The titles LaTeX package
title macros (Frankenstein's references)

Matt Swift <swift@alum.mit.edu>

Version: 1.2 Date: 2001/08/31
Documentation revision: 2001/08/31

Abstract

The *titles* package defines macros that typeset the titles of books, journals, etc. and handle following spacing and punctuation intelligently, based on context. These are useful for bibliographic databases, for example. Also defined is other markup like `\word`, `\defn`, `\phrase`, etc.

Contents

Part I
Discussion

1 Options

There are two package options, `british` and `american`, the default is `american`. They select the conventional way to use quotation marks: British style is use single quotes, and do not suck following period or comma inside; American style is to use double quotes and to suck following period or comma inside.

2 Words and phrases

`\word` Typeset a word or phrase referred to as a noun with `\word {`⟨*word*⟩`}`. The argument is not expected to contain punctuation.

```
\word{Elephant} is such a silly word.
```

LOOKS LIKE:

Elephant is such a silly word.

`\phrase` Typeset a phrase used as a noun rather than direct quotation with `\phrase {`⟨*phrase*⟩`}`. The argument might well have punctuation, including final punctuation, which should not be considered to be punctuation of the containing sentence.

```
The sentence \phrase{And stop calling me Shirley!} occurs
twenty-seven times.
```

LOOKS LIKE:

The sentence 'And stop calling me Shirley!' occurs twenty-seven times.

`\foreign` Typeset a foreign word or phrase with `\foreign {`⟨*foreign text*⟩`}`.

```
I couldn't think of the \foreign{mot juste} at the time.
```

LOOKS LIKE:

I couldn't think of the *mot juste* at the time.

`\foreignword` Typeset a foreign word or phrase referred to as a noun with `\foreignword {`⟨*foreign word*⟩`}`.

```
Only later did I realize that the right word was
\foreignword{bouffon}.
```

LOOKS LIKE:

Only later did I realize that the right word was '*bouffon*'.

Warning: *Notice that writing* `\foreign{\word{text}}` *or vice versa is not necessarily going to do the right thing. Suppose* `\foreign` *and* `\word` *were both set to* `\textitswitch` *(which are in fact the default settings below). Then*

`\foreign{\word{text}}` is going to cancel out and look just like the surrounding text. This is not the most intuitive fact, but it's not worth it to try to make `\foreign` and `\word` smart enough to see each other inside themselves.

`\term` `\term {`⟨*technical term*⟩`}` typesets a techincal term in a different font. You might want to use this where a techincal term is first used, or defined. One could enhance this macro and `\defn` to help build an automatic glossary

```
This sort of thing is called a \term{blibnil}.
```

LOOKS LIKE:

This sort of thing is called a *blibnil*.

`\defn` `\defn {`⟨*definition*⟩`}` typesets a definition, perhaps of a technical term. One could enhance this macro and `\defn` to help build an automatic glossary

```
We may describe a \term{blibnil} as \defn{a slibnil with
three arms}.
```

LOOKS LIKE:

We may describe a *blibnil* as *a slibnil with three arms*.

3 Titles

`\book` `\book {`⟨*book title*⟩`}` typesets a book title.

```
Some people find \book{Moby-Dick} dull, but I thought
it was exciting.
```

LOOKS LIKE:

Some people find *Moby-Dick* dull, but I thought it was exciting.

`\journal` `\journal {`⟨*journal title*⟩`}` typesets a journal title.

```
I liked it so much I started a scholarly journal called
\journal{The Melville Times} with the inheritance from
my grandmother.
```

LOOKS LIKE:

I liked it so much I started a scholarly journal called *The Melville Times* with the inheritance from my grandmother.

`\music` `\music {`⟨*music title*⟩`}` typesets a music title.

```
My journal didn't do very well; I moped around my office
and listened to Schubert's \music{Winterreise}.
```

LOOKS LIKE:

My journal didn't do very well; I moped around my office and listened to Schubert's *Winterreise*.

`\article` `\article {`⟨*article title*⟩`}` typesets a article title.

```
Then one day I received an article, \article{Pip and
the Milk of Human Kindness}, by express mail from Wales.
```

LOOKS LIKE:

Then one day I received an article, “Pip and the Milk of Human Kindness,” by express mail from Wales.

`\poemtitle` `\poemtitle {⟨poem title⟩}` typesets a poem title.

```
I then wrote my famous poem \poemtitle{Jump for Joy like
the Butterflies of Troy} in five minutes.
```

LOOKS LIKE:

I then wrote my famous poem “Jump for Joy like the Butterflies of Troy” in five minutes.

Sometimes longer poems are distinguished from shorter ones in type when they have been published separately as a book [FIX give reference]. This package defines a macro `\longpoem` in the configuration file in the following way:

```
\newlet\longpoem\textitswitch
```

`\play` `\play {⟨play title⟩}` typesets a play title.

```
To celebrate the popularity of the article, I took the
author to the theater to see the acclaimed play
\play{Grave in Waterloo}, starring Vincent Price.
```

LOOKS LIKE:

To celebrate the popularity of the article, I took the author to the theater to see the acclaimed play *Grave in Waterloo*, starring Vincent Price.

`\craft` `\craft {⟨craft title⟩}` typesets a title of a craft or ship.

```
With tears in my heart, I put the author on the \craft{HMS Shangrila}
bound for Wales.
```

LOOKS LIKE:

With tears in my heart, I put the author on the *HMS Shangrila* bound for Wales.

`\species` `\species {⟨[ genus ] species [ subspecies ]⟩}` typesets the Latin generic and/or specific names for an organism.

