The abbrevs LaTeX package
abbreviation macros (Frankenstein's briefs)

Matt Swift <swift@alum.mit.edu>

Version: 1.4 Date: 2001/09/08
Documentation revision: 2001/09/08

Abstract

"Abbreviation macros" expand to defined text and insert following space intelligently, based on context. They can also expand to one thing the first time they are used and another thing on subsequent invocations. Thus they can be abbreviations in two senses, in the source and in the document. Useful applications include the abstraction of textual elements such as names without fussing over spacing and the automatic expansion of abbreviations and acronyms at their first use. The initial and subsequent expansions of an abbreviation macro are available at any time via explicit commands. Abbreviation macros are grouped into categories; there are hooks applicable to each category. Categories can be reset so that subsequent abbreviation macros in that category behave as if used for the first time again.

A generic facility is also provided for suffixes like 1900 B.C. and 6:00 P.M., which correctly handles following periods.

Contents

Part I
Discussion

1 General

`\nospacelist` An abbreviation macro `\foo` that expands to ⟨*text*⟩ is robust; `\foo` can be used in place of ⟨*text*⟩ almost anywhere. A space is inserted following an abbreviation macro when the first non-white character following it is *not* in the set `\nospacelist`, whose default value is `,.':;?-/~!)]{}\␣\/\@xobeysp`.

When an abbreviation macro has different initial and subsequent expansions, either may be explicitly requested by adding a suffix to the abbreviation macro. The commands `\`⟨*command*⟩`short` and `\`⟨*command*⟩`long` are also defined whenever an abbreviation macro `\`⟨*command*⟩ is defined. Using the `\`⟨*command*⟩`long` command does not affect what the next abbreviation macros expands to.

All abbreviation macros are assigned categories, identified by a string. Four categories are defined by the package, and it is easy to add more. Categories facilitate handling different groups of abbreviation macros in different ways.

Warning: *Regarding CJK macros and probably other 8-bit input. If you use the abbrevs package with the CJK macros for typesetting Chinese, Japanese, and Korean text, you must define your abbreviations within the CJK environment. I believe that the CJK macros work by interpreting 8-bit input in the source file. But this input is only interpreted properly within the CJK environment. If you define the abbrevs outside, such as in the preamble, you will just get a bunch of numbers when your abbreviation expands.*

I would use capital letters for the name of this macro, since it doesn't seem like a user command to me, but I'm modelling after the kernel's `\nocorrlist`.

2 Usage

Examples of how to define abbreviation macros:

```
\newbook\worst{Worstward Ho}
\newbook\fall{All That Fall}
\newbook\nacht{Nacht und Tr\"aume}
\newbook\csp{Collected Shorter Plays \emph{(}CSP\emph{)}}[CSP]
\newname\joyce{James Joyce}[Joyce]
\newname\nixon{Richard Milhous Nixon}[Nixon]
\newname\ww{Wordsworth}
\newname\beckett{Samuel Beckett}[Beckett]
\newwork\godot{Waiting for Godot}[Godot]
\newbook\prelude{The Prelude}
\newabbrev\ART{American Repetrory Theater (ART)}[ART]
```

To do: *Give example of using* `short` *or* `long`*.*

Examples of how to use the macros, and how they are typeset:[1]

```
The manuscripts of \ww's \prelude differ. \lips Before he began \prelude,
\ww wrote \lips
```

[1] `\lips` is defined in the *lips* package, part of the `Frankenstein` bundle.

LOOKS LIKE:

The manuscripts of Wordsworth's *The Prelude* differ. . . . Before he began *The Prelude*, Wordsworth wrote . . .

```
\nixon was the 37st American President. \lips Many Americans like my uncle
Norm voted for \nixon enthusiastically in both 1968 and 1972.
```

LOOKS LIKE:

Richard Milhous Nixon was the 37st American President. . . . Many Americans like my uncle Norm voted for Nixon enthusiastically in both 1968 and 1972.

```
\beckett gained international noteriety with the play \godot in the early
1950s.  \beckett wrote \godot, he said, as a diversion from the novels he
was then writing.  I have seen this play at the \ART in Cambridge,
Massachusetts.  The \ART is often disappointing, but I liked their
production of \godot.
```

LOOKS LIKE:

Samuel Beckett gained international noteriety with the play *Waiting for Godot* in the early 1950s. Beckett wrote *Godot*, he said, as a diversion from the novels he was then writing. I have seen this play at the American Repetrory Theater (ART) in Cambridge, Massachusetts. The ART is often disappointing, but I liked their production of *Godot*.

\newabbrev `\newabbrev {⟨\*command*⟩}{⟨*initial*⟩}[⟨*subsequent*⟩]` defines an abbreviation macro ⟨*command*⟩ of category Generic.

\newname `\newname {⟨\*command*⟩}{⟨*initial*⟩}[⟨*subsequent*⟩]` defines an abbreviation macro ⟨*command*⟩ of category Name.

\newbook `\newbook {⟨\*command*⟩}{⟨*initial*⟩}[⟨*subsequent*⟩]` defines an abbreviation macro ⟨*command*⟩ of category Book.

\newwork `\newwork {⟨\*command*⟩}{⟨*bibliography key*⟩}{⟨*initial*⟩}[⟨*subsequent*⟩]` defines an abbreviation macro ⟨*command*⟩ of category Work. Works can be distinguished from books by being listed in a separate bibliography, e.g., of primary works referred to by short titles in the main text. The defining command therefore requires a BibTeX key as an argument. The first use of the work serves as a citation to that bibliography, and all uses of the work generate an index entry.

To do: *Works are not yet fully implemented. Presently they are the same as Books.*

3 Date Marks

\PM \AM \BC \AD These variants of abbreviation macros correctly handle following periods.

```
She left for work before 6\AM, but
did not arrive until 12\PM.  The
interval 5\BC--5\AD is one year
shorter than the interval
95\AD--105\AD.
```

LOOKS LIKE:

She left for work before 6 A.M., but did not arrive until 12 P.M. The interval 5 B.C.–5 A.D. is one year shorter than the interval 95 A.D.–105 A.D.

