oui, même *babel* est chargé par votre extension personnelle !). Dans ce cas, on ne peut pas déclarer les options comme indiqué ci-dessus. On utilise donc un joker : l'étoile (*).

Code : TeX

```
\DeclareOption*{action}
```

Évidemment, il n'y a pas de champ <*nom de l'option*> à compléter !
Dans le champ <*action*>, on utilise bien souvent la ligne suivante :

Code : TeX

```
\DeclareOption*{\PassOptionToPackage{\CurrentOption}{<package>}}
```

Ce qui permet de passer l'ensemble des options non déclarées explicitement à l'extension <*package*> souhaitée.

L'extension voulue doit être chargée avec un *(\RequirePackage[]{})* ***après*** *la commande \DeclareOption*{}.*

Puis vient la validation des options avec :

Code : TeX

```
\ProcessOptions
```

Commandes et environnements personnalisés

La dernière partie de votre extension est consacrée à la définition de vos propres commandes et environnements. On utilise pour ce faire les deux commandes suivantes, respectivement pour les commandes et les environnements :

Code : TeX

```
\newcommand{<\nomCommande>}[<nombreArguments]{<code>}
```

et :

Code : TeX

```
\newenvironnment{<nomEnvironnement>}{<clausesDébut>}{<clausesFin>}
```

La création de commandes et d'environnements n'étant pas le sujet de ce tutoriel, je ne m'étendrai pas d'avantage.

Et pour finir...

Eh bien, on indique au compilateur que c'est fini avec :

Code : TeX

```
\endinput
```

Et voilà !

L'astuce babel

Comme nous l'avons vu dans la sous-partie précédente, il faut recourir au joker étoilé pour optimiser le chargement de l'extension

babel. Je vous propose ici d'étudier plus en détail ce que j'ai baptisé "l'astuce babel", afin d'en maîtriser toutes les facettes.

Gestion des langues : ce qu'il faut savoir

Vous le savez peut-être : l'extension *babel*, habituellement chargée pour nous francophones, est représentée par l'un des codes suivants :

Code : TeX

```
\usepackage[francais]{babel}
%ou
\usepackage[frenchb]{babel}
%ou encore
\usepackage[frenchle]{babelfr}
%liste non exhaustive
```

Elle permet d'adapter différents paramètres (traduction des titres, gestion de la typographie, etc.) à la langue choisie. Cependant, *babel* ne se contente pas de gérer une seule langue à la fois. En effet, il est possible d'utiliser plusieurs langues dans un même document.
Imaginons, par exemple, que je veuille écrire en français, en anglais et en allemand. Dans une source normale, j'écrirai :

Code : TeX

```
\usepackage[deutsch,english,français]{babel}
```

Ce qui aura pour effet de charger les modules de traduction en anglais, en allemand et en français.

Mais il y a un problème. Pour les chapitres par exemple, il sera marqué ceci : Kapitel, Chapter, Chapitre 1 ?

Euh non ! Ce serait d'ailleurs bien embêtant ! En fait, c'est la dernière langue demandée (ici *français*) qui est appliquée par défaut. Il sera donc marqué, si je ne précise rien : Chapitre 1. Ouf !

Changement de langue

Pour changer de langue, *babel* offre plusieurs possibilités :

- un environnement :
  Code : TeX
  ```
  \begin{otherlanguage}{<langue>}
  \end{otherlanguage}
  ```
- une commande :
  Code : TeX
  ```
  \foreignlanguage{<langue>}{<texte dans la langue <langue>>}
  ```
- une déclaration :
  Code : TeX
  ```
  \selectlanguage{<langue>}
  ```

Dans tous les cas, la langue *(<langue>)* indiquée en argument doit avoir été chargée par *babel*.

Utilisation de *\DeclareOption*{}*

Nous allons à présent voir comment on peut utiliser intelligemment la commande *\DeclareOption*{}*.
Je vous propose le code suivant que je commenterai brièvement ensuite :

Code : TeX

```
%tout ce qui doit être là...
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{babel}}
\ProcessOptions
%ligne vide pour y voir plus clair
\RequirePackage{babel}
```

Comme je l'avais déjà proposé précédemment, on demande à passer l'ensemble des options non déclarées explicitement, c'est-à-dire avec une commande *\DeclareOption{}{}*, au package *babel*. Celui-ci doit donc être chargé **après**, comme indiqué sur le code ci-dessus et **sans aucune option**.

J'insiste bien sur le fait qu'il ne faut surtout pas indiquer d'options à l'extension *babel* dans la ligne :

Code : TeX

```
\RequirePackage{babel}
```

Sinon, vous risquez d'avoir un superbe message d'erreur :

Code : Console

```
! LaTeX Error : Option clash for package babel
```

Ce qui signifie que vous avez tenté de charger l'extension *babel* avec des options différentes, à deux moments différents. Et bien sûr, la compilation plante !