Lesson 1 Chicago Manual of Style *specifies italic type. Genus names should be capitalized, and may be abbreviated on subsequent appearances with the initial letter. Following designations should be in roman. E.g., “var.” for “variant” following species name and “sp.” for “species” following genus name, meaning “any species in the genus.”* §*7.102–4*

Higher groupings should be in capitalized roman. English derivatives of scientific names, e.g., amoeba, are lowercased. §*7.105–6*

To do: *abbrevs category for genus/species and/or datemark for suffixes*

Warning: *Right now there is a small discrepancy between the behavior of* `\textitswitch` *and* `\Wrapquotes` *regarding what happens when followed by a command sequence such as* `\footnote`*. I hope to make these things completely parallel one day, but for now, realize that after using a titling macro that uses* `\Wrapquotes`*, you must use* `{}` *before any following command sequence that you want to immediately follow the title with no intervening space. The only case I can think of is* `\footnote`*. If you forget the* `{}`*, you will have an extra space after the title and before the footnotemark. The following example illustrates this behavior and contrasts it with* `\textitswitch`*:*

```
\newabbrev\foo{Foo}

\book{Foo}\foo

\book{Foo} \foo

\book{Foo}\footnote{footie}

\book{Foo}{}\footnote{footie}

\poemtitle{Foo}\foo

\poemtitle{Foo} \foo

\poemtitle{Foo}\footnote{footie}

\poemtitle{Foo}{}\footnote{footie}

\poemtitle{Foo}.\footnote{footie}
```

LOOKS LIKE:

FooFoo
Foo Foo
Foo[a]
Foo[b]
"Foo" Foo
"Foo" Foo
"Foo" [c]
"Foo"[d]
"Foo."[e]

[a] footie
[b] footie
[c] footie
[d] footie
[e] footie

4 Programmer's interface

`\Wrapquotes`
`\WrapquotesNS`
`\WrapquotesIS`
`\WrapquotesNN`
`\WrapquotesIN`
`\WrapquotesSN`
`\WrapquotesDN`
`\WrapquotesSK`

`\Wrapquotes {`⟨*text*⟩`}` wraps ⟨*text*⟩ in quotes. Single quotes are used initially if

the `singlequotes` option is given to the package, and double quotes if no option or the `doublequotes` option is given to the package.

When quotation marks inserted by `\Wrapquotes` and friends are doubled up (this occurs sometimes when nesting them), a thinspace (`\,`) is inserted between the abutted quotes.

`\Wrapquotes` will be `\let` to one of the six macros `\Wrapquotes` ⟨*XY*⟩.

In the two-letter suffix ⟨*XY*⟩, first letter `N` means "normal" and `I` means "inverse." These are macros that switch between single and double quotes when they nest: an inverse wrapquotes wraps with single quotes when a normal wrapquotes would wrap with double quotes, and vice versa. First letter `S` for "single" and `D` for "double" are for macros that always wrap with single or double quotes. Spacing and punctuation following the closing quotes are handled intelligently by macros with second letter `S`, which means means suck a following period or comma into the closing quote, that is, if what follows is a comma or period, it is pulled inside the quotes (following American practice). Second letter `N` means "nosuck," that is, don't suck. Second letter `K` means "kill": the same as `N` but suppress the effect of any punctuation in the quoted argument on spacing that follows the closing quotes (i.e., execute `\@`, which sets the spacefactor to 1000). This is only useful in certain technical writing where punctuation in the quoted argument should not be considered puncutation of the containing sentence.

A space is inserted after the closing quotes unless what follows is in the set `;?:!-)]'\textquoteright{`, in which case no space is inserted.FIX: that would be `\nospacelist`

`\IfQuestionOrExclamation` `\IfQuestionOrExclamation {`⟨*text*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}` executes ⟨*true*⟩ clause iff ⟨*text*⟩ ends with a question mark or an exclamation point; executes ⟨*false*⟩ clause otherwise.

4.1 Limitations of Wrapquotes and friends

4.1.1 Nesting

Warning: *For proper nesting of* `\Wrapquotes` *and friends, user commands must be* `\let` *to* `\Wrapquotes` *or one of the six* `\Wrapquotes` ⟨*XY*⟩ *commands instead of using a* `\def`*-like defining command. It's OK to* `\let` *a user macro to something like* `\Wrapquotes` *which itself has been* `\let` *to one of the six* `\Wrapquotes` ⟨*XY*⟩ *macros.*

The user command which is `\let` *to one of the* `\Wrapquotes` *commands must furthermore appear in the source. That is, it must not appear as the result of an expansion. Among other things, this means that nesting won't work properly if you put* `\Wrapquotes` *into an abbrev (see the abbrevs package in the* `Frankenstein` *bundle).*

For applications where nesting will not occur, there should be nothing to worry about.

4.1.2 Italic corrections

Warning: *The question of when to insert an italic correction is not nearly as simple as it might seem. I cannot figure good rules which cover all cases, and I do not trust the behavior of the kernel's macros as a guide. So I can not tell you whether this package handles italic corrections properly. If you discover behavior which you think is wrong, please let me know with an example and an argument.*

4.1.3 A slight bug

Warning: *Right now there is a small bug in cases where closing quotes fall at the end of italic text, such as*

```
\normalfont
\book{My love of \poemtitle{Daffodils}}, by H.~Moneysworth.
```

LOOKS LIKE:

My love of "Daffodils," by H. Moneysworth.

These cases loose because the closing quotation marks and any sucked-in punctuation are going to be in roman, not italic, or italic, not roman. Only the more obsessive will notice this flaw. I'm sure I will come up with a way to handle this for a future version of this package.

Part II
Implementation

5 Version control

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

These definitions must be the first ones in the file.

```
\def\fileinfo{title macros (Frankenstein's references)}
\def\DoXPackageS {}
\def\initelyHavECitationS {}
\def\fileversion{v1.2}
\def\filedate{2001/08/31}
\def\docdate{2001/08/31}
\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{titles}[\PPOptArg]
```

6 Requirements

```
\RequirePackage{moredefs,slemph}
```

7 Options

\ti@domelater

```
\ReserveCS\ti@domelater
```

```
\DeclareOption{british} {%
  \def\ti@domelater {%
    \let\Wrapquotes\WrapquotesNN
    \@doublequotes@false
  }
}
\DeclareOption{american} {%
  \def\ti@domelater {%
    \let\Wrapquotes\WrapquotesNS
    \@doublequotes@true
  }
}
\ExecuteOptions{american}
\ProcessOptions
```

8 Wrapquotes

Here we go! This is not a picnic, so leave your jelly jar home.

