The **stringstrings** Package

Extensive array of string manipulation routines for cosmetic and programming application

Steven B. Segletes

steven@arl.army.mil

2010/02/23
v1.20

Abstract

The **stringstrings** package provides a large and sundry array of routines for the manipulation of strings. The routines are developed not only for cosmetic application, such as the changing of letter cases, selective removal of character classes, and string substitution, but also for programming application, such as character look-ahead applications, argument parsing, \if-tests for various string conditions, etc. A key tenet employed during the development of this package (unlike, for comparison, the \uppercase and \lowercase routines) was to have resultant strings be "expanded" (*i.e.*, the product of an \edef), so that the **stringstrings** routines could be strung together sequentially and nested (after a fashion) to achieve very complex manipulations.

Contents

1 Motivation

There were two seminal moments that brought about my motivation to develop this package. The first was the realization of the oft cited and infamous LaTeX limitation concerning the inability to nest letter-case changes with LaTeX's intrinsic `\uppercase` and `\lowercase` routines. The second, though not diminishing its utility in many useful applications, was the inherent limitations of the coolstr package, which is otherwise a useful tool for extracting substrings and measuring string lengths.

The former is well documented and need not be delved into in great detail. Basically, as it was explained to me, `\uppercase` and `\lowercase` are expanded by LaTeX at the last possible moment, and thus attempts to capture their result for subsequent use are doomed to failure. One is forced to adopt the left-to-right (rather than nested) approach to case changes.

In the case of the coolstr package, I again want to express my admiration for the utility of this package. I briefly considered building the stringstrings package around it, but it proved unworkable, because of some intrinsic limitations. First, coolstr operates on strings, not tokens, and so in order to fool it into working on tokenized inputs, one must use the cumbersome nomenclature of

    \expandafter\substr\expandafter{\TokenizedString}{...}{...}

in order to, for example grab a substring of `\TokenizedString`. One may `\def` the result of this subroutine, and use it elsewhere in an unaltered state. However, one may not expand, via `\edef`, the result of `\substr` in order to use it as input to a subsequent string manipulation. And thus, the desire to engage in successive string manipulations of different natures (*e.g.*, capitalization of leading characters, extraction of words, reversal of character sequence, removal of character classes, etc., etc.) are not achievable in the context of coolstr.

It was this state of affairs that brought me to hunger for routines that could thoroughly manipulate strings, and yet produce their result "in the clear" (*i.e.*, in an untokenized form) which could be used as input for the next manipulation. It turns out the heart of the stringstrings package which achieves this goal is based on the simple (if much maligned) `\if` construct of LaTeX, by using successive iterations of the following construct:

    \if <*test char.*><*string*><*manipulated test char.*>\else ...\fi

in which a character at the begining of a string is tested. If a match is found, the manipulated test character is replaced at the end of the string, while the original test character is lopped off from the beginning of the string. A false result is used to proceed to a different test character. In this manner, the string may be rotated through, character by character, performing the desired manipulations. And, most importantly, the product of this operation may be placed into an `\edef`.

It turns out there was one glitch to this process (which has been successfully remedied in the stringstrings package). And that is that there are several tokenized LaTeX symbols (*e.g.*, \$, \{, \}, \AE, \oe, etc.) which expand to more than a single byte. If I was more savvy on LaTeX constructs, I would probably have known how to handle this better. But my solution was to develop my own encoding scheme wherein these problematic characters were re-encoded in my intermediate calculations as a 2-byte (escape character-escape code) combination, and only converted back into LaTeX symbols at the last moment, as the finalized strings were handed back to the user.

There are also several tokens, like \dag, \ddag, \P, \d, \t, \b, and \copyright which can not be put into an \edef construct. The solution developed for strings containing these such characters was to convert the encoded string not into an expanded \edef construct, but rather back into a tokenized form amenable to \def. The \retokenize command accomplishes this task and several others.

There was also one glitch that I have not yet been able to resolve to my full satisfaction, though I have provided a workaround. And that is the occurance of LaTeX grouping characters, { and }, that might typically occur in math mode. The problem is that the character-rotate technique that is the core of stringstrings breaks when rotating these group characters. Why?? Because a string comprised of ...{...}..., during the rotation process, will eventually become ...}......{ during an intermediate stage of character rotation. This latter string breaks LaTeX because it is not a properly constructed grouping, even if subsequent rotations would intend to bring it back into a proper construction.

And so, while stringstrings can handle certain math-mode constructs (*e.g.*, \$, ^, and _), it is unable to *directly* handle groupings that are brought about by the use of curly braces. Note that \{ and \} are handled just fine, but not { and }. As a result of this limitation regarding the use of grouping braces within strings, stringstrings support for various math symbols remains quite limited.

While it is also common to use curly braces to delimit the arguments of diacritical marks in words like m\"{u}de *etc.*, the same result can be achieved without the use of braces as m\"ude, with the proper result obtained: müde. For diacritical marks that have an alphabetic token such as the breve, given by \u, the curly braces can also be omitted, with the only change being a space required after the \u to delimit the token. Thus, c\u at becomes căt. Therefore, when manipulating strings containing diacritical marks, it is best to formulate them, if possible, without the use of curly braces.

The workaround fix I have developed, to provide grouping functions within stringstrings arguments, involves the use of newly defined tokens \LB and \RB (to be used in lieu of { and }), along with a command \retokenize. This workaround will be described subsequently, in the Disclaimers section.