Remarques concernant l'utilisation de "l'astuce babel".

Il existe cependant un désavantage majeur à l'utilisation de cette astuce qui est néanmoins très utile. En effet, vous êtes obligés de donner une option de langage à votre extension. De plus, n'oubliez pas que c'est la dernière langue entrée qui est utilisée par défaut dans votre document. D'autre part, il est ensuite impossible de passer d'autres options non déclarées explicitement à d'autres packages que *babel*.
Exemple : vous voulez utiliser les options non déclarées explicitement pour les extensions *hyperref* et *babel*. Dans la source de votre document, vous écrivez alors :

Code : TeX

```
\usepackage[francais,colorlinks=true]{mon_package}
```

Et, lors de la compilation, vous obtenez :

Code : Console

```
! Package babel Error : Language definition file colorlinks.ldf not found
```

Autrement dit, si vous ne voulez pas avoir d'erreur de compilation, il ne vous reste plus qu'à inventer la langue *colorlinks*  !

Conditions TeX, conditions LaTeX

Nous allons, pour finir, aborder un point un peu plus technique, mais pas de quoi s'arracher les cheveux, rassurez-vous. Il s'agit simplement d'approfondir un point essentiel de la création d'extensions, que j'ai auparavant quelque peu estompé : les conditions.

Les conditions en TeX

Le système TeX de base utilise le système de conditions suivant :

Code : TeX

```
\newif \if@ca \@ifcatrue
%
%
\if@ca
%commandes
\fi
```

Explications
On commence par créer une variable de type *if* avec la commande :

Code : TeX

```
\newif
```

Cette variable possède pour nom :

Code : TeX

```
\if@ca
```

Soit en français :

Code : Autre

```
si "ça" est ...
```

Ben oui, il faut encore assigner une valeur à notre variable :

Code : TeX

```
\@ifcatrue
```

Dans notre cas, on assigne à notre variable la valeur *true*, soit *vrai*.

Les programmeurs, tous langages confondus, reconnaîtront ici un booléen. La logique booléenne est à la base de tous les langages de programmation, mais également de la logique en général. C'est cette logique que l'on utilise lorsqu'on effectue des associations de conditions avec les opérateurs logiques AND ("et"), OR ("ou"), XOR ("ou" exclusif) et NOT (négation).
Sa formalisation est appelée algèbre de Boole.

Enfin, le code devant s'exécuter lorsque la condition est vraie est dans le bloc :

Code : TeX

```
\if@ca
\fi
```

Application aux extensions

Il est possible d'utiliser ce système de conditions dans les extensions, notamment dans la gestion des options.
Exemple :

Code : TeX

```
\newif \if@couleur \@couleurfalse
\DeclareOption{color}{\@couleurtrue}
%...
```

```
\if@couleur
\RequirePackage{xcolor}
\fi
```

On charge l'extension *xcolor* si l'option *color* est envoyée à l'extension personnelle.

Les conditions en LaTeX

Les conditions en LaTeX utilisent une syntaxe beaucoup plus familière.
Pour commencer, il faut charger l'extension *ifthen* :

Code : TeX

```
\RequirePackage{ifthen}
```

Ensuite, on applique le code suivant :

Code : TeX

```
\newboolean{couleur}
\setboolean{couleur}{false}
\DeclareOption{color}{\setboolean{couleur}{true}}
%...
\ifthenelse{\boolean{couleur}}{%
\RequirePackage{xcolor}}{}
```

J'utilise ici le même exemple que précédemment : on charge l'extension *xcolor* si l'option *color* est envoyée à l'extension personnelle.
En LaTeX, *surcouche* de TeX, l'utilisation de booléens est beaucoup plus évidente. En effet, le booléen est créé par la commande :

Code : TeX

```
\newboolean{<nom>}
```

On lui affecte une valeur avec :

Code : TeX

```
\setboolean{<nom>}{<valeur>}
```

Et pour finir, on teste sa valeur avec :

Code : TeX

```
\boolean{<nom>}
```

Dans une grosse commande :

Code : TeX

```
\ifthenelse{<condition>}{<code si vrai>}{<code si faux>}
```

On pourrait reprocher à la syntaxe LaTeX d'être très lourde (ceci est visible notamment dans la comparaison des deux codes donnant le même résultat présentés ci-dessus), mais il est bon de connaître les deux syntaxes. La syntaxe TeX est majoritairement utilisée dans les squelettes des extensions et des classes, la syntaxe LaTeX l'est beaucoup plus dans les environnements, notamment car elle permet d'y voir plus clair. Enfin, chacun choisit celle qu'il préfère.