8.1 Titles that are Questions or Exclamations

```
\IfQuestionOrExclamation
          \ti@checkfor@q
          \ti@checkfor@e
                \ti@prev
           \ti@prev@prev
            \@ti@sw@true
           \@ti@sw@false
              \if@ti@sw@
```

```
\newcommand\IfQuestionOrExclamation [1] {%
  \@tempswafalse
  \ti@checkfor@q #1?\@nil
  \ti@checkfor@e #1!\@nil
  \if@tempswa
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
```

The large majority of titles will not contain a question mark or exclamation point. The large majority of those that do will have a single mark or point at the end. We could (I think) use a simpler check that processed all titles by looping through to examine the end, but a slightly more complicated check will handle the majority of cases very quickly (and at a constant speed, rather than proportional to title length) and not greatly slow down processing the remaining two unusual cases. We divide our argument (with an extra question mark tacked onto the end) into what's before the first question mark and what's after it. Then we examine what's after it and interpret the results thus:

empty no question mark in title

question mark title ends with question mark (and there are no other question marks)

text ending with one question mark a question mark occurs in the title, but not at the end

text ending with two question marks title ends with a question mark (and there is a previous question mark)

We set switch `a` to true if the title ends with a question mark.

```
\newboolean{@ti@sw@}
\ReserveCS\ti@prev
\ReserveCS\ti@prev@prev
\NewName{ti@checkfor@q} {#1?#2\@nil} {%
  \def\sc@t@a{#2}%
  \def\sc@t@b{?}%
  \ifx\sc@t@a\ShortEmpty
  \else
    \ifx\sc@t@a\sc@t@b
      \@tempswatrue
    \else
```

We use a loop to whittle down `#2` until `\ti@prev` contains the last character and `\ti@prev@prev` contains the second-to-last. We know that `\ti@prev` is going to be a question mark. Iff `\ti@prev@prev` is a question mark, we are in the final case above.

```
      \let\ti@prev\sc@t@a
      \let\ti@prev@prev\sc@t@a
      \@ti@sw@true
      \@whilesw \if@ti@sw@ \fi {%
        \ifx\sc@t@a\ShortEmpty
          \@ti@sw@false
        \else
          \let\ti@prev@prev\ti@prev
          \let\ti@prev\sc@t@a
          \edef\sc@t@a{\E@cdr\sc@t@a\@nil}%
        \fi
      }%
      \edef\ti@prev@prev{\E@car\ti@prev@prev\@nil}%
      \ifx\ti@prev@prev\sc@t@b
        \@tempswatrue
      \fi
    \fi
  \fi
}
```

Exact same logic applies to exclamation points.

```
\NewName{ti@checkfor@e} {#1!#2\@nil} {%
  \def\sc@t@a{#2}%
  \def\sc@t@b{!}%
  \ifx\sc@t@a\ShortEmpty
  \else
    \ifx\sc@t@a\sc@t@b
      \@tempswatrue
    \else
      \let\ti@prev\sc@t@a
      \let\ti@prev@prev\sc@t@a
      \@ti@sw@true
      \@whilesw \if@ti@sw@ \fi {%
        \ifx\sc@t@a\ShortEmpty
          \@ti@sw@false
        \else
          \let\ti@prev@prev\ti@prev
```

```
          \let\ti@prev\sc@t@a
          \edef\sc@t@a{\E@cdr\sc@t@a\@nil}%
        \fi
      }%
      \edef\ti@prev@prev{\E@car\ti@prev@prev\@nil}%
      \ifx\ti@prev@prev\sc@t@b
        \@tempswatrue
      \fi
    \fi
  \fi
}
```

8.2 Highlevel macros

`\ti@wrapquotes@suck`
`\ti@wrapquotes@nosuck`

These two are the top-level internal macros, and they are pretty sane. One sucks in a following period or comma, the other does not. `\ti@wrapquotes@suck` does not suck, however, when the title ends in a question or exclamation point.

The group here is necessary to scope the `\@doublequotes@` boolean.

```
\newcommand*\ti@wrapquotes@suck [1] {%
%  \DTypeout{top of wrapquotes@suck}%
  \IfQuestionOrExclamation {#1} {%
      \ti@wrapquotes@nosuck{#1}%
    }{% ELSE
%    \DTypeout{top of wrapquotes@suck ELSE}%
    \begingroup
      \if@doublequotes@
%        \DTypeout{double true in suck}%
        \@doublequotes@false
        \def\sc@t@a {\ti@open@double #1\ti@close@double@suck}%
      \else
%        \DTypeout{double false in suck}%
        \@doublequotes@true
        \def\sc@t@a {\ti@open@single #1\ti@close@single@suck}%
      \fi
      \sc@t@a
    \endgroup
    }%
}
\newcommand*\ti@wrapquotes@nosuck [1] {%
  \begingroup
    \if@doublequotes@
%      \DTypeout{double true in nosuck}%
      \@doublequotes@false
      \def\sc@t@a {\ti@open@double #1\ti@close@double@nosuck}%
    \else
%      \DTypeout{double false in nosuck}%
      \@doublequotes@true
      \def\sc@t@a {\ti@open@single #1\ti@close@single@nosuck}%
    \fi
    \sc@t@a
  \endgroup
}
```

`\WrapquotesNS`
`\WrapquotesIS`
`\WrapquotesNN`
`\WrapquotesIN`
`\WrapquotesSN`
`\WrapquotesDN`
`\WrapquotesSK`
`\Wrapquotes`
`\if@doublequotes@`
`\@doublequotes@true`
`\@doublequotes@false`

Now we can define the secondary programmers' macros.