2 Philosophy of Operation

There are several classes of commands that have been developed as part of the stringstrings package. In addition to **Configuration Commands**, which set parameters for subsequent string operations, there are the following command classes:

- **Commands to Manipulate Strings** – these commands take an input string or token and perform a specific manipulation on the string;

- **Commands to Extract String Information** – these commands take an input string or token, and ascertain a particular characteristic of the string; and

- **Commands to Test Strings** – these commands take an input string or token and test for a particular alphanumeric condition.

Of course, there are also **Support Commands** which are low-level routines which provide functionality to the package, which are generally not accessible to the user.

To support the intended philosophy that the user may achieve a complex string manipulation though a series of simpler manipulations (which is otherwise known as nesting), a mechanism had to be developed. True command nesting of the form \commandA{\commandB{\commandC{string}}} is **not** supported by the stringstrings package, since many of the manipulation commands make use of (and would thus inadvertantly overwrite) the same sets of variables used by other routines. Furthermore, there is that 'ol left-to-right philosophy of LaTeX to contend with.

\thestring
Instead, for the case of commands that manipulate strings, the expanded (*i.e.*, \edef'ed) result of the manipulation is placed into a string called \thestring. Then, \thestring may either be directly used as the input to a subsequent operation, or \edef'ed into another variable to save it for future use.

String manipulation commands use an optional first argument to specify what to do with the manipulated string (in addition to putting it in \thestring). Most [v] string manipulation commands default to verbose mode [v], and print out their result immediately on the assumption that a simple string manipulation is, many times, all that is required. If the user wishes to use the manipulated result as is, [q] but needs to use it later in the document, a quiet mode [q] is provided which suppresses the immediate output of \thestring.

In the absence of symbol tokens (*e.g.*, \$, \&, \oe, \^, *etc.*), the verbose and quiet modes would prove sufficient. However, when a tokenized symbol is \edef'ed, the token is expanded to the actual symbolic representation of the character. If this expanded symbol is used as part of an input string to a subsequent stringstrings manipulation routine, it gets confused, because the means to detect the token are characteristically different than the means to detect the expanded

4

[e]   symbol. Thus, if one wishes to use \thestring as an input to a subsequent manipulation routine, stringstrings provides an encoded mode [e] which places an encoded version of the resulting manipulation into \thestring. The encoded mode is also a quiet mode, since it leaves \thestring in a visually unappealing state that is intended for subsequent manipulation.

The encoded mode is not a LaTeX standard, but was developed for this application. And therefore, if the result of a stringstrings manipulation is needed as input for a routine outside of the stringstrings package, the encoded mode will be

\retokenize   of no use. For this reason (and others), the \retokenize command is provided. Its use is one of only three times that a stringstrings command returns a tokenized \def'ed string in \thestring, rather than an expanded, \edef'ed string. And in the other two cases, both call upon \retokenize.

In addition to providing tokenized strings that can be passed to other LaTeX packages, \retokenize can also remedy stringstrings problems associated with inadequate character encodings (OT1) and the use of grouping characters { and } within stringstrings arguments. This issue is discussed more fully in the Disclaimers section, and in the actual \retokenize command description.

Therefore, for complex multistage string manipulations, the recommended procedure is to perform each stage of the manipulation in encoded [e] mode, passing along \thestring to each subsequent stage of the manipulation, until the very last manipulation, which should be, at the last, performed in verbose [v] or quiet [q] modes. If the resulting manipulation is to be passed to a command outside of the stringstrings package for further manipulation (or if the string contains characters which cannot be placed into an \edef), \thestring may need to be \retokenize'ed. If concatenations of two (or more) different manipulations are to be used as input to a third manipulation, \thestring from the first manipulation will need to be immediately \edef'ed into a different variable, since \thestring will be overwritten by the second manipulation (see Table 1 for summary).

Table 1: **Execution Modes of stringstrings Commands**

Mode	Coding	Use when result is	\thestring is
[v] verbose	decoded or retokenized	final	echoed
[q] quiet	decoded or retokenized	final	not echoed
[e] encoded	encoded	intermediate	not echoed

Moving on to commands that extract string information, this class of commands (unless otherwise noted) output their result into a token which is given the

\theresult   name \theresult. This token does not contain a manipulated form of the string, but rather a piece of information about the string, such as "how many characters are in the string?", "how many words are in the string?", "how many letter 'e's are in the string?", *etc*.

The final class of stringstrings commands are the string-test commands. While some of this class of commands also store their test result in \theresult, most of these commands use the \test*condition*{*string*} \if*condition* constructs (see ifthen package) to answer true/false questions like "is the string composed entirely of lowercase characters?", "is the string's first letter capitalized?" *etc.*

3   Configuration Commands

\Treatments{*U-mode*}{*l-mode*}{*p-mode*}{*n-mode*}{*s-mode*}{*b-mode*}
\defaultTreatments
\encodetoken[*index*]{*token*}
\decodetoken[*index*]{*token*}
\+
\?

\Treatments   The command \Treatments is used to define how different classes of characters are to be treated by the command \substring, which is the brains of the stringstrings package. As will be explained in the next section, most string manipulation routines end up calling \substring, the difference between them being a matter of how these character treatments are set prior to the call. Because most string manipulation commands will set the treatments as necessary to perform their given task, and reset them to the default upon conclusion, one should set the \Treatments immediately prior to the call upon \substring.

