Abbreviations (abbrev.dtx)

Charles Duan

February 29, 2024

This package provides for abbreviating texts according to tables of abbreviations. It also provides functions for removing spaces between dotted initials. The abbreviation algorithms are fully expandable so that they may be used within `\edef` definitions.

```
\ProvidesPackage{abbrev}[2017/08/14 Abbreviations]
\RequirePackage{etoolbox}
\RequirePackage{strings}
```

1 Abbreviation Classes

There may be multiple abbreviation classes, namely separate schemes of abbreviations with different words abbreviated in each.

An abbreviation class is defined in the macro `\abb@class@`⟨*name*⟩. Within that class are a further set of macros of the form `\abb@class@`⟨*name*⟩`@`⟨*word*⟩, where ⟨*word*⟩ is the first word of the text to be abbreviated. The value of the macro is a list of two-item pairs of the form:

{⟨*remaining text*⟩}{⟨*abbrev.*⟩}

`\abb@new`{⟨*class*⟩} creates a new abbreviation class.

```
\def\abb@new#1{\@namedef{abb@class@#1}{}}%
```

`\abb@add`{⟨*class*⟩}{⟨*word*⟩}{⟨*abbrev*⟩} Adds an item to the abbreviation class. ⟨*class*⟩ is the class, ⟨*word*⟩ the unabbreviated form, and {⟨*abbrev*⟩} the abbreviated form.

If the unabbreviated form starts with an empty set of braces, then the given abbreviation pair is treated as an initial-text abbreviation, which will only abbreviate at the beginning of a text. That is, the abbrevation pairs:

{}Building ⟶ Bldg.
Building ⟶ Build.

would produce abbreviations as follows:

The Building and Loan Office ⟶ The Build. and Loan Office
Building and Loan Office ⟶ Bldg. and Loan Office

```
\def\abb@add#1#2#3{%
    \find@word{#2}{%
        \abb@add@punct{#1}{#3}%
    }{%
        \abb@add@group{#1}{#3}%
    }{%
        \abb@add@{#1}{#3}%
    }{%
        \abb@add@error{nothing}{}{}%
    }%
}
```

Processes an abbreviation pair where the unabbreviated text starts with a group. #1 is the class, #2 the abbreviated text, #3 the group contents, and #4 the remainder.

If the group contains any text, then this is an error. Otherwise, an empty group at the beginning indicates an abbreviation that only matches at the start of the text. The initial-text abbreviation is implemented by storing the abbreviation as a continuation of a zero-width word.

```
\def\abb@add@group#1#2#3#4{%
    \ifstrempty{#3}{%
        \abb@add@{#1}{#2}{#3}{#4}%
    }{%
        \abb@add@error{group}{{#3}#4}%
    }%
}
\def\abb@add@error#1#2#3{%
    \PackageError{abbrev}{%
        Abbreviation text `#2#3' should not start with #1 `#2'
    }{%
        Review the text to be abbreviated
    }%
}
\def\abb@add@punct#1#2#3#4{%
    \@test \ifcat \relax\noexpand#3\fi{%
        \abb@add@error{control sequence}{#3}{#4}%
    }{%
        \abb@add@{#1}{#2}{#3}{#4}%
    }%
}
```

#1 is the class, #2 is the abbreviated form, #3 the first word, #4 the remainder.

```
\def\abb@add@#1#2#3#4{%
    \make@find@start{#3}% I think this is superfluous
    \expandafter\abb@add@@
        \csname abb@class@#1@#3\endcsname
        {#2}{#4}%
}
```

#1 is the macro containing the class and first word, #2 the abbreviated form, #3 the remainder of the abbreviation.

```
\def\abb@add@@#1#2#3{%
    \make@find@start{#3}%
    \ifx#1\relax
        \def#1{{#3}{#2}}%
    \else
        \expandafter\abb@add@insert#1\@stop{#1}{#3}{#2}{}%
    \fi
}
```

Decides where to insert the new abbreviation. The key is that, because abbreviation matches are searched in sequence, longer strings need to come before substrings. For example, an abbreviation list:

```
{Aaa Bbb}{A.B.}{Aaa Bbb Ccc}{A.B.C.}
```

would be problematic since to abbreviate “Aaa Bbb Ccc Y”, the first abbreviation pair would match and the second would never be considered. So the list properly needs to be ordered:

```
{Aaa Bbb Ccc}{A.B.C.}{Aaa Bbb}{A.B.}
```