4 Emulation of *acromake*

We emulate the *acromake* package by Paul A. Thompson (version of 1995/7/16 at `CTAN:/macros/latex/contrib/other/misc/acromake.sty`). *Abbrevs* will issue an informative warning when it guesses it is about to fail because *acromake* is already loaded (we cannot know for certain if it is). I will add an option so that *abbrevs* and *acromake* can both be loaded if anyone persuades me it will be useful.

One reason to emulate *acromake* with *abbrevs* is that it can be done easily and by doing so we can avoid keeping two packages around when one will do. Another is that *abbrevs* is a more general and powerful package which adds value to *acromake* functionality. *Abbrevs* should be a drop-in replacement for *acromake*, but you can also take advantage of features of *abbrevs*: acromake-style abbreviations obey `\TMInhibitSwitching`, and they are defined as their own category of abbreviations, Acromake, so that *acromake*-style abbreviations can be manipulated with the general mechanisms available to any category.

The following three *acromake* user commands are implemented in *abbrevs*.

`\acromake` `\acromake {`⟨*csname*⟩`}{`⟨*initial text*⟩`}{`⟨*final text*⟩`}` FIX dox

`\ACRcnta` The macro `\ACRcnta` contains the number of times (default 1) the initial text (full text) is given. Use `\renewcommand` to redefine it.

`\ACRcntb` The macro `\ACRcntb` contains the number of iterations (default 2) before the final text is given. The intermediate text (final text plus page reference) is therefore given `ACRcntb` − `ACRcnta` times.

`\AcromakePageref` Define the macro `\AcromakePageref` to contain the text that expresses the page reference. *Abbrevs* will replace the string `##1` in the definition of this macro with the page number where the abbreviation was first used (more precisely, with `\pageref{`⟨*label*⟩`}`). The default value is `(see Page ##1)` for compatibility with *acromake*. The styles I am familiar with would call for a lowercase "page."

Warning: *`\AcromakePageref` not implemented yet*

4.1 Possible discrepancies

The counter `util` and the macros `\pv` and `\addtomacro` are used internally by *acromake* and are not defined in *abbrevs*. (If you managed to find some use for `\addtomacro`, you will probably see easily how to redefine it in this context—and if not, write me.)

The emulation may behave slightly differently due to the difference beteween the way the *xspace* package handles following punctuation and space and the way *abbrevs* does. I think *abbrevs* is very likely to be as good or better than *xspace* at making these decisions. Let me know if you think otherwise.

5 Programmers' interface

`\ResetAbbrevs` When abbreviation macros are reset, their next invocation will expand to the initial text. Subsequent occurrences will expand to the subsequent text again. For

exmaple, using `\ResetAbbrevs {Name}` at the beginning of chapters will cause the full name to be used only for the first occurrence in each chapter. `\ResetAbbrevs {`⟨*category list*⟩`}` resets all abbreviation macros of the listed categories. The list is comma-separated, and the category All is a shorthand for all defined categories. Example:

```
\SaveCS\chapter
\renewcommand\chapter {%
  \ResetAbbrevs{All}%
  \MDSavedchapter
}
```

`\NewAbbrevCategory`
`\TMFontAll`
`\TMHookAll`
`\TMResetAll`
`\TMFont<category>`
`\TMHook<category>`
`\TMReset<category>`
`\NewUserAbbrevDefiner`

To create new categories of abbreviation, use `\NewAbbrevCategory {`⟨*category name*⟩`}`. Macros `\TMFont` ⟨*category*⟩, `\TMHook` ⟨*category*⟩, and `\TMReset`⟨*category*⟩ are all reserved. The hook and font slots start empty. The virtual category All is predefined and refers to all defined categories. `\TMHookAll` and `\TMFontAll` are called *before* the respective category-specific commands.

`\NewUserAbbrevDefiner {`⟨*defining command*⟩`}{`⟨*category*⟩`}[`⟨*definer*⟩`]` defines a user command ⟨*defining command*⟩. With the default ⟨*definer*⟩, `\TMDefineAbbrevStandard`, the ⟨*defining command*⟩ will take the arguments `{`⟨*abbrev command*⟩`}{`⟨*initial text*⟩`}[`⟨*subsequent text*⟩`]` and defines ⟨*abbrev command*⟩ to be a plain or switching abbreviation macro as appropriate. If given, the optional argument ⟨*definer*⟩ should be a macro name, which will be first be passed a `{`⟨*category*⟩`}`, then will read user arguments (e.g., in the case of `\TMDefineAbbrevStandard`, `{`⟨*cs*⟩`}{`⟨*initial*⟩`}[`⟨*subsequent*⟩`]`). The ⟨*definer*⟩ is expected of course to do something like define `{`⟨*cs*⟩`}`.

`\TMInitialSuffix`
`\TMSubsequentSuffix`

The factory default suffixes "short" and "long" may be changed by changing the definitions of `\TMSubsequentSuffix` and `\TMInitialSuffix`. The change should be made after the package is loaded but before any abbreviation macros have been defined.

`\DateMark`

Abbreviation macros like `\PM` are defined as `\DateMark`s, like this, without the final period:

```
\newcommand\PM {%
  \DateMark{p.m}%
}
```

`\ifTMInhibitSwitching`
`\TMInhibitSwitchingfalse`
`\TMInhibitSwitchingtrue`
`\ifTMAlwaysLong`
`\TMAlwaysLongtrue`
`\TMAlwaysLongfalse`

When `\ifTMInhibitSwitching` is true, first occurrences of an abbreviation macro will expand to the initial expansion as usual, but they will not trigger the change to subsequent expansions. Example: inhibit switching inside footnotes, and abbreviations will not be spelled out for the first and only time in a footnote. That is, if their first appearance is in a footnote, their first appearance in the main text will also expand to the long version. See the configuration file for how to do this.

When `\TMAlwaysLong` is true, every abbreviation macro expands to its initial expansion.