\Treatments has six arguments, that define the mode of treatment for the six classes of characters that stringstrings has designated. All modes are one-digit integers. They are described below:

- *U-mode*— This mode defines the treatment for the upper-case characters (A–Z, Œ, Æ, Å, Ø, and Ł). A mode of 0 tells \substring to remove upper-case characters, a mode of 1 indicates to leave upper-case characters alone, and a mode of 2 indicates to change the case of upper-case characters to lower case.

- *l-mode*— This mode defines the treatment for the lower-case characters (a–z, œ, æ, å, ø, ł, and ß). A mode of 0 tells \substring to remove lower-case characters, a mode of 1 indicates to leave lower-case characters alone, and a mode of 2 indicates to change the case of lower-case characters to upper case. In the case of the eszett character (ß), there is no uppercase equivalent, and so an *l-mode* of 2 will leave the eszett unchanged.

- *p-mode*— This mode defines the treatment for the punctuation characters. stringstrings defines the punctuation characters as ; : ' " , . ? ' and ! A mode of 0 tells \substring to remove punctuation characters, while a mode of 1 indicates to leave punctuation characters as is.

- *n-mode*— This mode defines the treatment for the numerals (0–9). A mode

of 0 tells \substring to remove numerals, while a mode of 1 indicates to leave numerals as is.

- *s-mode*— This mode defines the treatment for the symbols. stringstrings defines symbols as the following characters and diacritical marks: / * ( ) - = + [ ] < > & \& \% \# \{ \} _ \$ † ‡ § ¶ L £ © x̌ x̂ x̃ ẍ x̀ x́ x̄ ẋ x̆ x̌ x̋ x̧ x x̂x x̱ as well as @, math and text carats, and the pipe symbol. A mode of 0 tells \substring to remove symbols, while a mode of 1 indicates to leave symbols as is. Note that the $ symbol, when used for entering and exiting math mode, is left intact, regardless of *s-mode*.

- *b-mode*— This mode defines the treatment for blankspaces. A mode of 0 tells \substring to remove blankspaces, while a mode of 1 indicates to leave blankspaces as is. The treatment apples to both soft ( ) as well as hard (˜) spaces.

\defaultTreatments   The command \defaultTreatments resets all treatment modes to their default settings, which are to leave individual characters unaltered by a string manipulation.

\encodetoken
\decodetoken   The commands \encodetoken and \decodetoken have been introduced in stringstrings v1.20. Prior to this version, the ability of stringstrings to handle a particular token was dependent on whether provisions for encoding that token had been explicitly hardwired into the stringstrings package. A large number of alphabetic and diacritical marks had reserved encodings set aside in stringstrings for their treatment (see next paragraph or Table 2 for their enumeration). However, requests would invariable come in for treating yet another token, which required a new stringstrings release for each revision. The command \encodetoken allows the user to specify an arbitrary token, to be assigned to the reserved encoding slot associated with the index (permissible indices are in the range 1–3, 1 being the default). Once assigned an encoding slot, a token may be successfully manipulated in stringstrings routines. Once stringstrings manipulation is complete, the token must undergo a \decodetoken operation in order for that token to be reset to a normal LaTeX token again (lest it display in its encoded stringstrings form).

\+
\?   The commands \+ and \? are a pair that work in tandem to turn on stringstrings encoding and turn off stringstrings encoding, respectively. Generally, the user will not need these commands unless he is writing his own routines to take advantage of the stringstrings library. After \+ is called, tokens which would otherwise expand to multi-byte sequences are instead encoded according to the stringstrings methodology. The affected tokens include \$ \^ \" \{ \} _ \dag \ddag \P \S \ss \AA \aa \O \o \AE \ae \OE, \oe, \~, \`, \', \=, \., \u, \v, \H, \c, \d, \t, \b, \copyright, \pounds, \L, \l, and \ss. In addition, pipes, text carats, and hard spaces (˜) are encoded as well. The command \? restores the standard LaTeX encoding for these tokens.

4   Commands to Manipulate Strings

These commands take an input string or token and perform a specific manipulation on the string. They include:

\substring[*mode*]{*string*}{*min*}{*max*}
\caseupper[*mode*]{*string*}
\caselower[*mode*]{*string*}
\solelyuppercase[*mode*]{*string*}
\solelylowercase[*mode*]{*string*}
\changecase[*mode*]{*string*}
\noblanks[*mode*]{*string*}
\nosymbolsnumerals[*mode*]{*string*}
\alphabetic[*mode*]{*string*}
\capitalize[*mode*]{*string*}
\capitalizewords[*mode*]{*string*}
\capitalizetitle[*mode*]{*string*}
\addlcword{*word*}
\addlcwords{*word1 word2 word3 . . .*}
\resetlcwords
\reversestring[*mode*]{*string*}
\convertchar[*mode*]{*string*}{*from-char*}{*to-string*}
\convertword[*mode*]{*string*}{*from-string*}{*to-string*}
\rotateword[*mode*]{*string*}
\removeword[*mode*]{*string*}
\getnextword[*mode*]{*string*}
\getaword[*mode*]{*string*}{*n*}
\rotateleadingspaces[*mode*]{*string*}
\removeleadingspaces[*mode*]{*string*}
\stringencode[*mode*]{*string*}
\stringdecode[*mode*]{*string*}
\gobblechar[*mode*]{*string*}
\gobblechars[*mode*]{*string*}{*n*}
\retokenize[*mode*]{*string*}

Unless otherwise noted, the *mode* may take one of three values: [v] for verbose mode (generally, the default), [q] for quiet mode, and [e] for encoded mode. In all cases, the result of the operation is stored in \thestring. In verbose mode, it is also output immediately (and may be captured by an \edef). In quiet mode, no string is output, though the result still resides in \thestring. Encoded mode is also a quiet mode. However, the encoded mode saves the string with its stringstrings encodings. Encoded mode indicates that the result is an intermediate result which will be subsequently used as input to another stringstrings manipulation.