We simply reserve `\Wrapquotes` here, and assign it in the user options section above.

```
\newboolean{@doublequotes@}
\newcommand*\WrapquotesNS {%
%  \DTypeout{starting wrapquotes NS}%
  \ti@wrapquotes@suck
}
\newcommand*\WrapquotesIS {%
%  \DTypeout{starting wrapquotes IS}%
  \ToggleBoolean{@doublequotes@}%
  \ti@wrapquotes@suck
}
\newcommand*\WrapquotesNN {%
%  \DTypeout{starting wrapquotes NN}%
  \ti@wrapquotes@nosuck
}
\newcommand*\WrapquotesIN {%
%  \DTypeout{starting wrapquotes IN}%
  \ToggleBoolean{@doublequotes@}%
  \ti@wrapquotes@nosuck
}
\newcommand*\WrapquotesSN [1] {%
%  \DTypeout{starting wrapquotes SN}%
  \begingroup
    \ti@open@single #1\ti@close@single@nosuck
  \endgroup
}
\newcommand*\WrapquotesDN [1] {%
%  \DTypeout{starting wrapquotes DN}%
  \begingroup
    \ti@open@double #1\ti@close@double@nosuck
  \endgroup
}
\newcommand*\WrapquotesSK [1] {% FIX: test
%  \DTypeout{starting wrapquotes SK}%
  \begingroup
    \ti@open@single #1\ti@close@single@nosuck\@%
  \endgroup
}
\ReserveCS\Wrapquotes
\ti@domelater
```

8.3 Opening quotes

`\ti@open@double`
`\ti@open@single`
`\ti@openquote`

We start by putting an opening mark in scratch `f` with a global definition.

I can't remember why it's global. In the macros that close quotes, we want to keep that information around past a group end because we're using `\aftergroup`, but that doesn't seem to apply for opening them. Best not to change what's not broke, however.

```
\newcommand\ti@open@double {%
  \gdef\sc@t@f {\textquotedblleft}%
  \ti@openquote
}
```

```
\newcommand\ti@open@single {%
  \gdef\sc@t@f {\textquoteleft}%
  \ti@openquote
}
```

Then we look ahead with scratch `a`. We are looking ahead at the first character of the contents of the `\Wrapquotes`.

```
\newcommand\ti@openquote {%
  \futurelet\sc@t@a\ti@@openquote
}
```

Insert the opening mark. Then, if we are about to open another quote, insert the space appropriate to separate contiguous quotation marks.

```
\newcommand\ti@@openquote {%
  \sc@t@f
  \ifx\sc@t@a\WrapquotesNS
%    \DTypeout{Quotation marks are doubled up (next is NS); inserting padding.}%
    \,%
  \else \ifx\sc@t@a\WrapquotesNN
%    \DTypeout{Quotation marks are doubled up (next is NN); inserting padding.}%
    \,%
  \else \ifx\sc@t@a\WrapquotesIN
%    \DTypeout{Quotation marks are doubled up (next is IN); inserting padding.}%
    \,%
  \else \ifx\sc@t@a\WrapquotesIS
%    \DTypeout{Quotation marks are doubled up (next is IS); inserting padding.}%
    \,%
  \else \ifx\sc@t@a\WrapquotesSN
%    \DTypeout{Quotation marks are doubled up (next is SN); inserting padding.}%
    \,%
  \else \ifx\sc@t@a\WrapquotesDN
%    \DTypeout{Quotation marks are doubled up (next is DN); inserting padding.}%
    \,%
  \else \ifx\sc@t@a\WrapquotesSK
%    \DTypeout{Quotation marks are doubled up (next is SK); inserting padding.}%
    \,%
  \else
  \fi \fi \fi \fi \fi \fi \fi
}
```

8.4 Closing macros that don't suck

This case that doesn't suck is easier, so we warm up with it.

`\ti@close@single@nosuck`
`\ti@close@double@nosuck`
`\ti@close@single@@nosuck`
`\ti@close@double@@nosuck`

```
\newcommand*\ti@close@single@nosuck {%
  \aftergroup\ti@close@single@@nosuck
}
\newcommand*\ti@close@double@nosuck {%
  \aftergroup\ti@close@double@@nosuck
}
\newcommand*\ti@close@single@@nosuck {%
  \gdef\sc@t@f {\textquoteright}%
  \ti@close@quote@nosuck
```

```
}
\newcommand*\ti@close@double@@nosuck {%
  \gdef\sc@t@f {\textquotedblright}%
  \ti@close@quote@nosuck
}
```

\ti@close@quote@nosuck
\if@look@nosuck@
\@look@nosuck@true
\@look@nosuck@false

To do: *Document this flag. It's a hack, we must set it before each call to* `\ti@q@ifnextcharin` *I think. What it stands for is something like the presence of the tokens* `\ti@close@single@nosuck` *and* `\ti@close@double@nosuck` *in the list of chars to look for, but since they aren't really chars they can't go in the list, so instead we set the flag. Somewhat cleaner would be putting a flag char in the list, but I can't think of what char I could safely use.*

```
\newboolean{@look@nosuck@}
\@look@nosuck@false
\newcommand\ti@close@quote@nosuck {%
%  \DTypeout{Starting ti@close@quote@nosuck}%
  \@look@nosuck@true
```

FIX Aha, but here is a good reason to leave in `.,` in our substitute for `\nospacelist`.

```
  \expandafter \ti@q@ifnextcharin \expandafter {\nospacelist} {%
%      \DTypeout{Found a nosuck no-spacer.  C=[\meaning\sc@t@c] F=[\meaning\sc@t@f]}%
      \sc@t@f
    }{% ELSE
%      \DTypeout{Found a nosuck spacer. C=[\meaning\sc@t@c] F=[\meaning\sc@t@f]}%
      \sc@t@f\space
  }%
}
```

8.5 Closing macros that suck

\ti@close@double@suck
\ti@close@single@suck

We need to look ahead beyond the `\endgroup` that ends `\Wrapquotes??`. The lookahead mechanism that gets invoked in scratch `a` below could handle looking past the `\endgroup`, but I think it is more efficient to skip it by using `\aftergroup`.

```
\newcommand\ti@close@double@suck {%
  \aftergroup\ti@close@double@@suck
}
\newcommand\ti@close@single@suck {%
  \aftergroup\ti@close@single@@suck
}
```

