L'extension xspace*

David Carlisle Morten Høgholm

28/10/2014

Ce fichier est maintenu par l'équipe du « LaTeX Project ».
Les rapports d'anomalie peuvent être envoyés en anglais à
`http://latex-project.org/bugs.html` (catégorie `tools`).

Résumé

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

1 Introduction

`\xspace` Après avoir défini `\newcommand{\gb}{Great Britain\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 constructions suit `\xspace`, ce dernier n'ajoute pas d'espace. Ceci implique qu'il est d'ajouter `\xspace` à la fin de commandes déjà présentes sans faire trop de changements dans votre document. En particulier, `\xspace` insèrera toujours une espace si l'élément qui le suit est une lettre normale, ce qui est cas classique.

Parfois, `\xspace` peut prendre une mauvaise décision et ajouter alors une espace non souhaitée. Il y a différentes raisons à ce comportement mais ce problème

*Ce fichier a pour numéro de version v1.13 et a été revu le 28/10/2014. 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.

peut toujours être géré 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 `\xspace` d'insérer une espace non souhaitée se produit quand il est suivi d'une commande qui n'est pas dans sa liste des exceptions. Avec `\xspaceaddexceptions`, vous pouvez ajouter de nouveaux caractères ou commandes pour qu'ils soient reconnus par la mécanique d'analyse de `\xspace`. Les utilisateurs d'extensions avancées sur les notes en bas de page comme manyfoot vont souvent définir de nouvelles commandes de notes de bas de page qui ne devraient pas impliquer l'ajout d'une espace à la suite d'une commande « améliorée » par `\xspace`. 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 où `\xspace` ne traite pas bien la situation se produit en présence de caractères actifs. Généralement, cette extension doit être chargée *après* toute extension linguistique (ou autre) qui rend la ponctuation « active ». Ceci complexifie la tâche pour xspace lors de travaux avec l'extension populaire babel tout particulièrement parce que les signes de ponctuation peuvent 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.

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 l'implémentation pour un exemple de ce qu'il faut alors faire.

1.3 Toujours pas satisfait ?

Certaines personnes n'aiment pas la liste des exceptions, aussi peuvent-ils retrancher un élément à la fois de cette liste avec une commande dédiée

\xspaceremoveexception
\@xspace@hook

`\xspaceremoveexception{`⟨*unité-lexicale*⟩`}`. De plus, 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] obtienne une espace tandis qu'un trait d'union non.

```
\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 suivante. 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 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`.

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

2. Ce tiret est codé par -- en LaTeX.

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

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

Et 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 s'il est bien dans la liste.

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

Il est dans la liste, il est alors retranché.

```
      \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 utilisteurs 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 et nous avons besoin de les modifier. Comme il existe un risque que quelqu'un ait défini par accident `\eTeXversion` comme valant `\relax`, nous vérifions ce cas précis mais sans définir pour autant la commande à `\relax` si elle ne l'était pas.

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

Si nous utilisons un TEX normal, 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. The `alltt` environment also makes `,`, `'`, and `-` active so we add them as well.

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

✖

`\@xspace@eTeX@setup` When we're running ε-TEX, we have the advantage of `\scantokens` which will rescan tokens with current catcodes. This little expansion trick makes sure that the exception list is redefined to itself but with the contents of it exposed to the current catcode regime. That is why we must make sure the catcode of space is 10, since we have a `\␣` inside the list.

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

We may also be so unfortunate that the re-reading of the list takes place when the catcodes of `\`, `{` and `}` are "other," e.g., if it takes place in a header and the output routine was called in the middle of a `verbatim` environment.

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

`\@xspace` If the next token is one of a specified list of characters, do nothing, otherwise add a space. With version 1.07 the approach was altered dramatically to run through the exception list `\@xspace@exceptions@tlp` and check each token one at a time.

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

Before we start checking the exception list it makes sense to perform a quick check on the token in question. Most of the time `\xspace` is used in regular text so `\@let@token` is set equal to a letter. In that case there is no point in checking the list because it will definitely not contain any tokens with catcode 11.

You may wonder why there are special functions here instead of simpler `\ifx` conditionals. The reason is that a) this way we don't have to add many, many `\expandafter`s to get the nesting right and b) we don't get into trouble when `\@let@token` has been let equal to `\if` etc.

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

Otherwise we start testing after setting up a few things. If running ε-TEX we rescan the catcodes but only the first time around.

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

If an exception was found `\@xspace@maybespace` is let to `\relax` and we do nothing.

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

Otherwise we check to see if we found something expandable and try again with that token one level expanded. If no expandable token is found we insert a space and then execute the hook.

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

`\@xspace@check@token` This macro just checks the current item in the exception list against the `\@let@token`. If they are equal we make sure that no space is inserted and break the loop.

```
\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 ! 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
space@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⟩