\substring   The command \substring is the brains of the stringstrings package, in that most of the commands in this section call upon \substring in one form or another.

Nominally, the routine returns a substring of *string* between the characters defined by the integers *min* and *max*, inclusive. However, the returned substring is affected by the designated `\Treatments` which have been defined for various classes of characters. Additionally, a shorthand of $ may be used in *min* and *max* to define END-OF-STRING, and the shorthand $–*integer* may be used to define an offset of *integer* relative to the END-OF-STRING.

Regardless of how many bytes a LaTeX token otherwise expands to, or how many characters are in the token name, each LaTeX symbol token counts as a single character for the purposes of defining the substring limits, *min* and *max*.

While the combination of `\Treatments` and `\substring` are sufficient to achieve a wide array of character manipulations, many of those possibilities are useful enough that separate commands have been created to describe them, for convenience. Several of the commands that follow fall into this category.

`\caseupper` The command `\caseupper` takes the input string or token, and converts all lowercase characters in the string to uppercase. All other character classes are left untouched. Default mode is [v].

`\caselower` The command `\caselower` takes the input string or token, and converts all uppercase characters in the string to lowercase. All other character classes are left untouched. Default mode is [v].

`\solelyuppercase` The command `\solelyuppercase` is similar to `\caseupper`, except that all punctuation, numerals, and symbols are discarded from the string. Blankspaces are left alone, and lowercase characters are converted to uppercase. Default mode is [v].

`\solelylowercase` The command `\solelylowercase` is similar to `\caselower`, except that all punctuation, numerals, and symbols are discarded from the string. Blankspaces are left alone, and uppercase characters are converted to lowercase. Default mode is [v].

`\changecase` The command `\changecase` switches lower case to upper case and upper case to lower case. All other characters are left unchanged. Default mode is [v].

`\noblanks` The command `\noblanks` removes blankspaces (both hard and soft) from a string, while leaving other characters unchanged. Default mode is [v].

`\nosymbolsnumerals` The command `\nosymbolsnumerals` removes symbols and numerals from a string, while leaving other characters unchanged. Default mode is [v].

`\alphabetic` The command `\alphabetic` discards punctuation, symbols, and numerals, while retaining alphabetic characters and blankspaces. Default mode is [v].

`\capitalize` The command `\capitalize` turns the first character of *string* into its upper case, if it is alphabetic. Otherwise, that character will remain unaltered. Default mode is [v].

`\capitalizewords` The command `\capitalizewords` turns the first character of every word in *string* into its upper case, if it is alphabetic. Otherwise, that character will remain unaltered. For the purposes of this command, "the first character of a word" is

defined as either the first character of the string, or the first non-blank character that follows one or more blankspaces. Default mode is [v].

\capitalizetitle   The command \capitalizetitle is a command similar to \capitalizewords, except that words which have been previously designated as "lower-case words" are not capitalized (*e.g.*, prepositions, conjunctions, *etc.*). In all cases, the first word of the string is capitalized, even if it is on the lower-case word list. Words
\addlcword   are added to the lower-case word list with the commands \addlcword, in the case
\addlcwords   of a single word, or with \addlcwords, in the case of multiple (space-separated) words. Because the addition of many words to the lower-case list can substantially slow-down the execution of the \capitalizetitle command, the command
\resetlcwords   \resetlcwords has been added to allow the user to zero out the lower-case word list.

\reversestring   The command \reversestring reverses the sequence of characters in a string, such that what started as the first character becomes the last character in the manipulated string, and what started as the last character becomes the first character. Default mode is [v].

\convertchar   The command \convertchar is a substitution command in which a specified match character in the original string (*from-char*) is substituted with a different string (*to-string*). All occurances of *from-char* in the original string are replaced. The *from-char* can only be a single character (or tokenized symbol), whereas *to-string* can range from the null-string (*i.e.*, character removal) to a single character (*i.e.*, character substitution) to a complete multi-character string. Default mode is [v].

\convertword   The command \convertword is a substitution command in which a specified match string in the original string (*from-string*) is substituted with a different string (*to-string*). All occurances of *from-string* in the original string are replaced. If *from-string* includes spaces, use hard-space (~) characters instead of blanks. Default mode is [v].

\rotateword   The command \rotateword takes the first word of *string* (and its leading and trailing spaces) and rotates them to the end of the string. Care must be taken to have a blankspace at the beginning or end of *string* if one wishes to retain a blankspace word separator between the original last word of the string and the original first word which has been rotated to the end of the string. Default mode is [v].

\removeword   The command \removeword removes the first word of *string*, along with any of its leading and trailing spaces. Default mode is [v].