\ti@close@double@@suck
\ti@close@single@@suck

This part isn't so bad yet. To close the quotes, we again start with the closing mark in scratch `f`, with a global definition.

```
\newcommand\ti@close@double@@suck {%
  \gdef\sc@t@f {\textquotedblright}%
  \ti@close@quote@suck
}
\newcommand\ti@close@single@@suck {%
  \gdef\sc@t@f {\textquoteright}%
  \ti@close@quote@suck
}
```

\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
}
```

\ti@close@quote@suck Then we use \ti@q@ifnextcharin to look as far ahead as necessary for a significant character. The latest significant character found is available in scratch c. The work of handling all the cases of what we might find while looking ahead is divided up between \ti@close@quote@suck and \ti@q@ifnextcharin. \ti@close@quote@suck handles the last step in the process, and \ti@q@ifnextcharin handles all the steps up to the last.

Here is what \ti@close@quote@suck does, in English. If we find a comma or period, we put it inside the closing quote, and gobble the one we found. That is, we print out scratch c, then scratch f, then gobble a character. If we find something in the set given in \nospacelist, do not leave a space after the closing mark. That is, just print out scratch f. If we find something else, we leave a space after the closing mark. That is, print scratch f and a space.

```
\newcommand\ti@close@quote@suck {%
%  \DTypeout{Starting ti@close@quote@suck}%
  \@look@nosuck@false
  \ti@q@ifnextcharin {.,} {%
%      \DTypeout{Found a comma or period. C=[\meaning\sc@t@c] F=[\meaning\sc@t@f]}%
      \sc@t@c\sc@t@f\DGobbleM % This gobbles the original punctuation.
    }{% ELSE
%    \DTypeout {Before second ti@qifnextcharin.  C=[\meaning\sc@t@c] F=[\meaning\sc@t@f]}%
    \@look@nosuck@true
```

To do: *Using* \nospacelist *is inefficient here, since some of the cases, namely* ,.\@sptoken*, are never going to be there and shouldn't be checked for, since they are passed over by* \ti@q@ifnextcharin *before the list is compared. But it would be good to have this parallelism between abbrevs and titles.*

```
    \expandafter \ti@q@ifnextcharin \expandafter {\nospacelist} {%
%       \DTypeout{Found a suck no-spacer.  C=[\meaning\sc@t@c] F=[\meaning\sc@t@f]}%
        \sc@t@f
      }{% ELSE
%      \DTypeout{Found a suck spacer. C=[\meaning\sc@t@c] F=[\meaning\sc@t@f]}%
      \sc@t@f\space
    }%
  }%
}
```

8.6 Looking ahead

Now things are getting fun.

\ti@q@ifnextcharin These macros are modeled after the definition of \@ifnextchar which skips spaces.
\ti@q@check While looking ahead for the next significant character, these macros skip spaces,
\ti@q@ifnch \egroup, \endgroup, \check@icr, \ti@close@double and \ti@close@single
\ti@q@@ifnch while doing the right thing after each.

The first argument should be a list of tokens. If the next significant char is in the list, then the `true` clause is executed, otherwise the `false` clause is executed. The next significant char is left in scratch `c` so it can be accessed by the clauses.

The three arguments to `\ti@q@ifnextcharin` are saved in global variables because while looking ahead we must continue past the ends of groups.

FIX Not sure I need gdef for scratch `e`.

```
\newcommand\ti@q@ifnextcharin [3] {% args: charlist true false
%  \sc@toks@a{#1}%
%  \DTypeout{charlist unexpanded is =[\the\sc@toks@a]}%
  \gdef\sc@t@e {#1}%
  \gdef\sc@t@a {#2}%
  \gdef\sc@t@b {#3}%
  \ti@q@check
}
```

Having saved the arguments, we look ahead with scratch `c`. This step is not in the macro above so that we can jump back to `\ti@q@check` whenever we want to look ahead another character.

```
\newcommand\ti@q@check {%
  \futurelet\sc@t@c\ti@q@ifnch
}
```

Scratch `c` contains the current char. Scratch `d` is the action to take at the end of this macro. We attempt to order these possibilities to make `\Wrapquotes` most efficient, though it is a guess which items will be encountered most frequently.

The actions taken for each of the possibilities are the following:

`\ifvmode` Assume that the `\Wrapquotes` was the argument of a `\TextFontCommand` from certain LaTeX kernels. Gobble three more tokens expected to follow the `\ifvmode`, execute them, and continue on to look ahead another character. See documentation of `\ti@q@handle@ifvmode` for more details.

`\check@icr` This means the `\Wrapquotes` was the argument of a `\TextFontCommand`. Gobble the `\check@icr` and look ahead another character after we exit the group that the `\TextFontCommand` has given us.

`\endgroup` **and** } Pass right by an `\endgroup` or `}` and look ahead another char.

`\@sptoken` **(a non-explicit space)** Handle a non-explicit space by calling `\ti@q@handle@space`, which gobbles the space and looks ahead another char. When the user or a macro has followed the titles with an *explicit* space such as a tie, or the `\␣` or `\space` macros, we do nothing and let this be caught by the comparison to the tokens in the argument of `\ti@q@ifnextcharin`.

`\ti@close@double@suck`, `\ti@close@single@suck`

`\ti@close@double@nosuck`, **and** `\ti@close@single@nosuck` We are in a nested `\Wrapquotes`. Call `\ti@q@handle@single/double@suck/nosuck` as appropriate, which gobbles the closequotes token, adds properly-padded closing quotes to scratch `f`, and then goes on to look ahead another character.

The lookahead process stops when it finds something not on this list. Then it compares what it found to the list of characters given to `\ti@q@ifnextcharin` and executes the `true` or `false` clause as appropriate.

First we have to handle the case of finding an `\ifvmode`. We can't bundle this test in with the tests for other tokens, so it gets its own macro, `\ti@q@handle@ifvmode`, which see for details. The remaining cases are handled in `\ti@q@@ifnch`.