Part II
Implementation

6 Version control

\fileinfo
\DoXUsepackagE
\HaveECitationS
\fileversion
\filedate
\docdate
\PPOptArg

These definitions must be the first ones in the file.

```
\def\fileinfo{abbreviation macros (Frankenstein's briefs)}
\def\DoXPackageS {abbrevs}
\def\fileversion{v1.4}
\def\filedate{2001/09/08}
\def\docdate{2001/09/08}
\edef\PPOptArg {%
  \filedate\space \fileversion\space \fileinfo
}
```

If we're loading this file from a `\ProcessDTXFile` command (see the *compsci* package), then `\JusTLoaDInformatioN` will be defined; othewise we assume it is not (that's why the FunkY NamE).

If we're loading from `\ProcessDTXFile`, we want to load the packages listed in `\DoXPackageS` (needed to typeset the documentation for this file) and then bail out. Otherwise, we're using this file in a normal way as a package, so do nothing. `\DoXPackageS`, if there are any, are declared in the `dtx` file, and, if you're reading the typeset documentation of this package, would appear just above. (It's OK to call `\usepackage` with an empty argument or `\relax`, by the way.)

```
\makeatletter% A special comment to help create bst files.  Don't change!
\@ifundefined{JusTLoaDInformatioN} {%
  }{% ELSE (we know the compsci package is already loaded, too)
  \UndefineCS\JusTLoaDInformatioN
  \SaveDoXVarS
  \eExpand\csname DoXPackageS\endcsname\In {%use \csname in case it's undefined
    \usepackage{#1}%
  }%
  \RestoreDoXVarS
  \makeatother
  \endinput
}% A special comment to help create bst files.  Don't change!
```

Now we check for LaTeX2e and declare the LaTeX package.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{abbrevs}[\PPOptArg]
```

7 Requirements

```
\NeedsTeXFormat{LaTeX2e}[1995/12/01]
\RequirePackage{moredefs,slemph}
```

Warning: *These docs could be much improved. There are far too many things called "definers." Cleaning up the basic code concepts wouldn't hurt either.*

8 Basics

Let's begin with the tricky part of inserting space based on context. The strategy is: first, if the following character is not in `\nocorr` and the current font is not slanted, insert an italic correction with `\sw@slant`; second, if the following character is not in `\nospacelist`, insert a space.

Again, in pseudocode:

```
LET T = the next token
IF (slanted font is current AND T NOT IN \nocorrlist)
  \sw@slant
FI
IF T NOT IN \nospacelist
  \space
FI
```

`\nospacelist` Put these in the order of their frequency. Anything in `\nocorrlist` should also be in here, most likely. I'm putting in `\@xobeysp` because it's in the *xspace* package, but I can't tell you when it would come up.

```
\requirecommand\nospacelist {%
  ,.':;?-/\slash~!)]\bgroup\egroup\@sptoken\ \space\/\@xobeysp
}
```

`\maybe@ic@space` `\maybe@ic@space@` `\maybe@ic@space` checks the next character and inserts an italic correction and space as appropriate.

```
\newcommand\maybe@ic@space {%
  \futurelet\@let@token\maybe@ic@space@
}
```

We first call the kernel's `\maybe@ic@`, then our own `\maybe@space@`.

```
\newcommand\maybe@ic@space@ {%
  \maybe@ic@
  \maybe@space@
}
```

`\maybe@space` `\maybe@space@` `\maybe@space` and `\maybe@space@` are very similar to the kernel's analogs `\maybe@ic` and `\maybe@ic@`, but they check `\nospacelist` instead of `\nocorr`. `\t@st@ic` sets `\@tempswa` false if `\@let@token` is in `\nospacelist`.

```
\newcommand\maybe@space {%
  \futurelet\@let@token\maybe@space@
}
\newcommand\maybe@space@ {%
  \@tempswatrue
%  \DTypeout{In maybe@space@ my lettoken is [\meaning\@let@token]}%
  \expandafter   \@tfor
    \expandafter \reserved@a
    \expandafter :%
    \expandafter =%
                 \nospacelist
                 \do \t@st@ic
  \if@tempswa
    \space
  \fi
}
```

9 Categories

\ResetAbbrevs
\NewAbbrevCategory
\TMResetAll
\TMHookAll
\TMFontAll

Each time an abbreviation of category C is defined, some tokens are added to the contents of \TMReset⟨C⟩.

```
\ReserveCS\TMResetAll
\ReserveCS\TMHookAll
\ReserveCS\TMFontAll

\newcommand\NewAbbrevCategory [1] {% args:  category
  \expandafter\ReserveCS\csname TMReset#1\endcsname
  \expandafter\ReserveCS\csname TMFont#1\endcsname
  \expandafter\ReserveCS\csname TMHook#1\endcsname
  \expandafter\g@addto@macro
    \expandafter\TMResetAll\csname TMReset#1\endcsname
}
\newcommand\ResetAbbrevs [1] {% args: category-list
  \@for\sc@t@a:=#1\do {%
    \@ifundefined{TMReset\sc@t@a} {%
        \FrankenWarning{abbrevs}{The abbreviation category \sc@t@a\space
                                  is not defined!}%
      }{% ELSE
      \@nameuse{TMReset\sc@t@a}%
    }%
  }%
}
```

10 Suffixes

\TMInitialSuffix
\TMSubsequentSuffix

When an abbreviation macro is created, two additional commands with these suffixes are also created. For example, \foo, \foolong, and \fooshort. When abbrevs are used in such a way that "long" and "short" don't make sense, it would make sense to change these to something more descriptive.