\getnextword   The command \getnextword returns the next word of *string*. In this case, "word" is a sequence of characters delimited either by spaces or by the beginning or end of the string. Default mode is [v].

\getaword   The command \getaword returns a word of *string* defined by the index, *n*. In this case, "word" is a sequence of characters delimited either by spaces or by the first or last characters of the string. If the index, *n*, requested exceeds the number of words available in the string, the index wraps around back to the first

argument of the string, such that asking for the tenth word of an eight word string will return the second word of the string. Default mode is [v].

\rotateleadingspaces   The command \rotateleadingspaces takes any leading spaces of the string and rotates them to the end of the string. Default mode is [v].

\removeleadingspaces   The command \removeleadingspaces removes any leading spaces of the string. Default mode is [v].

\stringencode   The command \stringencode returns a copy of the string that has been encoded according to the stringstrings encoding scheme. Because an encoded string is an intermediate result, the default mode for this command is [e].

\stringdecode   The command \stringdecode returns a copy of the string that has been decoded. Default mode is [v].

\gobblechar   The command \gobblechar returns a string in which the first character of *string* has been removed. Unlike the LaTeX system command \@gobble which removes the next **byte** in the input stream, \gobblechar not only takes an argument as the target of its gobble, but also removes one **character**, regardless of whether that character is a single-byte or multi-byte character. Because this command may have utility outside of the stringstrings environment, the result of this command is retokenized (*i.e.*, def'ed) rather than expanded (*i.e.*, edef'ed). Default mode is [q]. Mode [e] is not recognized.

\gobblechars   The command \gobblechars returns a string in which the first *n* characters of *string* have been removed. Like \gobblechar, \gobblechars removes characters, regardless of whether those characters are single-byte or multi-byte characters. Likewise, the result of this command is retokenized (*i.e.*, def'ed) rather than expanded (*i.e.*, edef'ed). Default mode is [q]. Mode [e] is not recognized.

\retokenize   The command \retokenize takes a string that is encoded according to the stringstrings encoding scheme, and repopulates the encoded characters with their LaTeX tokens. This command is particularly useful for exporting a string to a routine outside of the stringstrings library or if the string includes the following characters: \{, \}, \|, \dag, \ddag, \d, \t, \b, \copyright, and \P. Default mode is [q]. Mode [e] is not recognized.

5   Commands to Extract String Information

These commands take an input string or token, and ascertain a particular characteristic of the string. They include:

\stringlength[*mode*]{*string*}
\findchars[*mode*]{*string*}{*match-char*}
\findwords[*mode*]{*string*}{*match-string*}
\whereischar[*mode*]{*string*}{*match-char*}
\whereisword[*mode*]{*string*}{*match-string*}
\wordcount[*mode*]{*string*}

11

\getargs[*mode*]{*string*}

Commands in this section return their result in the string \theresult, unless otherwise specified. Unless otherwise noted, the *mode* may take one of two values: [v] for verbose mode (generally, the default), and [q] for quiet mode. In both cases, the result of the operation is stored in \theresult. In verbose mode, it is also output immediately (and may be captured by an \edef). In quiet mode, no string is output, though the result still resides in \theresult.

\stringlength   The command \stringlength returns the length of *string* in characters (not bytes). Default mode is [v].

\findchars   The command \findchars checks to see if the character *match-char* occurs anywhere in *string*. The number of occurances is stored in \theresult and, if in verbose mode, printed. If it is desired to find blankspaces, *match-char* should be set to {~} and not { }. Default mode is [v].

\findwords   The command \findwords checks to see if the string *match-string* occurs anywhere in *string*. The number of occurances is stored in \theresult and, if in verbose mode, printed. If it is desired to find blankspaces, those characters in *match-string* should be set to hardspaces (*i.e.*, tildes) and not softspaces (*i.e.*, blanks), regardless of how they are defined in *string*. Default mode is [v].

\whereischar   The command \whereischar checks to see where the character *match-char* first occurs in *string*. The location of that occurance is stored in \theresult and, if in verbose mode, printed. If the character is not found, \theresult is set to a value of 0. If it is desired to find blankspaces, *match-char* should be set to {~} and not { }. Default mode is [v].

\whereisword   The command \whereisword checks to see where the string *match-string* first occurs in *string*. The location of that occurance is stored in \theresult and, if in verbose mode, printed. If *match-string* is not found, \theresult is set to a value of 0. If it is desired to find blankspaces, those characters in *match-string* should be set to hardspaces (*i.e.*, tildes) and not softspaces (*i.e.*, blanks), regardless of how they are defined in *string*. Default mode is [v].

\wordcount   The command \wordcount counts the number of space-separated words that occur in *string*. Default mode is [v].

\getargs   The command \getargs mimics the Unix command of the same name, in that it parses *string* to determine how many arguments (*i.e.*, words) are in *string*, and extracts each word into a separate variable. The number of arguments is placed in \narg and the individual arguments are placed in variables of the name \argi, \argii, \argiii, \argiv, etc. This command may be used to facilitate simply the use of multiple optional arguments in a LaTeX command, for example \mycommand[*option1 option2 option3*]{*argument*}. In this case, \mycommand should exercise \getargs{#1}, with the result being that *option1* is stored in \argi, *etc.* The command \mycommand may then proceed to parse the optional arguments and branch accordingly. Default mode is [q]; [e] mode is permitted, while [v] mode is disabled.