```
\newcommand\ti@q@ifnch {%
%  \DTypeout{The lookahead in ti@q@ifnch: [\meaning\sc@t@c]}%
  \ifx\sc@t@c\ifvmode
    \let\sc@t@d\ti@q@handle@ifvmode
  \else
    \let\sc@t@d\ti@q@@ifnch
  \fi
  \sc@t@d
}
\newcommand\ti@q@@ifnch {%
%  \DTypeout{entering ti@q@@ifnch}%
%  \expandafter\sc@toks@a\expandafter{\sc@t@c}%
%  \DTypeout{ti@q@@ifnch: C expanded once is =[\the\sc@toks@a]}%
  \ifx\sc@t@c\check@icr
%    \DTypeout{Handling check@icr}%
    \defcommand\sc@t@d [1] {%
%      \DTypeout{check@icr handler: gobbling [\meaning ##1]}%
      ##1\aftergroup\ti@q@check
    }%
  \else \ifx\sc@t@c\endgroup
%    \DTypeout{Handling endgroup}%
    \def\sc@t@d {\aftergroup\ti@q@check}%
  \else \ifx\sc@t@c\@sptoken
%    \DTypeout{Handling space}%
    \let\sc@t@d\ti@q@handle@space
  \else \ifx\sc@t@c\egroup
%    \DTypeout{Handling egroup}%
    \def\sc@t@d {\aftergroup\ti@q@check}%
  \else \ifx\sc@t@c\ti@close@double@suck
%    \DTypeout{Handling ti@close@double@suck}%
    \let\sc@t@d\ti@q@handle@double@suck
  \else \ifx\sc@t@c\ti@close@single@suck
%    \DTypeout{Handling ti@close@single@suck}%
    \let\sc@t@d\ti@q@handle@single@suck
  \else \ifx\sc@t@c\ti@close@double@nosuck
%    \DTypeout{Handling ti@close@double@nosuck}%
    \let\sc@t@d\ti@q@handle@double@nosuck
  \else \ifx\sc@t@c\ti@close@single@nosuck
%    \DTypeout{Handling ti@close@single@nosuck}%
    \let\sc@t@d\ti@q@handle@single@nosuck
  \else
```

We've handled all the lookahead cases, so now we are left with the simple comparison of the next char with the charlist.

```
              \@tempswafalse
              \expandafter   \@tfor
                \expandafter \sc@t@g
                \expandafter :%
                \expandafter =%
                             \sc@t@e
                             \do {%
```

```
                  \expandafter\ifx\sc@t@g\sc@t@c
% \DTypeout{We have a match of [\meaning\sc@t@c]
%          with [\expandafter\meaning\sc@t@g]}%
                  \@tempswatrue
                  \@break@tfor
                \else
% \DTypeout{We have NO match between [\meaning\sc@t@c]
%          with [\meaning\sc@t@g]}%
                \fi
              }%
              \if@tempswa
% \DTypeout{Choosing true clause [\meaning\sc@t@a]}%
                \let\sc@t@d\sc@t@a % the ``true'' clause
              \else
% \DTypeout{Choosing false clause [\meaning\sc@t@b]}%
                \let\sc@t@d\sc@t@b % the ``false'' clause
              \fi
  \fi \fi \fi \fi \fi \fi \fi \fi
%  \DTypeout{About to fall out of ti@q@@ifnch and do this [\meaning\sc@t@d]}%
  \sc@t@d
}
```

`\ti@q@handle@ifvmode` This is in its own macro for clarity and to avoid problems with skipping over clauses.

`\ti@q@ifnch` has to take two different kinds of LaTeX kernel into account. The 1996/12/01 and 1997/06/01 kernels used a different definition of `\DeclareTextFontCommand`:

```
%\def \DeclareTextFontCommand #1#2{%
%  \DeclareRobustCommand#1[1] {%
%    \ifmmode
%      \nfss@text{#2##1}%
%    \else
%      \leavevmode
%      {\text@command{##1}%
%       #2\check@icl ##1\ifvmode\else\check@icr\fi
%       \expandafter}%
%    \fi
%  }%
%}
```

All other kernels leave out the check for vertical mode (kernels from 1997/12/01 include it when necessary inside `\check@icr`). The macro `\ti@q@ifnch`, which will be called immediately before this point of difference, handles both cases by looking for both `\ifvmode` and `\check@icr`. For the history, see LaTeX bug report 2646.

The check for `\ifvmode` must not be part of a nested conditional. TeX can't match `\if`s with `\fi`s properly when you nest tests for `\if`-type tokens. See p. 211 of the TeXbook.

When we encounter an `\ifvmode`, we must assume we are inside a TextFontCommand declared by one of the two kernel versions mentioned above. If not, we are in an unknown situation and we will bomb. Since the error message in this case won't be helpful, we warn the user in the log file. We use scratch `d` to gobble both the `\ifvmode` and what we expect will follow the `\ifvmode`, namely

\else\check@icr\fi. After swallowing those, we reissue those same commands and then proceed with our lookahead. We want to issue those commands, which conditionally introduce an italic correction, before looking further ahead.

```
\newcommand\ti@q@handle@ifvmode {%
%  \DTypeout{Handling ifvmode}%
  \FrankenInfo{titles}
    {Handling an \string\ifvmode\space following a title.\MessageBreak
     If you now get an error that \string\sc@t@d\space does not\MessageBreak
     match its definition, this \string\ifvmode\space is\MessageBreak
     unexpected}%
  \DefName{sc@t@d} {\ifvmode\else\check@icr\fi} {%
    \ifvmode
    \else
      \check@icr
    \fi
    \aftergroup\ti@q@check
  }%
  \sc@t@d
}
```

`\ti@q@handle@space` Handle the case of a following space: gobble the space and call `\ti@q@check`.

This little bit of trickery sneaks a space in as the `\def` template, thereby causing a space following `\ti@q@handle@space` to get gobbled. We use the control character `\:` and restore its value.

```
\ReserveCS\ti@q@handle@space
\let\sc@t@a\:
\def\:{\ti@q@handle@space} \expandafter\def\: {\ti@q@check}
\let\:\sc@t@a
```

`\ti@q@handle@single@suck` `\ti@q@handle@double@suck` Handle the single and double sucking cases: gobble the closequotes token with a `\def` template, add some stuff to scratch `f` and call `\ti@q@check`. These are put in their own macros only to avoid clutter above.