```
\newcommand\TMInitialSuffix {%
  long%
}
\newcommand{\TMSubsequentSuffix} {%
  short%
}
```

11 Plain abbreviations

The checking that \sw@slant does for skips and penalties on the list is going to be superfluous for the applications I imagine. But we trade that for a more flexible macro.

We don't check for \nocorr or an empty body; maybe we should when it's first defined; but I ran into really hairy expansion troubles trying to do that and use \DeclareRobustCommand. FIX.

\TMNewAbbrevPlain

Things are easy when the abbreviation doesn't switch between initial and subsequent expansions.

To do: *pass root and suffix instead of* `\csname` *so that we don't have to parse it out again later from tmcurrentmacro*

```
\ReserveCS\TMCurrentMacro
\newcommand\TMNewAbbrevPlain [3] {% args: \csname category body
  \NewRobustCommand #1 {%
%    \xdef\TMCurrentMacro {\expandafter\Gobble\string#1}%
    \@bsphack
    \TMHookAll
    \@nameuse{TMHook#2}%
    \@esphack
    \ifmmode
      \def\sc@t@a {%
        \nfss@text{\@nameuse{TMFont#2}#3}%
      }%
    \else
      \def\sc@t@a {%
        \leavevmode
        \begingroup
```

We can skip the check for emptiness and containing just a space, since those won't occur with abbreviation macros except by accident, I think. We proceed straight to a check for `\nocorr`s.

```
          \tm@check@nocorr #3\nocorr\@nil
          \TMFontAll
          \@nameuse{TMFont#2}%
          \tm@check@left
          #3%
          \tm@check@right
        \endgroup
      }%
    \fi
    \sc@t@a
  }%
}
```

`\tm@check@nocorr` This corresponds to the kernel's `\check@nocorr@`. We simply substitute `\maybe@ic@space` and `\maybe@space` in where necessary. We also use `\tm@check@left` and `\tm@check@right` instead of `\check@icl` and `\check@icr`.

```
\NewName{tm@check@nocorr} {#1#2\nocorr#3\@nil} {%
  \let\tm@check@left\maybe@ic
  \def\tm@check@right {\aftergroup\maybe@ic@space}%
  \def\reserved@a {\nocorr}%
  \def\reserved@b {#1}%
  \def\reserved@c {#3}%
  \ifx\reserved@a\reserved@b
    \ifx\reserved@c\@empty
      \let\check@icl\@empty
    \else
      \let\check@icl\@empty
      \def\check@icr {\aftergroup\maybe@space}%
    \fi
  \else
    \ifx\reserved@c\@empty\else
```

```
      \def\tm@check@right {\aftergroup\maybe@space}%
    \fi
  \fi
}
```

12 Control booleans

\ifTMInhibitSwitching
\TMInhibitSwitchingtrue
\TMInhibitSwitchingfalse
\ifTMAlwaysLong
\TMAlwaysLongtrue
\TMAlwaysLongfalse

Control booleans.

```
\newboolean{TMInhibitSwitching} % initially false
\newboolean{TMAlwaysLong}       % initially false
```

13 Switching abbreviations

\TMNewAbbrevSwitcher

Here is the main abbreviation macro definer. It works by defining two macros, one for the initial text and one for the subsequent text, and setting up a third user command to choose between the two as appropriate. (The first two are made available to the user by explicit call as well.) The function used to define the two macros is passed as the first argument to this function. Supplied definers are `\TMNewAbbrevPlain` (I will write `\TMNewAbbrevWork` and `\TMNewAbbrevDotclose` soon FIX). The second argument is the category—each definer takes at least three arguments: a command name, a category, and the content. The third argument is the user macro name to be created, and the fourth and fifth arguments are the initial and subsequent expansion texts.

The first part sets three token variables to the three command sequences that this macro is going to define—the user, initial, and subsequent commands. The user command checks its associated boolean variable to see whether it has been called before. If so, it calls the "subsequent" macro; if not, the "initial" macro.

```
\newcommand\TMNewAbbrevSwitcher [5] {% args: definer category csname
%                                    %       initial subseq.
  \expandafter#1\csname #3\TMInitialSuffix\endcsname{#2}{#4}
  \expandafter#1\csname #3\TMSubsequentSuffix\endcsname{#2}{#5}
  \newboolean{@#3@mentioned}
  \expandafter\g@addto@macro\csname TMReset#2\endcsname {%
    \global\csname @#3@mentionedfalse\endcsname
  }
```

We've created the initial and subsequent macros, and the boolean. Now we define the user macro. This definition is tricky. In pseudocode, it looks like this:

```
if #3 definable then
  #3 := { if (#3-mentioned AND NOT TMAlwaysLong) then
            #3-short
          else
            if NOT TMInhibitSwitching then #3-mentioned := (global) true
            #3-long
          fi }
fi
```

Hmm, I'm not sure this is any more readable than a sea of `\expandafter \noexpand`s.

Notice that in a switching abbrev, the -mentioned boolean is set to true *before* calling the macro itself, so that the hook can check and possibly alter the value. The *acromake* emulation takes advantage of this.

```
  \expandafter\@ifdefinable\csname #3\endcsname {%
%                 is ##1 below:
    \EExpand\csname #3\endcsname\In {%
%                   ####1:
    \EExpand\csname if@#3@mentioned\endcsname\In {%
%                   ########1:
    \EExpand\csname #3\TMSubsequentSuffix\endcsname\In {%
%                   ################1:
    \EExpand\csname @#3@mentionedtrue\endcsname\In {%
%                   ################################1:
    \EExpand\csname #3\TMInitialSuffix\endcsname\In {%
%     \gdef\<csname>{%
      \gdef ##1{%  must be NO SPACE before `{' !
        \@tempswafalse
%       \if@<csname>mentioned
        ####1%
          \ifTMAlwaysLong\else
            \@tempswatrue
          \fi
        \fi
        \if@tempswa
%         \def\sc@t@a {\<csname>\TMSubsequentSuffix}%
          \def\sc@t@a {########1}%
        \else
          \ifTMInhibitSwitching\else
%           \global\@<csname>@mentionedtrue
            \global ################1%
          \fi
%         \def\sc@t@a {\<csname>\TMInitialSuffix}%
          \def\sc@t@a {################################1}%
        \fi
%        \expandafter  \gdef
%          \expandafter\TMCurrentMacro
%          \expandafter{\sc@t@a}%
        \sc@t@a
      }%   close \gdef
    }}}}}% close \EExpand...\In's
  }%       close \@ifdefinable
}
```

Warning: *The* `\csname`*s (e.g., either* `\foolong` *or* `\fooshort`*)* must *be the very last thing to occur in the definitions, or the* `\futurelet` *that checks following spacing in, e.g.,* `\TMNewAbbrevPlain` *will break. This is why we use the construction with* `\sc@t@a`*. No space must sneak into the macros, either!*

The hard work is done. Now we define some macros to help create new categories.

14 Defining commands

A ⟨*definer*⟩ is always called with a category as a first argument. The only definers in this version of this package are this one and the one that emulates the *acromake* package. More later!

\TMDefineAbbrevStandard
\tm@defineabbrevstandard

`\TMDefineAbbrevStandard` is the standard ⟨*definer*⟩ that makes the choice between defining an switching or a plain abbreviation, depending on whether the user supplies a subsequent text.