#1 and #2 are the abbreviation pair being considered, #3 is the rest of the list, #4 is the list macro, #5 the unabbreviated text to match, #6 the abbreviation, and #7 any material to prepend to the list.

```
\def\abb@add@insert#1#2#3\@stop#4#5#6#7{%
    \find@start{#1}{#5}{%
```

```
        \def#4{#7{#5}{#6}{#1}{#2}#3}%
        \@gobble
    }{%
        \ifstrempty{#3}{%
            \def#4{#7{#1}{#2}{#5}{#6}}%
        }{%
            \abb@add@insert#3\@stop{#4}{#5}{#6}{#7{#1}{#2}}%
        }%
    }%
}
```

2 Algorithm for Abbreviation

`\abb@abbrev{`⟨*class*⟩`}{`⟨*text*⟩`}{`⟨*callback*⟩`}` is the entrypoint for abbreviating a text. ⟨*class*⟩ is the class, ⟨*text*⟩ is the text to abbreviate, and ⟨*callback*⟩ is a callback function for the result.

Text is abbreviated by reading each word in the text, using `\find@word` from `strings.dtx`.

1. If it is not a word or punctuation, do not process the item and continue.
2. Otherwise, retrieve the list of abbreviations starting with this word. If there are none, then continue.
3. Iterate through each of the abbreviations associated with the word, and see if the remainder of the text to be abbreviated starts with any of the abbreviation texts. If so, then insert the relevant abbreviation text, correcting for dots as appropriate.

To provide the special feature of initial-text abbreviations, we assume that the text starts off with a zero-width word, and try to abbreviate it. Since the initial-text abbreviations are stored under the command for zero-width words, this produces the desired result.

```
\def\abb@abbrev#1#2#3{%
    \abb@abbrev@wd{}{#2}{}{#1}{#3}%
}
```

Initial state for abbreviation. This is called to process the next abbreviation off of the list of words.

#1 is the text to abbreviate, #2 is the accumulated abbreviated text, #3 is the class, #4 is the callback. (#2-#4 are left on the stack.)

```
\def\abb@abbrev@acc#1{%
    \find@word{#1}\abb@abbrev@punct\abb@abbrev@grp\abb@abbrev@wd\abb@abbrev@end
}
```

The token being abbreviated is punctuation. If it is a macro, nothing is done. Otherwise, treat it like a word.

#1 is the punctuation, #2 the remaining text, #3 the accumulated text, #4 the class, #5 the callback. (#4-#5 are left on the stack.)

```
\def\abb@abbrev@punct#1#2#3{%
    \@test \ifcat \relax\noexpand#1\fi{%
        \abb@abbrev@acc{#2}{#3#1}%
    }{%
        \abb@abbrev@wd{#1}{#2}{#3}%
    }%
}
```

The token being abbreviated is a group. Nothing is done.

#1 is the group, #2 the remaining text, #3 the accumulated text, #4 the class, #5 the callback. (#4-#5 are left on the stack.)

```
\def\abb@abbrev@grp#1#2#3{%
    \abb@abbrev@acc{#2}{#3{#1}}%
}
```

There is no more text left to abbreviate. Invoke the callback with the accumulated text.

#1 is the accumulated text, #2 the class, #3 the callback.

```
\def\abb@abbrev@end#1#2#3{%
    #3{#1}%
}
```

The token to be abbreviated is a word or abbreviatable punctuation. If there is no defined abbreviation for the word, then continue to the next token. Otherwise, call `\abb@abbrev@wd@`, placing on the stack the list of abbreviations associated with the word.

#1 is the word, #2 the remaining text, #3 the accumulated text, #4 the class, #5 the callback. (#5 is left on the stack.)

```
\def\abb@abbrev@wd#1#2#3#4{%
    \@ifundefined{abb@class@#4@#1}{%
        \abb@abbrev@acc{#2}{#3#1}{#4}%
    }{%
        \expandafter\expandafter\expandafter\abb@abbrev@wd@
            \csname abb@class@#4@#1\endcsname \@stop
            {#1}{#2}{#3}{#4}%
    }%
}
```