```
\newcommand*\ti@q@handle@double@suck [1] {%
%  \DTypeout{handle double suck: gobbling [\meaning#1]}%
%  \DTypeout{scratch f before: [\meaning\sc@t@f]}%
  \g@addto@macro\sc@t@f {\,\textquotedblright}%
%  \DTypeout{scratch f after: [\meaning\sc@t@f]}%
  \ti@q@check
}
\newcommand*\ti@q@handle@single@suck [1] {%
%  \DTypeout{handle single suck: gobbling [\meaning#1]}%
%  \DTypeout{scratch f before: [\meaning\sc@t@f]}%
  \g@addto@macro\sc@t@f {\,\textquoteright}%
%  \DTypeout{scratch f after: [\meaning\sc@t@f]}%
  \ti@q@check
}
```

`\ti@q@handle@single@nosuck` `\ti@q@handle@double@nosuck` Handle the single and double nosucking cases. Add inter-quote space to scratch `f` and exit `\ti@q@@ifnch` with `true` or `false` depending on whether we were looking for it. We had to do it this way instead of the normal `\if` test above at the end of `\ti@q@@ifnch` because `\ti@close@double@nosuck` is more than one character long.

```
\newcommand*\ti@q@handle@double@nosuck [1] {%
% \DTypeout{handle double nosuck: gobbling [\meaning#1]}%
  \if@look@nosuck@
%      \DTypeout{And we're looking for \string\ti@close@double@nosuck.}%
       \g@addto@macro\sc@t@f {\,\textquotedblright}%
%      \DTypeout{After adding padding, F=[\meaning\sc@t@f]}%
       \let\sc@t@d\sc@t@a % the ``true'' clause of ti@q@ifnextcharin
  \else
%    \DTypeout{But we're not looking for \string\ti@close@double@nosuck.}%
%    \DTypeout{F is unchanged, F=[\meaning\sc@t@f]}%
     \let\sc@t@d\sc@t@b % the ``false'' clause of ti@q@ifnextcharin
  \fi
  \ti@q@check
}
\newcommand*\ti@q@handle@single@nosuck [1] {%
% \DTypeout{handle single nosuck: gobbling [\meaning#1]}%
  \if@look@nosuck@
%      \DTypeout{And we're looking for \string\ti@close@single@nosuck.}%
       \g@addto@macro\sc@t@f {\,\textquoteright}%
%      \DTypeout{After adding padding, F=[\meaning\sc@t@f]}%
       \let\sc@t@d\sc@t@a % the ``true'' clause of ti@q@ifnextcharin
  \else
%    \DTypeout{But we're not looking for \string\ti@close@single@nosuck.}%
%    \DTypeout{F is unchanged, F=[\meaning\sc@t@f]}%
     \let\sc@t@d\sc@t@b % the ``false'' clause of ti@q@ifnextcharin
  \fi
  \ti@q@check
}
```

9 Words and phrases

\word
\foreign
\foreignword
\phrase
\term
\defn

```
\newlet\word\textitswitch
\newlet\foreign\textitswitch
```

To do: *`\phrase` is the result of expansion here: what effect will this have on its proper nesting, and is this something to worry about?*

```
\newcommand\foreignword [1] {%
  \phrase{\word{#1}}%
}
```

The `\@` cancels the effect on spacing of any final punctuation in the argument.

```
% \newlet\phrase\WrapquotesSK -- whoops, doesn't work as intended,
% \phrase{foo}s puts a space before the following `s'

% old definition:
\newcommand\phrase [1] {%
  \textquoteleft #1\textquoteright\@%
}
\newlet\term\textitswitch
\newlet\defn\textslswitch
```

10 Titles

\book
\journal
\music
\article
\storytitle
\poemtitle
\play
\craft
\species

```
\newlet\book\textitswitch
\newlet\journal\textitswitch
\newlet\music\textitswitch
\newlet\article\Wrapquotes
\newlet\storytitle\Wrapquotes
\newlet\poemtitle\Wrapquotes
\newlet\play\textitswitch % \manualref{7.145}
\newlet\craft\textitswitch
\newlet\species\textitswitch
```

Part III
Configuration

User alterations and additions and package testing are in a configuration file.

```
\InputIfFileExists{titles.cfg}{}{}
```

The contents of the *distributed* configuration file are below.

```
\def\fileinfo{titles package configuration}
\def\fileversion{v1.4}
\def\filedate{2001/08/31}
\def\docdate{2001/08/31}
\ProvidesFile{titles.cfg}
```

11 User Customization

Put your own alterations and additions here. For example.