```
\newcommand\TMDefineAbbrevStandard [3] {% args: category \csname
                                        %       initial [subsequent]
  \@ifnextchar [ {%
      \tm@defineabbrevstandard{#1}{#2}{#3}%
    }{% ELSE
    \TMNewAbbrevPlain{#2}{#1}{#3}%
  }%
}
\NewName{tm@defineabbrevstandard} {#1#2#3[#4]} {% args: category \csname
                                                 %       initial subsequent
  \eExpand\expandafter\Gobble\string#2\In {%
    \TMNewAbbrevSwitcher\TMNewAbbrevPlain{#1}{##1}{#3}{#4}%
  }%
}
```

\NewUserAbbrevDefiner
\tm@newuserabbrevdefiner

```
\newcommand\NewUserAbbrevDefiner [2] {% args: \csname category [definer]
  \@ifnextchar [ {%
      \tm@newuserabbrevdefiner{#1}{#2}%
    }{% ELSE
    \tm@newuserabbrevdefiner{#1}{#2}[\TMDefineAbbrevStandard]%
  }%
}
\NewName{tm@newuserabbrevdefiner}{#1#2[#3]} {% args: \csname category definer
  \newcommand #1 {%
    #3{#2}%
  }%
}
```

15 Basic categories

\TMResetGeneric
\TMResetName
\TMResetBook
\TMResetWork
\TMHookGeneric
\TMHookName
\TMHookBook
\TMHookWork
\TMFontGeneric
\TMFontName
\TMFontBook
\TMFontWork
\newabbrev
\newname
\newbook
\newwork

Right now, the Book and Work categories are separate but equal. A future revision will distinguish them by keeping track of more information about Works, with the idea of using them to generate a separate bibliography and index in a long document that refers to a certain list of books by short titles. E.g., my thesis is on Samuel Beckett, and I want to refer to his works by short titles, and automatically generate a Beckett bibliography of only the ones I use, listed by title.

```
\NewAbbrevCategory{Generic}
\NewAbbrevCategory{Name}
\NewAbbrevCategory{Book}
\NewAbbrevCategory{Work}
\NewUserAbbrevDefiner{\newabbrev}{Generic}
\NewUserAbbrevDefiner{\newname}{Name}
```

```
\NewUserAbbrevDefiner{\newbook}{Book}
\NewUserAbbrevDefiner{\newwork}{Work}

\def\TMFontBook {%
  \itswitch
}
\def\TMFontWork {%
  \itswitch
}
```

16 Date marks

\DateMark
\DateMarkSize

```
\newcommand\DateMark [1] {%
  \hspace{.2em}{\DateMarkSize\scshape #1}%
  \@ifnextchar. {%
      \spacefactor\@m
    }{% ELSE
    .\maybe@ic@space
  }%
}
\newlet\DateMarkSize\small
```

\PM
\AM
\BC
\AD

Some common time abbreviations.

```
\newcommand{\PM} {%
  \DateMark{p.m}%
}
\newcommand{\AM} {%
  \DateMark{a.m}%
}
\newcommand{\BC} {%
  \DateMark{b.c}%
}
\newcommand{\AD} {%
  \DateMark{a.d}%
}
```

17 Emulation of *Acromake*

Warning: *This code is a* mess! *Consider all but the user functions "internal." I will reimplement it all another time. Meanwhile it works, and the user interface is acromake's anyway, so it won't change.*

\TMResetAcromake
\TMFontAcromake
\TMHookAcromake
\TMHookAcromakeHook
\TMAcromakeSecondarySuffix
\acromake
\tm@acromake@pageref

Define the category Acromake and declare its user defining command to be \acromake. Create a suffix secondaryanalogous to \TMInitialSuffix and \TMSubsequentSuffix.

Instead of building more generality into *abbrevs*, I emulate *acromake* with a few hacks, since I don't see a *general* need for more than two expansions. Counting iterations, on the other hand, is something I would like to do for all abbrevs. Doing so is tantamount to replacing the present -mentioned booleans with "counter" macros.

The emulation is done in the following way. Let's call the three expansions of an *acromake* macro the ⟨*am-initial*⟩, ⟨*am-secondary*⟩, and ⟨*am-subsequent*⟩ expansions, in order. These must be mapped onto the *abbrevs* concepts of *inital* and *subsequent* expansions. The `\acromake` command as I define it here defines a switching abbrev whose initial text contains ⟨*am-initial*⟩ and subsequent text contains ⟨*am-subsequent*⟩. It also defines a plain abbrev with a suffix `\TMAcroSecondarySuffix` (analogous to `\TMInitialSuffix` and `\TMSubsequentSuffix`) that expands to ⟨*am-secondary*⟩. In a switching abbrev, the associated `-mentioned` boolean is set to `true` *before* calling the macro itself (and therefore its hook). The hook can therefore reset the boolean to `false`, and I do this in `\TMHookAcromake` until it is time to go from the ⟨*am-initial*⟩ to the ⟨*am-secondary*⟩ expansions. `\TMInhibitSwitching` affects the Acromake category like all others, and `\TMResetAcromake` behaves as expected.