#1 and #2 are the first elements of the abbreviation word list; #3 is the remainder of that list. #4-#8 are the args to `\abb@abbrev@wd`. (#7-#8 are left on the stack.)

```
\def\abb@abbrev@wd@#1#2#3\@stop#4#5#6{%
    \find@start{#1}{#5}{%
        \abb@abbrev@found{#6}{#2}%
    }{%
        \@iden
    }%
    {% This is left on the stack after whatever \find@start calls
        \ifstrempty{#3}{%
            \abb@abbrev@acc{#5}{#6#4}%
        }{%
            \abb@abbrev@wd@#3\@stop{#4}{#5}{#6}%
        }%
    }%
}
```

#1 is the accumulated text if we abbreviate, #2 is the abbreviation to add, #3 is the remaining text if we abbreviate, #4 is the command to run if we bail out, #5 is the class, and #6 is the callback. (#5-#6 are left on the stack.)

```
\def\abb@abbrev@found#1#2#3#4{%
    %
    \@ifstartswithwordchar{#3}{%
        #4%
    }{%
        \find@try\find@start{%
            { }{\abb@abbrev@maybechop{ }}%
            {.}{\abb@abbrev@maybechop{.}}%
        }{#3}{\abb@abbrev@acc}%
        {#3}{#1#2}%
    }%
}
```

#1 is the character to test for potential removal at the end of the accumulated text, #2 is the remaining text with the character removed, #3 is the remaining text without the character removed, and #4 is the accumulated text. #5 is the class, #6 the callback (#5-#6 are left on the stack).

```
\def\abb@abbrev@maybechop#1#2#3#4{%
    \find@end{#1}{#4}{%
        \@firstoftwo{\abb@abbrev@acc{#2}{#4}}%
    }{\abb@abbrev@acc{#3}{#4}}%
}%
\make@find@start{.}
\make@find@start{ }
\make@find@end{.}
\make@find@end{ }
```

3 Removal of Spaces Between Dotted Initials

`\abb@initialdots{`⟨*text*⟩`}{`⟨*dot*⟩`}{`⟨*callback*⟩`}` processes a text to remove dots between single-letter initials. Given a text, it will convert sequences such as “A.

A. Milne" to "A.A. Milne" and "F. 4th" to "F.4th". ⟨*text*⟩ is the text to process and ⟨*callback*⟩ is the callback to run on the resulting output.

⟨*dot*⟩ is the dot character to search for. According to some abbreviation schemes, it is desirable not to collapse spaces between single initials when the initials refer to different classes of words. The macro `\abb@dot` can be used in the place of the period character to indicate this differentiation between classes.

For example, the text "Northern University Law Journal" may be abbreviated to "N.U. L.J." with only the first and third spaces collapsed, but not the second because institution words should not be joined with publication words. This can be implemented by abbreviating publication words with `\abb@dot` (shown here for an abbreviation class `journal`):

```
\abb@new{journal}
\abb@add{journal}{Northern}{N.}
\abb@add{journal}{University}{U.}
\abb@add{journal}{Law}{L\abb@dot}
\abb@add{journal}{Journal}{J\abb@dot}
```

Now when the text is abbreviated with `\abb@abbrev`, it produces:

```
N.␣U.␣L\abb@dot␣J\abb@dot
```

The dot removal algorithm can now be run twice, once with a period as ⟨*dot*⟩ and once with `\abb@dot`, thereby producing the desired text.

This uses `\find@next` on the text with a state machine having states described below. Each state has macros for each of the possible find results from d.tionally, these items are always left on the token stack:

- the processed text
- the deferred text
- the dot character
- the final callback

Initial entrypoint. #1 is the text, #2 is the character to treat as a dot, and #3 is the callback for the result.

```
\def\abb@initialdots#1#2#3{%
    \abb@id@a{#1}%
    {}% Initially the processed text is empty.
    {}% Deferred text is empty.
    {#2}% The dot character.
    {#3}% At end of the macro stack for use by \abb@id@done
}
\def\abb@id@done#1#2#3#4{%
    #4{#1#2}%
}
```

a. Initial state.

- On letter, go to b.
- Otherwise go to a.

```
\def\abb@id@a#1{%
    \find@next{#1}\abb@id@backtoa\abb@id@backtoa\abb@id@a@letter\abb@id@done
}
\def\abb@id@backtoa#1#2#3#4{%
    \abb@id@a{#2}{#3#4#1}{}%
}
\let\abb@id@a@space\abb@id@backtoa
\let\abb@id@a@group\abb@id@backtoa
\def\abb@id@a@letter#1#2#3#4{%
    \@ifletter{#1}{%
```

```
        \abb@id@b{#2}{#3#4#1}{}%
    }{%
        \@ifdigit{#1}{%
            \abb@id@d{#2}{#3#4#1}{}%
        }{%
            \abb@id@a{#2}{#3#4#1}{}%
        }%
    }%
}
```

b. Seen a single letter.