```
% \let\word\textslswitch
  \newlet\longpoem\textitswitch
  \newlet\film\textitswitch
  \newlet\essaytitle\Wrapquotes
  \newlet\chaptertitle\Wrapquotes
```

Part IV
Testing

11.1 Question and exclamation marks

Test string: [Title] Result: Declarative
Test string: [Title?] Result: Question or Exclamation
Test string: [Title!] Result: Question or Exclamation
Test string: [Title??] Result: Question or Exclamation
Test string: [Title!!] Result: Question or Exclamation
Test string: [Title? Title] Result: Declarative
Test string: [Title! Title] Result: Declarative
Test string: [Title!?] Result: Question or Exclamation
Test string: [Title?!] Result: Question or Exclamation
Test string: [Title? Title?] Result: Question or Exclamation
Test string: [Title? Title!] Result: Question or Exclamation
Test string: [Title! Title?] Result: Question or Exclamation
Test string: [Title?? Title] Result: Declarative
Test string: [Title!! Title] Result: Declarative

11.2 Plain

Book Title. Test.
Book Title, test.
Book Title; test.
Book Title test.
Play Title. Test.
“Play Title.” Test.
Play Title, test.
“Play Title,” test.
Play Title; test.
“Play Title”; test.
Play Title test.
“Play Title” test.
title tie
“title” tie
title explicit space
“title” explicit space
title `\space`
“title” `\space`
title/slash
“title”/slash
title italcorr
“title” italcorr

title xobey
“title” xobey

11.3 Nested beginnings

Book Title *begins first book title* and outside.
Book Title*, begins first book title*, and outside.
Book Title*. begins first book title.* and outside.
Book Title*; begins first book title*; and outside.
Play Title *begins first book title* and outside.
Play Title*, begins first book title*, and outside.
Play Title*. begins first book title.* and outside.
Play Title*; begins first book title*; and outside.
Book Title *begins first play title* and outside.
Book Title*, begins first play title*, and outside.
Book Title*. begins first play title.* and outside.
Book Title*; begins first play title*; and outside.
Play Title *begins first play title* and outside.
Play Title*, begins first play title*, and outside.
Play Title*. begins first play title.* and outside.
Play Title*; begins first play title*; and outside.

11.4 Nested endings

There are too many cases I think to test them all. I’m testing to three levels of nesting.
This is a *Book Title including* Book Title including *Book Title and ending first one* and outside.
This is a *Book Title including* Book Title including *Book Title, and ending first one*, and outside.
This is a *Book Title including* Book Title including *Book Title. and ending first one.* and outside.
This is a *Book Title including* Book Title including *Book Title; and ending first one*; and outside.
This is a *Book Title including* Book Title including *Play Title and ending first one* and outside.
This is a *Book Title including* Book Title including *Play Title, and ending first one*, and outside.
This is a *Book Title including* Book Title including *Play Title. and ending first one.* and outside.
This is a *Book Title including* Book Title including *Play Title; and ending first one*; and outside.
This is a *Book Title including* Play Title including *Book Title and ending first one* and outside.
This is a *Book Title including* Play Title including *Book Title, and ending first one*, and outside.
This is a *Book Title including* Play Title including *Book Title. and ending first one.* and outside.
This is a *Book Title including* Play Title including *Book Title; and ending first one*; and outside.

This is a *Book Title including* Play Title including *Play Title and ending first one* and outside.
This is a *Book Title including* Play Title including *Play Title, and ending first one*, and outside.
This is a *Book Title including* Play Title including *Play Title. and ending first one.* and outside.
This is a *Book Title including* Play Title including *Play Title; and ending first one*; and outside.
This is a *Play Title including* Play Title including *Play Title and ending first one* and outside.
This is a *Play Title including* Play Title including *Play Title, and ending first one*, and outside.
This is a *Play Title including* Play Title including *Play Title. and ending first one.* and outside.
This is a *Play Title including* Play Title including *Play Title; and ending first one*; and outside.
This is a *Play Title including* Play Title including *Book Title and ending first one* and outside.
This is a *Play Title including* Play Title including *Book Title, and ending first one*, and outside.
This is a *Play Title including* Play Title including *Book Title. and ending first one.* and outside.
This is a *Play Title including* Play Title including *Book Title; and ending first one*; and outside.
This is a *Play Title including* Book Title including *Play Title and ending first one* and outside.
This is a *Play Title including* Book Title including *Play Title, and ending first one*, and outside.
This is a *Play Title including* Book Title including *Play Title. and ending first one.* and outside.
This is a *Play Title including* Book Title including *Play Title; and ending first one*; and outside.
This is a *Play Title including* Book Title including *Book Title and ending first one* and outside.
This is a *Play Title including* Book Title including *Book Title, and ending first one*, and outside.
This is a *Play Title including* Book Title including *Book Title. and ending first one.* and outside.
This is a *Play Title including* Book Title including *Book Title; and ending first one*; and outside.

11.5 double and single nosuck

OS=open-single OD=open-double CS=close-single CD=close-double
The following pairs of lines in medium weight roman should look identical.
The line in typewriter font is the source text.

The following line in medium weight roman is what that source produces.
The third line is what the second line *ought* to produce:
`The word \WrapquotesDN{quoted} is quoted.`
The word “quoted” is quoted.
The word “quoted” is quoted.

`The word \WrapquotesSN{scare} is in scare quotes.`
The word ‘scare’ is in scare quotes.
The word ‘scare’ is in scare quotes.

Nesting with no abuttment:
`\WrapquotesDN{The \WrapquotesSN{quick} brown fox \WrapquotesDN{jumped} over the lazy d`
“The ‘quick’ brown fox “jumped” over the lazy dogs.”
“The ‘quick’ brown fox “jumped” over the lazy dogs.”

`\WrapquotesSN{The \WrapquotesSN{quick} brown fox \WrapquotesDN{jumped} over the lazy d`
‘The ‘quick’ brown fox “jumped” over the lazy dogs.’
‘The ‘quick’ brown fox “jumped” over the laxy dogs.’

OS+OS, CD+CS:
`\WrapquotesSN{\WrapquotesSN{The quick} brown fox jumped over the \WrapquotesDN{lazy do`
‘ ‘The quick’ brown fox jumped over the “lazy dogs.” ’
‘ ‘The quick brown fox jumped over the “lazy dogs.” ’

OS+OD, CD+CS:
`\WrapquotesSN{\WrapquotesDN{The quick} brown fox jumped over the \WrapquotesSN{lazy do`
‘ “The quick” brown fox jumped over the ‘lazy dogs.’ ’
‘ “The quick” brown fox jumped over the ‘lazy dogs.’ ’

OD+OD, CS+CD:
`\WrapquotesDN{\WrapquotesDN{The quick} brown fox jumped over the \WrapquotesSN{lazy do`
“ “The quick” brown fox jumped over the ‘lazy dogs.’ ”
“ “The quick” brown fox jumped over the ‘lazy dogs.’ ”

OS+OD, CS+CS:
`\WrapquotesSN{\WrapquotesDN{The quick} brown fox jumped over the \WrapquotesSN{lazy do`
‘ “The quick” brown fox jumped over the ‘lazy dogs.’ ’
‘ “The quick” brown fox jumped over the ‘lazy dogs.’ ’

References

University of Chicago Press. 1993. *The Chicago Manual of Style*. 14th ed. Chicago: University of Chicago Press.

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

W