L'extension xspace*

David Carlisle Morten Høgholm

28/10/2014

Résumé

La commande `\xspace` s'utilise à la fin de commandes pensées pour être intégrées principalement dans du texte. Elle ajoute là une espace à moins que la commande ne soit suivie par certains signes de ponctuation.

1 Introduction

`\xspace` Après avoir défini `\newcommand{\gb}{La Grande Bretagne\xspace}`, la commande `\gb` détermine si une espace doit être insérée après elle ou pas. Ainsi, la saisie

```
\gb est un très bel endroit pour vivre.\\
\gb, une petite île au large des côtes françaises.\\
\gb\footnote{La petite île au large des côtes françaises.}
est un très bel endroit pour vivre.
```

conduit au résultat suivant

La Grande Bretagne est un très bel endroit pour vivre.
La Grande Bretagne, une petite île au large des côtes françaises.
La Grande Bretagne[1] est un très bel endroit pour vivre.

`\xspace` évite à l'utilisateur de saisir `\␣` ou `{}` après la plupart des occurrences d'une commande dans du texte. Cependant, si l'une de ces deux constructions suit `\xspace`, ce dernier n'ajoute pas d'espace. Ceci implique qu'il est possible d'ajouter `\xspace` à la fin de commandes déjà présentes sans risques et sans faire trop de changements dans votre document. En particulier, `\xspace` insérera toujours une espace si l'élément qui le suit est une lettre normale, ce qui est le cas classique.

*Ce fichier a pour numéro de version v1.13 et a été mis à jour le 28/10/2014. Son titre original est « *The xspace package* ». La première traduction, basée sur la version 1.06 a été publiée par Jean-Drucbert en 2001.

1. La petite île au large des côtes françaises.

Parfois, \xspace peut prendre une mauvaise décision en ajoutant une espace non souhaitée. Il y a différentes raisons à ce comportement mais ce problème peut toujours être traité en faisant suivre la commande d'un {}, dans la mesure où ceci a pour effet de supprimer l'espace.

1.1 Ajout de nouvelles exceptions

\xspaceaddexceptions

Une des raisons les plus courantes pour que \xspace insère une espace non souhaitée survient quand il est suivi d'une commande qui n'est pas dans sa liste d'exceptions. Avec \xspaceaddexceptions, vous pouvez ajouter (*add*) de nouveaux caractères ou commandes pour qu'ils soient reconnus par le mécanisme d'analyse de \xspace. Les utilisateurs d'extensions avancées sur les notes de bas de page comme manyfoot définissent souvent de nouvelles commandes de notes de bas de page : elles ne devraient pas impliquer l'ajout d'une espace à la suite d'une commande « améliorée » par \xspace. Aussi, Si vous définissez les commandes de note de page \footnoteA et \footnoteB, ajoutez la ligne suivante à votre préambule.

```
\xspaceaddexceptions{\footnoteA \footnoteB}
```

1.2 Support des caractères actifs

L'autre exemple courant que \xspace ne traite pas bien est lié aux caractères actifs. Généralement, cette extension doit être chargée *après* toute extension linguistique (ou autre) qui rend la ponctuation « active ». La tâche se complexifie tout particulièrement pour xspace lors de travaux avec l'extension populaire babel car les signes de ponctuation peuvent alors basculer du statut « actif » à « autre » et inversement. À partir de la version 1.08 de xspace, deux manières de gérer ce point sont disponibles en fonction du moteur utilisé par votre format LaTeX :

TeX Les signes de ponctuation sont ajoutés à la liste d'exceptions à la fois dans leur version normale et active, assurant ainsi qu'ils seront toujours reconnus.

ε-TeX Les caractères sont lus à nouveau lors du passage dans la liste des exceptions, ce qui signifie que la comparaison interne se fait sur l'état courant du caractère. Ceci fonctionne quelque soit l'astuce de code de catégorie utilisée par les autres extensions.