```
\@ifpackageloaded{acromake}{%
    \FrankenWarning{abbrevs}{LaTeX is about to fail because \protect\acromake
is already defined.\MessageBreak Probably you have loaded acromake.sty, and if
so,\MessageBreak you should simply not load it, since abbrevs.sty
emulates\MessageBreak acromake.sty.}
  }{%ELSE
}
\NewAbbrevCategory{Acromake}
\NewUserAbbrevDefiner{\acromake}{Acromake}[\TMAcromakeDefiner]

\newcommand\TMAcromakeSecondarySuffix {secondary}

\ReserveCS\tm@acromake@pageref
```

We're going to use the main hook, so provide another free one.

```
\ReserveCS\TMHookAcromakeHook
```

I'm not sure why *acromake* does this check for odd values of `\ACRcnta`. I use logic below that I think does reasonable things with odd values.

I think *acromake* tried to inhibit using ⟨*am-secondary*⟩ when it appeared on the same page as the (first!) ⟨*am-initial*⟩ instance, but I also think there was a spurious 0 in the source that broke this feature. I've emulated the working feature.

```
% consider these acromake functions internal for now!
% differs from regular version in passing args to definer
\newcommand\TMNewAbbrevSwitcherAcromake [5] {% args: definer category csname
%                                       %       initial subseq.
  #1{#3}{\TMInitialSuffix}{#2}{#4}
  #1{#3}{\TMSubsequentSuffix}{#2}{#5}
  \newboolean{@#3@mentioned}
  \expandafter\g@addto@macro\csname TMReset#2\endcsname {%
    \global\csname @#3@mentionedfalse\endcsname
  }
  \expandafter\@ifdefinable\csname #3\endcsname {%
%                 is ##1 below:
    \EExpand\csname #3\endcsname\In {%
%                   ####1:
    \EExpand\csname if@#3@mentioned\endcsname\In {%
%                   ########1:
    \EExpand\csname #3\TMSubsequentSuffix\endcsname\In {%
%                   ################1:
```

```
    \EExpand\csname @#3@mentionedtrue\endcsname\In {%
%                   ################################1:
    \EExpand\csname #3\TMInitialSuffix\endcsname\In {%
%     \gdef\<csname>{%
      \gdef ##1{%  must be NO SPACE before `{' !
        \@tempswafalse
%       \if@<csname>mentioned
        ####1%
          \ifTMAlwaysLong\else
            \@tempswatrue
          \fi
        \fi
        \if@tempswa
%         \def\sc@t@a {\<csname>\TMSubsequentSuffix}%
          \def\sc@t@a {########1}%
        \else
          \ifTMInhibitSwitching\else
%           \global\@<csname>@mentionedtrue
            \global ################1%
          \fi
%         \def\sc@t@a {\<csname>\TMInitialSuffix}%
          \def\sc@t@a {################################1}%
        \fi
%        \expandafter  \gdef
%          \expandafter\TMCurrentMacro
%          \expandafter{\sc@t@a}%
        \sc@t@a
      }%   close \gdef
    }}}}}% close \EExpand...\In's
  }%       close \@ifdefinable
}
\newcommand\TMAcromakeDefiner [4] {% args: category csname acronym fulltext
  \ifnum \ACRcnta < 1\relax
    \def\ACRcnta {1}%
  \fi%
```

Acromake uses a suffix of `z` here, which IMHO is a bad idea, so I use something a user will not put in a source file: a prefix of `tm@acromake@`. See below for why we start at -1.

```
  \Global\NewName{tm@acromake@#2}{} {-1}% macro for counting occurrences
```

Define an abbrev that switches from ⟨*am-initial*⟩ to ⟨*am-subsequent*⟩. Hack the resetting macro to reset the count as well as the `-mentioned` boolean. Reset to 0 not -1 so that we only label once! Is this an argument for having the "mentioned" boolean switch at the transition from ⟨*am-secondary*⟩ to ⟨*am-subsequent*⟩? (so that resetting goes back to ⟨*am-secondary*⟩ instead of ⟨*am-initial*⟩.) I think so; or, better, define a *new* label each time we reset.

```
  \TMNewAbbrevSwitcherAcromake\TMNewAbbrevAcromake{#1}{#2}{#4 (#3)}{#3}%
  \expandafter\g@addto@macro\csname TMReset#1\endcsname {%
    \global\DefName{tm@acromake@#2}{} {-1}%
  }%
```

Define an additional abbrev that expands to ⟨*am-secondary*⟩. In the code below, Fred replaces the `#1` with something else with `#1` in it; Ethel replaces the new `#1` with the contents of `\sc@toks@a`. (The `#` is quoted with more `#`'s below.) Hmm,

this expansion business could probably be simplified by thinking it through from the beginning.

```
%   \def\tm@acromake@pageref {%
%     \sc@toks@a={\noexpand\pageref{TMacromake:#2}}%
%     \EExpand\AcromakePageref\In {%  ``Fred''
%       \EExpand\sc@toks@a\In {% ``Ethel''
%         ####1%
%       }%
%     }%
%   }%
%  \eExpand\tm@acromake@pageref\In {%
    \TMNewAbbrevAcromake{#2}{\TMAcromakeSecondarySuffix}
      {Acromake}
      {#3\ (see Page \pageref{TMacromake:#2})}%
%        {#3\ ##1}%
%  }%
}
```

Now define `\TMHookAcromake`. Arg, first have to define an alternative of `\TMNewAbbrevPlain` because of the odd problem described above. Same as `\TMNewAbbrevPlain` except takes first argument in two parts and defines , which will be used in the hook.

```
\ReserveCS\TMCurrentMacroRootname
% plain's args: csname category body
\newcommand\TMNewAbbrevAcromake [4] {% args: csname-root csname-suffix category body
  \expandafter\NewRobustCommand\csname #1#2\endcsname {%
    \gdef\TMCurrentMacroRootname {#1}%
    \@bsphack
    \TMHookAll
    \@nameuse{TMHook#3}%
    \@esphack
    \ifmmode
      \def\sc@t@a {%
        \nfss@text{\@nameuse{TMFont#3}#4}%
      }%
    \else
      \def\sc@t@a {%
        \leavevmode
        \begingroup
```

We can skip the check for emptiness and containing just a space, since those won't occur with abbreviation macros except by accident, I think. We proceed straight to a check for `\nocorr`s.