- On dot, go to c.
- On letter, go to d.
- Otherwise go to a.

```
\def\abb@id@b#1{%
    \find@next{#1}\abb@id@b@space\abb@id@a@group\abb@id@b@letter\abb@id@done
}
\let\abb@id@b@space\abb@id@backtoa
\let\abb@id@b@group\abb@id@backtoa
\def\abb@id@b@letter#1#2#3#4#5{%
    \@ifletter{#1}{%
        \abb@id@d{#2}{#3#4#1}{}%
    }{%
        \@test\ifx#1#5\fi{%
            \abb@id@c{#2}{#3#4#1}{}%
        }{%
            \abb@id@a{#2}{#3#4#1}{}%
        }%
    }%
    {#5}% Put the dot character back on the stack
}
```

c. Seen a dot after b.

- On space, go to e.
- On letter, go to b.
- Otherwise, go to a.

```
\def\abb@id@c#1{%
    \find@next{#1}\abb@id@c@space\abb@id@c@group\abb@id@c@letter\abb@id@done
}
\def\abb@id@c@space#1#2#3#4{%
    \abb@id@e{#2}{#3#4}{#1}%
}
\let\abb@id@c@group\abb@id@backtoa
\def\abb@id@c@letter#1#2#3#4{%
    \@ifletter{#1}{%
        \abb@id@b{#2}{#3#4#1}{}%
    }{%
        \@ifdigit{#1}{%
            \abb@id@d{#2}{#3#4#1}{}%
        }{%
            \abb@id@a{#2}{#3#4#1}{}%
        }%
    }%
}
```

d. In a word

- On letter, go to d.
- Otherwise go to a.

```
\def\abb@id@d#1{%
    \find@next{#1}\abb@id@d@space\abb@id@d@group\abb@id@d@letter\abb@id@done
}
\let\abb@id@d@space\abb@id@backtoa
\let\abb@id@d@group\abb@id@backtoa
\def\abb@id@d@letter#1#2#3#4{%
    \@ifwordchar{#1}{%
        \abb@id@d{#2}{#3#4#1}{}%
    }{%
        \abb@id@a{#2}{#3#4#1}{}%
    }%
}
```

e. Seen a space after c

- On letter, go to f.
- On number, collapse the space and go to d.
- Otherwise, go to a.

```
\def\abb@id@e#1{%
    \find@next{#1}\abb@id@e@space\abb@id@e@group\abb@id@e@letter\abb@id@done
}
\def\abb@id@e@space#1#2#3#4{%
    \abb@id@e{#2}{#3}{#1}%
}
\let\abb@id@e@group\abb@id@backtoa
\def\abb@id@e@letter#1#2#3#4{%
    \@ifletter{#1}{%
        \abb@id@f{#2}{#3}{#4#1}%
    }{%
        \@ifdigit{#1}{%
            \abb@id@d{#2}{#3#1}{}%
        }{%
            \abb@id@a{#2}{#3#4#1}{}%
        }%
    }%
}
```

f. Seen a single letter after e

- On dot, collapse the space and go to c.
- On letter, go to d.
- Otherwise, go to a.

```
\def\abb@id@f#1{%
    \find@next{#1}\abb@id@f@space\abb@id@f@group\abb@id@f@letter\abb@id@done
}
\let\abb@id@f@space\abb@id@backtoa
\let\abb@id@f@group\abb@id@backtoa
\def\abb@id@f@letter#1#2#3#4#5{%
    \@ifwordchar{#1}{%
        \abb@id@d{#2}{#3#4#1}{}%
    }{%
        \@test\ifx#1#5\fi{%
            \abb@id@f@stripspace{#1}{#2}{#3}#4%
        }{%
            \abb@id@a{#2}{#3#4#1}{}%
        }%
    }%
    {#5}%
}
\def\abb@id@f@stripspace#1#2#3#4{%
    \abb@id@c{#2}{#3#4#1}{}%
}
```

A token for dots.

```
\def\abb@dot{.}
```

1 Options

```
\ProcessOptions\relax
```