6 Commands to Test Strings

These commands take an input string or token and test for a particular alphanumeric condition. They include:

\isnextbyte[*mode*]{*match-byte*}{*string*}
\testmatchingchar{*string*}{*n*}{*match-char*}
\testcapitalized{*string*}
\testuncapitalized{*string*}
\testleadingalpha{*string*}
\testuppercase{*string*}
\testsolelyuppercase{*string*}
\testlowercase{*string*}
\testsolelylowercase{*string*}
\testalphabetic{*string*}

\isnextbyte   The command \isnextbyte tests to see if the first byte of *string* equals *match-byte*. It is the only string-testing command in this section which does not use the ifthen test structure for its result. Rather, \isnextbyte returns the result of its test as a T or F in the string \theresult. More importantly, and unlike other stringstrings commands, \isnextbyte is a *byte* test and not a *character* test. This means that, while \isnextbyte operates very efficiently, it cannot be used to directly detect multi-byte characters like \\$, \\^, \\{, \\}, _, \dag, \ddag, \AE, \ae, \OE, \oe, *etc.* (\isnextbyte will give false positives or negatives when testing for these multi-byte characters). The default mode of \isnextbyte is [v].

\testmatchingchar   If a character needs to be tested, rather than a byte, \testmatchingchar should be used. The command \testmatchingchar is used to ascertain whether character *n* of *string* equals *match-char* or not. Whereas \isnextbyte checks only a *byte*, \testmatchingchar tests for a *character* (single- or multi-byte character). After the test is called, the action(s) may be called out with \ifmatchingchar *true-code* \else *false-code* \fi.

\testcapitalized   The command \testcapitalized is used to ascertain whether the first character of *string* is capitalized or not. If the first character is non-alphabetic, the test will return FALSE. After the test is called, the action(s) may be called out with \ifcapitalized *true-code* \else *false-code* \fi.

\testuncapitalized   The command \testuncapitalized is used to ascertain whether the first character of *string* is uncapitalized. If the first character is non-alphabetic, the test will return FALSE. After the test is called, the action(s) may be called out with \ifuncapitalized *true-code* \else *false-code* \fi.

\testleadingalpha   The command \testleadingalpha is used to ascertain whether the first character of *string* is alphabetic. After the test is called, the action(s) may be called out with \ifleadingalpha *true-code* \else *false-code* \fi.

\testuppercase   The command \testuppercase is used to ascertain whether all the alphabetic characters in *string* are uppercase or not. The presence of non-alphabetic characters in *string* does not falsify the test, but are merely ignored. However, a

string completely void of alphabetic characters will always test FALSE. After the test is called, the action(s) may be called out with \ifuppercase *true-code* \else *false-code* \fi.

\testsolelyuppercase The command \testsolelyuppercase is used to ascertain whether *all* the characters in *string* are uppercase or not. The presence of non-alphabetic characters in *string* other than blankspaces will automatically falsify the test. Blankspaces are ignored. However, a null string or a string composed solely of blankspaces will also test FALSE. After the test is called, the action(s) may be called out with \ifsolelyuppercase *true-code* \else *false-code* \fi.

\testlowercase The command \testlowercase is used to ascertain whether all the alphabetic characters in *string* are lowercase or not. The presence of non-alphabetic characters in *string* does not falsify the test, but are merely ignored. However, a string completely void of alphabetic characters will always test FALSE. After the test is called, the action(s) may be called out with \iflowercase *true-code* \else *false-code* \fi.

\testsolelylowercase The command \testsolelylowercase is used to ascertain whether *all* the characters in *string* are lowercase or not. The presence of non-alphabetic characters in *string* other than blankspaces will automatically falsify the test. Blankspaces are ignored. However, a null string or a string composed solely of blankspaces will also test FALSE. After the test is called, the action(s) may be called out with \ifsolelylowercase *true-code* \else *false-code* \fi.

\testalphabetic The command \testalphabetic is used to ascertain whether *all* the characters in *string* are alphabetic or not. The presence of non-alphabetic characters in *string* other than blankspaces will automatically falsify the test. Blankspaces are ignored. However, a null string or a string composed solely of blankspaces will also test FALSE. After the test is called, the action(s) may be called out with \ifalphabetic *true-code* \else *false-code* \fi.

7 Disclaimers

Now that we have described the commands available in the stringstrings package, it is appropriate to lay out the quirks and warnings associated with the use of the package.

First, stringstrings is currently set to handle a string no larger than 500 characters. A user could circumvent this, presumably, by editing the style package to \@MAXSTRINGSIZE increase the value of \@MAXSTRINGSIZE .

It is important to remember that stringstrings follows the underlying rules of LaTeX. Therefore, a passed string could not contain a raw % as part of it, because it would, in fact, comment out the remainder of the line. Naturally, the string may freely contain instances of \%.

Tokens that take two or more characters to express (*e.g.*, \#, \oe, \ddag, *etc.*) are **counted as a single character** within the string. The rule applies if you

14

wanted to know the length of a string that was populated with such tokens, or wanted to extract a substring from a such a string. Of course, the exception that makes the rule is that of diacritical marks, which count as separate symbols from the characters they mark. For example, `\^a` counts as two characters, because the `a` is really just the operand of the `\^` token, even though the net result looks like a single character (â).