```
          \tm@check@nocorr #4\nocorr\@nil
          \TMFontAll
          \@nameuse{TMFont#3}%
          \tm@check@left
          #4%
          \tm@check@right
        \endgroup
      }%
    \fi
    \sc@t@a
  }%
```

```
}
%\def\tm@chop#1 {#1}%
%\def\tm@choplong#1long{#1}

\ReserveCS\tm@t % temp
\def\TMHookAcromake {%
```

We handle inhibition of switching as follows. If the count is -1, this is the first iteration, so make the `\label`, increment the count to 0, and proceed. If switching is not inhibited, increment the counter. Then proceed with choosing the right expansion based on the counter. A first iteration in the normal case will therefore increment the counter twice from -1 to 1. A first iteration in the case that switching is inhibited will advance the counter once to 0, where it will stay until switching is permitted.

I need to extract the root name from the three suffixed names – why can't I do that?!

```
\documentclass{minimal}
\begin{document}
\def\gobble#1{}
\def\one{j}
\edef\two{\expandafter\gobble\string\j}
%\edef\two{\two} % doesn't help
\edef\three{\two}
\typeout{one: [\meaning\one]}
\typeout{two: [\meaning\two]}
[\one] [\two]
\typeout{\ifx\one\two ifx same\else ifx different -- WHY?!\fi}
\typeout{\if\one\two if same\else if different\fi}
```

```
%  \edef\tm@t{\expandafter\strip@prefix\meaning\TMCurrentMacro}%
%  \edef\tm@t{\E@cdr\tm@t\@nil}%
%  \edef\tm@t{\expandafter\tm@choplong\TMCurrentMacro}%
  \edef\tm@t{\TMCurrentMacroRootname}%
  \ifnum\csname tm@acromake@\tm@t\endcsname = -1\relax
    \eExpand\tm@t\In{%
%      \typeout{initial: labeling ##1}%
      \label{TMacromake:##1}%
    }%
    \tm@incmacro{\tm@t}%
  \fi
  \ifTMInhibitSwitching\else
    \tm@incmacro{\tm@t}%
  \fi
%  \typeout{BEGIN: count is \csname tm@acromake@\tm@t\endcsname}%
```

When the count is $<$ `ACRcnta`, reset the `-mentioned` boolean so that the expansion will be the initial text i.e., ⟨*am-initial*⟩, again the next time.

```
% doesn't work:
%  \ifnum\csname tm@acromake@\tm@t\endcsname < \ACRcnta%
% doesn't work:
%  \ifnum\csname tm@acromake@\tm@t\endcsname < \ACRcnta %
% works:  I have no clue why...
  \ifnum\csname tm@acromake@\tm@t\endcsname < \ACRcnta\relax
```

```
%    \typeout{use before cnta}%
    \expandafter\global\csname @\tm@t@mentionedfalse\endcsname
  \else
```

When the count is = `ACRcnta`, use the initial text (i.e., ⟨*am-initial*⟩) one last time and switch to using ⟨*am secondary*⟩ next time. We allow the `-mentioned` boolean to become `true` by refraining from resetting it. We save the existing subsequent macro (which expands to ⟨*am-subsequent*⟩) and substitute the abbrev that expands to ⟨*am-secondary*⟩. The bounds check on `\ACRcnta` at the beginning guarantees that we execute this clause once.

```
    \ifnum\csname tm@acromake@\tm@t\endcsname = \ACRcnta\relax
      \ifTMInhibitSwitching\else
% \typeout{use at cnta A}%
\SaveName{\tm@t\TMSubsequentSuffix}%
% \typeout{use at cnta B}%
\global\EElet\csname \tm@t\TMSubsequentSuffix\endcsname
     \csname \tm@t\TMAcromakeSecondarySuffix\endcsname
% \typeout{use at cnta C}%
      \fi
    \fi
```

When the count is > `ACRcnta` and <= `ACRcntb`, the expansion is ⟨*am-secondary*⟩, we only check whether we are currently on the page of the original use of the ⟨*am-initial*⟩ text and in this case use ⟨*am-subsequent*⟩.

```
%    \@tempswafalse
%    \ifnum\csname tm@acromake@\tm@t\endcsname < \ACRcntb\relax
%      \ifnum\csname tm@acromake@\tm@t\endcsname > \ACRcnta\relax
%  \@tempswatrue
%      \fi
%    \else
%      \ifnum\csname tm@acromake@\tm@t\endcsname = \ACRcntb\relax
%        \@tempswatrue
%      \fi
%    \fi
%    % ie    cnta < count <= cntb
%    \if@tempswa
% %      \typeout{use between cnta and cntb}%
%       \eExpand\tm@t\In{%
%           \typeout{page [\thepage]  ref [\r@TMacromake:##1]}%
%         \expandafter\ifnum\expandafter\thepage\expandafter=\csname r@TMacromake:##1\endcsn
% %           \typeout{this instance should be subsequent instead of secondary}%
%           % insert a * to signal where:
% %           *%
%           % FIX how to do it??  I leave it broken for now, as it is broken in acromake
%     % itself,
%         \fi
%        }%
%    \fi
```

When the count is = `ACRcntb`, we want to restore the definition of the subsequent macro. This test is not in an `\else` clause to handle the case where `ACRcnta` = `ACRcntb`.

```
%    \ifnum\csname tm@acromake@\tm@t\endcsname < \ACRcntb\relax
%      \ifnum\csname tm@acromake@\tm@t\endcsname > \ACRcnta\relax
% \typeout{use between cnta and cntb}%
```

```
%      \fi
%    \fi
    \ifnum\csname tm@acromake@\tm@t\endcsname = \ACRcntb\relax
%      \typeout{use at cntb}%
       \RestoreName{\tm@t\TMSubsequentSuffix}%
    \fi
  \fi
  \TMHookAcromakeHook
}
```

`\tm@incmacro` `\c@tm@util` This is Paul's trick for using a macro like a counter. I reduce the command to its essential function in this context. It looks like Paul wanted a more general command. I think if you define such a command (or set of commands that emulate counters with macros) they do not belong here but in *moredefs* or their own package. I also

```
\newcounter{tm@util}
\newcommand{\tm@incmacro} [1] {% arg: acroabbrev
  \eExpand\csname tm@acromake@#1\endcsname\In {%
    \setcounter{tm@util}{##1}%
  }%
  \stepcounter{tm@util}%
  \expandafter\xdef\csname tm@acromake@#1\endcsname {\thetm@util}%
}
```

`\ACRcnta`

```
\newcommand\ACRcnta {1}
\newcommand\ACRcntb {2}
```

`\AcromakePageref` The string `#1` will make it into the macro, which will in another context be replaced with a `\pageref`.