Au moment de la rédaction de ce document, toutes les grandes distributions TeX utilisent ε-TeX comme moteur pour LaTeX, ce qui fait que tout devrait bien se passer. S'il se trouve que vous utilisez le TeX standard et que \xspace semble prendre la mauvaise décision, alors vous pouvez soit utiliser {} comme décrit ci-dessus pour le corriger, soit ajouter le caractère à la liste mais avec le code de catégorie souhaité. Voir le détail de l'implémentation pour un exemple de ce qu'il faut alors faire.

1.3 Toujours pas satisfait ?

Certaines personnes n'aiment pas la liste d'exceptions, aussi peuvent-elles retrancher (*remove*) un par un des éléments de cette liste avec la commande dédiée `\xspaceremoveexception{`⟨*unité-lexicale*⟩`}`. Par ailleurs, la commande `\@xspace@hook` peut être définie pour analyser plus avant la suite du texte dans le cas où vous souhaiteriez vérifier plus d'unités lexicales. Elle est appelée une fois que `\xspace` a déterminé s'il faut insérer une espace ou si une exception a été trouvée (la définition par défaut de `\@xspace@hook` est vide). Par conséquent, vous pouvez utiliser `\unskip` pour retirer l'espace insérée si `\@let@token` rencontre quelque chose de spécial. L'exemple ci-dessous montre comment garantir qu'un tiret demi-quadratin [2] placé après un `\xspace` obtienne une espace mais que ce ne soit pas le cas avec un trait d'union.

`\xspaceremoveexception`
`\@xspace@hook`

```
\xspaceremoveexception{-}
\makeatletter
\renewcommand*\@xspace@hook{%
  \ifx\@let@token-%
    \expandafter\@xspace@dash@i
  \fi
}
\def\@xspace@dash@i-{\futurelet\@let@token\@xspace@dash@ii}
\def\@xspace@dash@ii{%
  \ifx\@let@token-%
  \else
    \unskip
  \fi
  -%
}
\makeatother
```

2 Les commandes

`\xspace` jette un coup d'œil à l'unité lexicale qui suit. Si elle appartient à notre liste d'exception, `\xspace` sort de la boucle d'analyse et ne fait rien ; sinon il essaye de développer l'unité lexicale et recommence ensuite son analyse. Si ceci conduit à une unité lexicale non développable sans qu'une exception ait été trouvée, une espace est insérée.

1 ⟨∗package⟩

`\xspace` `\xspace` regarde juste un cran en avant et appelle alors `\@xspace`.

```
2 \DeclareRobustCommand\xspace{\@xspace@firsttrue
3   \futurelet\@let@token\@xspace}
```

`\if@xspace@first` `\@xspace@simple` Quelques aides pour éviter des appels multiples de `\@xspace@eTeX@setup`.

2. Ce tiret est codé par -- en LaTeX.

```
\newif\if@xspace@first
\def\@xspace@simple{\futurelet\@let@token\@xspace}
```

`\@xspace@exceptions@tlp` La liste d'exceptions. Si le mécanisme d'analyse trouve une de ces exceptions, il n'y a pas insertion d'une espace après la commande. Le `tlp` dans le nom de la commande signifie « pointeur de liste d'unité lexicale » (*token list pointer*).

```
\def\@xspace@exceptions@tlp{%
  ,.'/?;:!~-)\ \/\bgroup\egroup\@sptoken\space\@xobeysp
  \footnote\footnotemark
  \xspace@check@icr
}
```

Ici, nous avons la définition non vide de `\check@icr`.

```
\begingroup
  \text@command\relax
  \global\let\xspace@check@icr\check@icr
\endgroup
```

`\xspaceaddexceptions` La commande utilisateur qui permet d'ajouter (*add*) des unités lexicales à la liste.

```
\newcommand*\xspaceaddexceptions{%
  \g@addto@macro\@xspace@exceptions@tlp
}
```