Consistent with LaTeX convention, groups of spaces are treated as a single blankspace, unless encoded with ~ characters. And finally, again consistent with the way LaTeX operates, the space that follows an alphabetic token is not actually a space in the string, but serves as the delimiter to the token. Therefore, `\OE dipus` (Œdipus) has a length of six characters, one for the `\OE` and five for the `dipus`. The intervening space merely closes out the `\OE` token, and does not represent a space in the middle of the string.

One quirk worthy of particular note concerns the tabbing character, meaning & as opposed to `\&` (which is handled without problem). As of version 1.01, stringstrings has the capability to operate on arguments containg the ampersand &, normally reserved as the LaTeX tabbing character. However, one adverse by-product is that & characters returned in `\thestring` lose their catcode-4 value, and thus lose their ability to function as tabbing characters. In the following example,

```
\caseupper[q]{a & b & c & d}
\begin{tabular}{|l|c|c|c|}
\hline
\thestring\\
\hline
\end{tabular}
```

will produce A & B & C & D instead of the desired A | B | C | D .

In the `\substring` command, no tests are performed to guarantee that the lower limit, *min*, is less than the upper limit, *max*, or that *min* is even positive. However, the upper limit, *max*, is corrected, if set larger than the string length. Also, the use of the '$' symbol to signify the last character of the string and '$–$n$' to denote an offset of n characters from the end of the string can be helpful in avoiding the misindexing of strings.

Table 2 shows a variety of characters and tokens, some of which pose a challenge to stringstrings manipulations. In all cases, a solution or workaround is provided. For symbols in the top two categories, the workaround solution includes the use of retokenized strings instead of expanded strings. For symbols in the next two categories, use of T1 encoding or retokenizing provides a satisfactory solution. In the bottom three categories, because of stringstrings encoded [e] mode, there is nothing to impede the use of these characters in stringstrings arguments, if encoded [e] mode is employed for intermediate calculations. Some of the details of these problematic cases is described below.

Table 2: **Problematic Characters/Tokens and stringstrings Solutions**

LaTeX	Symbol/Name	Problem/Solution
{ }	begin group end group	Cannot use { and } in stringstrings arguments. However, use \LB...\RB in lieu of {...}; manipulate string in [e] mode & \retokenize
\dag \ddag \P \d \t \b \copyright	† Dagger ‡ Double Dagger ¶ Pilcrow x̣ Underdot x̂x Joining Arch x̲ Letter Underline © Copyright	Cannot \edef these tokens; Thus, [v] mode fails with both OT1 and T1 encoding; manipulate string in [e] mode & \retokenize
_ \{ \} \S \c \pounds	‗ Underscore { Left Curly Brace } Right Curly Brace § Section Symbol x̧ Cedilla £ Pounds	Cannot \edef with OT1 encoding; either \renewcommand\encodingdefault{T1}, or manipulate string in [e] mode & \retokenize. With OT1, \S, \c and \pounds break stringstrings [v] mode.
|	stringstrings Pipe Char. | (T1) — (OT1)	Distinct from |, the stringstrings encoded-escape character
\$ \carat \^ \' \" \~ \` \. \= \u \v \H \ss \AE \ae \OE \oe \AA \aa \O \o \L \l ~	$ Dollar ˆ (text mode) x̂ Circumflex x́ Acute ẍ Umlaut x̃ Tilde x̀ Grave ẋ Overdot x̄ Macron x̆ Breve x̌ Caron x̋ Double Acute ß Eszett Æ æ æsc Œ œ œthel Å å angstrom Ø ø slashed O Ł ł barred L Hardspace	Either cannot \edef, or cannot identify uniquely with \if construct, or expanded character is more than one byte. *However*, Use these characters freely, stringstrings encoding functions transparently with them. \retokenize also works
$ ^ _	begin/end math mode math superscript math subscript	These characters pose no difficulties; However, cannot extract substring that breaks in middle of math mode. Other math mode symbols NOT supported.
&	ampersand	Version 1.01 stringstrings can manipulate the ampersand. However, returned strings containing the & character lose their catcode-4 status, making them unfit for use as tabbing characters.

16

Not surprisingly, you are not allowed to extract a substring of a string, if it breaks in the middle of math mode, because a substring with only one $ in it cannot be \edef'ed.

There are a few potential quirks when using LaTeX's native OT1 character encoding, most of which can be circumvented by using the more modern T1 encoding (accessed via \renewcommand\encodingdefault{T1} in the document preamble). The quirks arise because there are several characters that, while displayable in LaTeX, are not part of the OT1 character encoding. The characters include \{, \}, and the | symbol (accessed in stringstrings via \|). When using stringstrings to manipulate strings containing these characters in the presence of OT1 encoding, they come out looking like –, ″, and —, respectively. However, if the T1 encoding fix is not an option for you, you can also work around this problem by \retokenize'ing the affected string (the \retokenize command is provided to convert encoded, expanded strings back into tokenized form, if need be).

Likewise, for both OT1 and T1 encoding, the characters † (\dag), ‡ (\ddag), ¶ (\P), . (\d), ˆ (\t), _ (\b), and © (\copyright) cannot be in the argument of an \edef expression. For manipulated strings including these characters, \retokenize is the only option available to retain the integrity of the string.

As discussed thoroughly in the previous section, an "encoded" form of the string manipulation routines is provided to prevent the undesirable circumstance of passing an \edef'ed symbol as input to a subsequent manipulation. Likewise, never try to "decode" an already "decoded" string.