```
\newcommand\AcromakePageref {(see Page ##1)}
% suggestion:
% \renewcommand\AcromakePageref {(see page ##1)}
```

Part III
Configuration

We've built up the groundwork and leave the definitions of useful things to the configuration file.

```
\InputIfFileExists{abbrevs.cfg}{}{}
```

The contents of the distributed configuration file are below.

```
\def\fileinfo{Abbrevs package configuration}
\def\fileversion{v1.2}
\def\filedate{2001/08/31}
\def\docdate{1997/10/18}
\ProvidesFile{abbrevs.cfg}
```

18 \DateMarkSize

\DateMarkSize

I like to use this definition instead of the one in the main file, but I didn't want to require *abbrevs* to depend on *relsize*.

```
\RequirePackage{relsize}
\def\DateMarkSize {%
  \relsize{-1}%
}
```

19 Backwards compatibility

\TMNewCategory
\TMDefineAbbrevPlain

This can be uncommented to deal with anything you might have written that referred to these variables before I changed their names.

```
% \newlet\TMNewCategory\NewAbbrevCategory
% \newlet\TMDefineAbbrevPlain\TMDefineAbbrevStandard
```

20 Suggestions

Here are ideas commented out that you might want to try.

You can learn a helpful general strategy about how to work with hooks in LaTeX from this example. If you put the inhibitor directly into `\PreFootnote`, you could never take it out without either losing whatever else had been put into `\PreFootnote`, or using some thorny procedure that stepped through the macro and removed just the inhibitor (you don't want to try that). If you add a "subhook" to `\PreFootnote`, you can turn the subhook on or off without even knowing what else in in `\PreFootnote`. You can't redefine `\TMInhibitSwitchingtrue`. A `\newcommand` would work as well as the `\newlet` here, a tad less efficient.

```
% \newlet\FootnoteTMHook\TMInhibitSwitchingtrue
% \addto@macro\PreFootnote {%
%   \FootnoteTMHook
% }
```

To undo the effect later, say `\let\FootnoteTMHook\relax` or `\global\let` . . . as appropriate.

Part IV
Testing

I'm presently writing a dissertation on Samuel Beckett. Although there is comparatively little biographical material available, it is well known that he spent several years under the wing of James Joyce, another of the great writers in English this century. Joyce and Beckett, it is curious, like other great writers, both had trouble with their vision, and both were exiles in some sense. One of my favorite pieces by Beckett is *Worstward Ho*, a short work written in the 1980's not long before his death: "Fail again. Fail better." *Worstward Ho* is lyric and exalting to me. A work I feel is underrated is the radio play *All That Fall* (all but his three long plays are collected in *Collected Shorter Plays (CSP)*). It's extremely funny, and very touchingly compassionate. Because it is a radio play, it loses less from performance to reading. I would recommend *All That Fall* to anyone. His later plays (and fiction) are famously enigmatic, but with a little practice, it is not hard to see the same lyric beauty and compassion. Take the brief television play *Nacht und Träume* (in *CSP* of course), which has no dialogue, only a few murmured bars of the Schubert song, also brief, and also called *Nacht und Träume*—it's one of the most hauntingly beautiful few minutes of music I've ever heard, and I particularly recommend Cheryl Studer's recording on Deutsche Grammophone. Every other recording I've heard plays too fast.

Joyce is short for James Joyce, not Joyce Smith.

Now some more rigious and boring testing. Each pair should be identical.

initial *hello*
initial *hello*
subsequent hello
subsequent hello
subsequent tie
subsequent tie
subsequent regular text
subsequent regular text
subsequent: colon
subsequent: colon
subsequent; semicolon
subsequent; semicolon
subsequent. Period.
subsequent. Period.
subsequent! Exclamation point.
subsequent! Exclamation point.
subsequent? Question mark.
subsequent? Question mark.
subsequent-hyphen.
subsequent-hyphen.
subsequent `texttt`
subsequent `texttt`
subsequent (leftparen)
subsequent (leftparen)

(*subsequent*) rightparen
(*subsequent*) rightparen
subsequent, comma
subsequent, comma.
subsequent tmacro
subsequent tmacro
subsequent's face
subsequent's face
subsequent “quote”
subsequent “quote”
subsequent [leftbracket]
subsequent [leftbracket]
[*subsequent*] rightbracket
[*subsequent*] rightbracket
*subsequent*open group
*subsequent*opengroup
subsequent close group
subsequent close group
subsequent {realbrace}
subsequent {realbrace}
subsequent 666 number
subsequent 666 number
subsequent $x = y^2$ math
subsequent $x = y^2$ math
subsequent $realdollar
subsequent $realdollar
subsequent #numbersign
subsequent #numbersign
subsequent/slash
subsequent/slash
subsequent italic correction
subsequent italic correction
subsequent explicit space
subsequent explicit space
subsequent `\space`
subsequent `\space`
subsequent `\@xobeysp`
subsequent `\@xobeysp`
The Central Intelligence Agency (`CIA`) is overthrowing Nigeria.
The `CIA` (see Page 22) is watching in your window right now.
The `CIA` (see Page 22) will stop that missile.
The `CIA`! The `CIA`! The `CIA`. The `CIA` guys.
Resetting Acromake abbrevs.
The Central Intelligence Agency (`CIA`)! The `CIA` (see Page 22)! The `CIA` (see Page 22). The `CIA` guys.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

U

X