`\xspaceremoveexception` Cette commande retire (*remove*) une exception globalement.

```
\newcommand*\xspaceremoveexception[1]{%
```

Il faut d'abord vérifier si elle est bien dans la liste.

```
  \def\reserved@a##1#1##2##3\@@{%
    \@xspace@if@q@nil@NF##2{%
```

Elle est dans la liste : elle est alors retranchée.

```
      \def\reserved@a####1#1####2\@@{%
        \gdef\@xspace@exceptions@tlp{####1####2}}%
      \expandafter\reserved@a\@xspace@exceptions@tlp\@@
    }%
  }%
  \expandafter\reserved@a\@xspace@exceptions@tlp#1\@xspace@q@nil\@@
}
```

`\@xspace@break@loop` Pour arrêter (*break*) la boucle (*loop*).

```
\def\@xspace@break@loop#1\@nil{}
```

`\@xspace@hook` Un point d'entrée (*hook*) pour les utilisateurs avec des besoins spéciaux.

```
\providecommand*\@xspace@hook{}
```

Ici, nous vérifions si nous utilisons ε-TEX. Nous ne pouvons utiliser la commande `\@ifundefined` car elle bloque les codes de catégorie alors que nous avons besoin de les modifier. Comme il y a un risque que quelqu'un ait défini par accident `\eTeXversion` comme valant `\relax`, nous vérifions ce cas précis sans pour autant définir la commande en tant que `\relax` si ce n'était pas le cas.

```
\begingroup\expandafter\expandafter\expandafter\endgroup
  \expandafter\ifx\csname eTeXversion\endcsname\relax
```

Si nous utilisons une version classique de TEX, nous ajoutons les cas les plus courants de signes de ponctuation actifs. Nous les rendons tout d'abord actifs.

```
  \begingroup
    \catcode`\;=\active  \catcode`\:=\active
    \catcode`\?=\active  \catcode`\!=\active
```

L'environnement `alltt` rend « , », « ’ » et « - » actifs ; nous les ajoutons donc également.

```
    \catcode`\,=\active  \catcode`\’=\active  \catcode`\-=\active
    \xspaceaddexceptions{;:?!,’-}
  \endgroup
  \let\@xspace@eTeX@setup\relax
```

`\@xspace@eTeX@setup` Quand nous utilisons ε-TEX, nous avons l'avantage de disposer de la commande `\scantokens` qui va ici repasser en revue les codes de catégorie courants. Cette petite astuce pour le développement des unités lexicales garantit que la liste d'exceptions est redéfinie avec son contenu en tenant compte de la valeur présente des codes de catégorie. C'est pourquoi nous faisons en sorte que le code de catégorie de l'espace soit 10 : nous avons en effet un \␣ dans la liste.

```
\else
  \def\@xspace@eTeX@setup{%
    \begingroup
      \everyeof{}%
      \endlinechar=-1\relax
      \catcode`\ =10\relax
      \makeatletter
```

Nous pouvons aussi avoir la malchance de procéder à la relecture de la liste au moment où les codes de catégorie de \, { et } sont « autres », par exemple quand, dans un en-tête, la routine de sortie est appelée au milieu d'un environnement `verbatim`.

```
      \catcode`\\\z@
      \catcode`\{\@ne
      \catcode`\}\tw@
      \scantokens\expandafter{\expandafter\gdef
        \expandafter\@xspace@exceptions@tlp
        \expandafter{\@xspace@exceptions@tlp}}%
    \endgroup
  }
\fi
```

`\@xspace` Si l'unité lexicale suivante appartient à une liste de caractères, rien n'est fait, sinon une espace est ajoutée. Avec la version 1.07, l'approche a été modifiée radicalement pour passer à travers la liste d'exception `\@xspace@exceptions@tlp` et vérifier chaque unité lexicale une par une.

```
\def\@xspace{%
```

Avant de commencer à vérifier la liste d'exceptions, il est intéressant de procéder à une vérification rapide de l'unité lexicale en question. La plupart du temps, `\xspace` se trouve dans du texte normal et `\@let@token` est une lettre. Dans ce cas, il ne sert à rien de vérifier la liste d'exceptions car elle ne contiendra pas d'unité lexicale de catégorie 11.