Utilisation de votre extension

C'est bien beau de coder une extension, encore faudrait-il pouvoir l'utiliser ! Pour cela, suivez le guide.

Enregistrement

En suivant une architecture TeX classique, on enregistre les extensions personnelles dans le répertoire suivant (ici sous Windows avec une distribution TeXlive 2007) :

Code : Console

```
C:\TeXLive2007\texmf\tex\latex\perso
```

Pour les détenteurs d'autres distributions, je pense notamment à nos amis linuxiens, veillez à utiliser un répertoire similaire (le nom *perso* n'est pas obligatoire, vous pouvez choisir autre chose, mais sans espace de préférence). Vous trouverez quelques informations complémentaires pour l'utilisation sous Linux plus bas.

Et quelle extension dois-je mettre ?

J'allais l'oublier ! Effectivement, il ne faut pas enregistrer l'extension en **.tex* qui est réservée aux sources de documents. L'extension étant un ensemble de paramètres stylistiques, on met l'extension **.sty* (de l'anglais *style*).

Mise à jour de la base Is-R

Seconde et donc dernière étape, il s'agit de mettre à jour la base de données Is-R. Pour faire très simple, cette base de données regroupe tous les chemins d'accès aux extensions, aux classes et aux polices (entres autres) de votre distribution. Ainsi, si vous voulez que le compilateur connaisse votre extension, il faut qu'elle soit répertoriée dans Is-R.
Voici comment procéder (sous TeXLive 2007) :

- ouvrez TeXlive Manager ;
- allez dans l'onglet "Gestion de l'installation" ;
- cliquez sur "Mettre à jour", dans le champ "Mettre à jour la base de donnée Is-R" ;
- patientez...
- c'est fini, vous pouvez fermer TeXlive Manager.

Pour les autres distributions, une démarche similaire est à effectuer.

Pour les utilisateurs de Linux

La distribution LaTeX traditionnellement utilisée sous Linux est teTeX. Cependant, elle l'est de moins en moins et cède peu à peu la main à TeXLive du fait que cette dernière est bien plus maintenue (source). Je vous recommande donc, si vous ne disposez pas encore de distribution LaTeX sous Linux, d'installer le paquet texlive (utilisateurs d'Ubuntu, c'est par ici).

La distribution installée, vous devez enregistrer votre extension personnelle dans le répertoire suivant (le *perso* n'est à nouveau pas obligatoire) :

Code : Console

```
/usr/share/texmf-texlive/tex/latex/perso/
```

Il s'agit ensuite, comme sous Windows, de mettre à jour la base de données Is-R. Pour cela, ouvrez votre émulateur de console favori et tapez :

Code : Console

```
sudo texhash
```

Et voilà, votre extension personnelle est utilisable !

Utilisation dans un document

Votre extension personnelle s'utilise à présent de la même manière que toute autre extension. On la charge dans le préambule de la source, avec un :

Code : TeX

```
\usepackage[]{}
```

Tout ce qu'il y a de plus normal.

Un fichier **.sty* ne se compile pas. Ce n'est pas grave si vous le faites : vous aurez simplement droit à une belle erreur ! Pour enregistrer les modifications de votre extension, enregistrez-le simplement.

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.
Pour accéder aux suivants

Connectez-vous Inscrivez-vous

Quelle est la bonne commande pour charger une extension dans un fichier de style (*.sty) ?

- ◯ \usepapckage[]{}
- ◯ \package{}
- ◯ \RequirePackage[]{}
- ◯ \RequirePackage[]
- ◯ \package[]{}

Le code suivant est-il correct ?

Code : TeX

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{package}
\RequirePackage[latin1]{inputenc}
\RequirePackage[T1]{fontenc}
\RequirePackage{pifont}
\RequirePackage{beton}
\RequirePackage{amsmath}
\RequirePackage{amsfonts}
\RequirePackage{ifthen}
\newboolean{signature}
\setboolean{signature}{false}
\DeclareOption{signe}{\setboolean{signature}{true}}
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{babel}}
\RequirePackage{babel}
\newcommand{\moi}{\textsc{sp.conductos}}
\newcommand{\R}{\mathbb{R}}
\endinput
```

- ◯ Oui.
- ◯ Non.

Pour finir, la question de culture générale :

le code suivant peut-il exister ?

Code : TeX

```
\usepackage[col=5,lig=12]{package}
```

- ◯ Oui.
- ◯ Non.

Correction !

Statistiques de réponses au QCM

Ce tutoriel s'achève ici. J'espère avoir été suffisamment clair dans l'explication de toutes les étapes de la création d'une extension personnalisée, et bien sûr, que cela vous soit utile !
Pour tous renseignements complémentaires, merci de vous tourner vers les commentaires.

sp.conductos

Ce tutoriel a été corrigé par les zCorrecteurs.