The xspace package*

David Carlisle Morten Høgholm

2014/10/28

This file is maintained by the LaTeX Project team.
Bug reports can be opened (category `tools`) at
https://latex-project.org/bugs.html.

Abstract

`\xspace` should be used at the end of a macro designed to be used mainly in text. It adds a space unless the macro is followed by certain punctuation characters.

1 Introduction

`\xspace` After defining `\newcommand{\gb}{Great Britain\xspace}`, the command `\gb` will determine when to insert a space after itself and when not. Thus the input

```
\gb is a very nice place to live.\\
\gb, a small island off the coast of France.\\
\gb\footnote{The small island off the coast of France.}
is a very nice place to live.
```

results in the output

Great Britain is a very nice place to live.
Great Britain, a small island off the coast of France.
Great Britain[1] is a very nice place to live.

`\xspace` saves the user from having to type `\␣` or `{}` after most occurrences of a macro name in text. However if either of these constructions follows `\xspace`, a space is not added by `\xspace`. This means that it is safe to add `\xspace` to the end of an existing macro without making too many changes in your document. In particular, `\xspace` will always insert a space if the thing following it is a normal letter which is the usual case.

Sometimes `\xspace` may make the wrong decision, and add a space when it is not required. There may be different reasons for this behavior but it can always be handled by following the macro with `{}`, as this has the effect of suppressing the space.

*This file has version number v1.13, last revised 2014/10/28.
[1]The small island off the coast of France.

1.1 Adding new exceptions

\xspaceaddexceptions

One of the most common reasons for \xspace to add an unwanted space is when it is followed by a macro not on its list of exceptions. With \xspaceaddexceptions you can add new commands or characters to be recognized by \xspace's scanning mechanism. Users of advanced footnote packages like manyfoot will often define new footnote macros that should not cause a command "enhanced" with \xspace to insert a space. If you define the additional footnote macros \footnoteA and \footnoteB, simply add the following line to your preamble:

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

1.2 Support for active characters

The other common instance of \xspace not working quite right happens with active characters. Generally this package must be loaded *after* any language (or other) packages that make punctuation characters 'active'. This makes it difficult for xspace to work flawlessly with the popular babel package especially since the punctuation characters can switch between being 'active' and 'other'. Starting at xspace version 1.08 there are two different ways to handle this depending on which engine your LaTeX-format uses:

TeX The punctuation characters are added to the exception list in both their normal and active states thus ensuring that they are always recognized.

ε-TeX The characters are re-read when going through the exception list which means the internal comparison will test against the current state of the character. This works for whatever category code tricks some packages may use.

At the time of writing all major TeX distributions are using ε-TeX as engine for LaTeX so usually everything should work out of the box. If you find that you're running normal TeX and \xspace seems to be making the wrong choice then either use {} as described above to fix it or add the character to the list but with the desired category code. See the implementation for an example of how to do that.

1.3 Still not satisfied?

\xspaceremoveexception
\@xspace@hook

Some people don't like the default list of exceptions so they can remove one item at a time with the command \xspaceremoveexception{⟨*token*⟩}. Furthermore the command \@xspace@hook can be redefined to scan forward in the input stream in case you want to check more tokens. It is called after \xspace has determined if it needed to insert a space or if an exception was found (the default definition is for \@xspace@hook to be empty). Hence you can use \unskip to remove the space inserted if \@let@token matches something special. Below is an example of how one can make sure an endash gets a space inserted before it but a single dash not.

```
\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 The Macros

`\xspace` peeks ahead for the next token. If the token is in our exception list we break the loop and do nothing; else we try to expand the token once and start over again. If this leads us to an unexpandable token without finding one of the exceptions we insert a space.

```
1 ⟨*package⟩
```

`\xspace` — `\xspace` just looks ahead, and then calls `\@xspace`.

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

`\if@xspace@first` `\@xspace@simple` — Some helpers to avoid multiple calls of `\@xspace@eTeX@setup`.

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

`\@xspace@exceptions@tlp` — The exception list. If the scanning mechanism finds one of these, it won't insert a space after the command. The `tlp` in the name means 'token list pointer.'

```
6 \def\@xspace@exceptions@tlp{%
7   ,.'/?;:!~-)\ \/\bgroup\egroup\@sptoken\space\@xobeysp
8   \footnote\footnotemark
9   \xspace@check@icr
10 }
```

And here we get the non-empty definition of `\check@icr`.

```
11 \begingroup
12   \text@command\relax
13   \global\let\xspace@check@icr\check@icr
14 \endgroup
```

`\xspaceaddexceptions` — The user command, which just adds tokens to the list.

```
15 \newcommand*\xspaceaddexceptions{%
16   \g@addto@macro\@xspace@exceptions@tlp
17 }
```

`\xspaceremoveexception` — This command removes an exception globally.

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

First check that it is in the list at all.

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

It's in the list, remove it.

```
      \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` To stop the loop.

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

`\@xspace@hook` A hook for users with special needs.

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

Now we check if we're running ε-TEX. We can't use `\@ifundefined` as that will lock catcodes and we need to change some of those. As there is a small risk that someone already set `\eTeXversion` to `\relax` by accident we make sure we check for that case but without setting it to `\relax` if it wasn't already.

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

If we are running normal TEX we add the most common cases of active punctuation characters. First we make them active.

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

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
}
```

That's all, folks! That is, if we were running LaTeX3. In that case we would have had nice functions for all the conditionals but here we must define them ourselves. We also optimize them here as `\@let@token` will always be the argument in some cases.

`@xspace@if@lettoken@letter@TF`
`ace@if@lettoken@expandable@TF`
`\@xspace@cs@if@equal@NNF`

First a few comparisons.

```
\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`

Some macros dealing with 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⟩