Vous pourriez ici vous demander pourquoi il y a ici des fonctions spéciales plutôt que de simples `\ifx` conditionnels. Les raisons à cela sont que a) de cette façon, nous n'avons pas à ajouter une multitude d'`\expandafter` pour obtenir le bon regroupement et b) nous ne rencontrons pas de problème quand `\@let@token` est égale à `\if`.

```
\@xspace@lettoken@if@letter@TF \space{%
```

Sinon nous commençons à tester après avoir défini quelques éléments. Si nous utilisons ε-TEX, nous repassons en revue les codes de catégorie mais seulement la première fois.

```
    \if@xspace@first
      \@xspace@firstfalse
      \let\@xspace@maybespace\space
      \@xspace@eTeX@setup
    \fi
    \expandafter\@xspace@check@token
      \@xspace@exceptions@tlp\@xspace@q@nil\@nil
```

Si une exception a été trouvée, `\@xspace@maybespace` est définie comme `\relax` et nous ne faisons rien.

```
    \@xspace@token@if@equal@NNT \space \@xspace@maybespace
```

Sinon nous vérifions si nous avons trouvé un élément développable et réessayons avec cette unité lexicale développée une fois. Si aucune unité développable n'est trouvée, nous insérons une espace et appliquons alors le code du point d'entrée.

```
    {%
      \@xspace@lettoken@if@expandable@TF
      {\expandafter\@xspace@simple}%
      {\@xspace@maybespace\@xspace@hook}%
    }%
  }%
}
```

`\@xspace@check@token` La commande compare juste l'élément courant dans la liste d'exceptions avec `\@let@token`. S'ils sont égaux, nous faisons en sorte qu'aucune espace ne soit insérée et nous sortons de la boucle.

```
\def\@xspace@check@token #1{%
  \ifx\@xspace@q@nil#1%
    \expandafter\@xspace@break@loop
  \fi
  \expandafter\ifx\csname @let@token\endcsname#1%
    \let\@xspace@maybespace\relax
    \expandafter\@xspace@break@loop
  \fi
```

```
  \@xspace@check@token
}
```

Et c'est tout pour aujourd'hui ! Du moins, si nous avions utilisé LaTeX3. Dans ce cas, nous aurions eu de belles fonctions pour toutes les conditions mais nous devons ici les définir nous-même. Nous les optimisons également car `\@let@token` sera toujours l'argument dans certains cas.

`\@xspace@lettoken@if@letter@TF` `\@xspace@lettoken@if@expandable@TF` `\@xspace@token@if@equal@NNT`

D'abord quelques comparaisons.

```
\def\@xspace@lettoken@if@letter@TF{%
  \ifcat\noexpand\@let@token @% letter
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
\def\@xspace@lettoken@if@expandable@TF{%
  \expandafter\ifx\noexpand\@let@token\@let@token%
    \expandafter\@secondoftwo
  \else
    \expandafter\@firstoftwo
  \fi
}
\def\@xspace@token@if@equal@NNT#1#2{%
  \ifx#1#2%
    \expandafter\@firstofone
  \else
    \expandafter\@gobble
  \fi}
```

`\@xspace@q@nil` `\@xspace@if@q@nil@NF`

Quelques commandes pour traiter les quarks.

```
\def\@xspace@q@nil{\@xspace@q@nil}
\def\@xspace@if@q@nil@NF#1{%
  \ifx\@xspace@q@nil#1%
    \expandafter\@gobble
  \else
    \expandafter\@firstofone
  \fi}
```

⟨/package⟩