When stringstrings doesn't understand a token, it is supposed to replace it with a period. However, some undecipherable characters may inadvertantly be replaced with a space, instead. Of course, neither of these possibilities is any comfort to the user.

As mentioned already, stringstrings cannot handle curly braces that are used for grouping purposes, a circumstance which often arises in math mode. Nonetheless, \LB and \RB may be used within stringstrings arguments in lieu of grouping braces, *if the final result is to be retokenized*. Thus, \caselower[e]{$X^\LB Y + Z\RB$} followed by \convertchar[e]{\thestring}{x}{(1+x)}, when finished up with the following command, \retokenize[v]{\thestring} yields as its result: $(1 + x)^{y + z}$.

One might ask, "why not retokenize everything, instead of using the [v] mode of the stringstrings routines?" While one *could* do this, the answer is simply that \retokenize is a computationally intensive command, and that it is best used, therefore, only when the more efficient methods will not suffice. In many, if not most cases, strings to be manipulated will be solely composed of alphanumeric characters which don't require the use of \retokenize, T1 encoding, or even stringstrings encoding.

Despite these several disclaimers and workarounds required when dealing with problematic characters, I hope you find the stringstrings architecture and feel to be straightforward and useful. There is only one thing left, and that is to dissect

the code. . . and so here we go.

8 Code Listing

I'll try to lay out herein the workings of the stringstrings style package.

```
1 ⟨*package⟩
2
3 %%%% INITIALIZATIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 \catcode`\&=12
```

ifthen   This package makes wide use of the ifthen style package.

```
5 \usepackage{ifthen}
```

\@MAXSTRINGSIZE   The parameter \@MAXSTRINGSIZE defines the maximum allowable string size that stringstrings can operate upon.

```
6 \def\@MAXSTRINGSIZE{500}

7 \def\endofstring{@E@o@S@}%
8 \def\undecipherable{.}% UNDECIPHERABLE TOKENS TO BE REPLACED BY PERIOD
9 \def\@blankaction{\BlankSpace}
```

Save the symbols which will get redefined stringstrings encoding.

```
10 \let\SaveDollar\$
11 \let\SaveHardspace~
12 \let\SaveCircumflex\^
13 \let\SaveTilde\~
14 \let\SaveUmlaut\"
15 \let\SaveGrave\`
16 \let\SaveAcute\'
17 \let\SaveMacron\=
18 \let\SaveOverdot\.
19 \let\SaveBreve\u
20 \let\SaveCaron\v
21 \let\SaveDoubleAcute\H
22 \let\SaveCedilla\c
23 \let\SaveUnderdot\d
24 \let\SaveArchJoin\t
25 \let\SaveLineUnder\b
26 \let\SaveCopyright\copyright
27 \let\SavePounds\pounds
28 \let\SaveLeftBrace\{
29 \let\SaveRightBrace\}
30 \let\SaveUnderscore\_
31 \let\SaveDagger\dag
32 \let\SaveDoubleDagger\ddag
```

18

```
33 \let\SaveSectionSymbol\S
34 \let\SavePilcrow\P
35 \let\SaveAEsc\AE
36 \let\Saveaesc\ae
37 \let\SaveOEthel\OE
38 \let\Saveoethel\oe
39 \let\SaveAngstrom\AA
40 \let\Saveangstrom\aa
41 \let\SaveSlashedO\O
42 \let\SaveSlashedo\o
43 \let\SaveBarredL\L
44 \let\SaveBarredl\l
45 \let\SaveEszett\ss
46 \let\SaveLB{
47 \let\SaveRB}
```

The BlankSpace character is the only character which is reencoded with a 1-byte re-encoding...in this case the Œ character.

```
48 \def\EncodedBlankSpace{\SaveOEthel}
49 \edef\BlankSpace{ }
```

All other reencoded symbols consist of 2 bytes: an escape character plus a unique code. The escape character is a pipe symbol. the unique code comprises either a single number, letter, or symbol.

```
50 \def\EscapeChar{|}
51
52 % |0 IS AN ENCODED |, ACCESSED VIA \|
53 \def\PipeCode{0}
54 \def\EncodedPipe{\EscapeChar\PipeCode}
55 \def\Pipe{|}
56 \let\|\EncodedPipe
57
58 % |1 IS AN ENCODED \$
59 \def\DollarCode{1}
60 \def\EncodedDollar{\EscapeChar\DollarCode}
61 % THE FOLLOWING IS NEEDED TO KEEP OT1 ENCODING FROM BREAKING;
62 % IT PROVIDES AN ADEQUATE BUT NOT IDEAL ENVIRONMENT FOR T1 ENCODING
63 \def\Dollar{\symbol{36}}
64 % THE FOLLOWING IS BETTER FOR T1 ENCODING, BUT BREAKS OT1 ENCODING
65 %\def\Dollar{\SaveDollar}
66
67 % |W IS RESERVED TO BE ASSIGNED TO AN ARBITRARY TOKEN
68 \def\UvariCode{W}
69 \def\EncodedUvari{\EscapeChar\UvariCode}
70 \def\Uvari{Uvari}
71 \let\uvari\EncodedUvari
72
73 % |X IS RESERVED TO BE ASSIGNED TO AN ARBITRARY TOKEN
74 \def\UvariiCode{X}
```

```
75 \def\EncodedUvarii{\EscapeChar\UvariiCode}
76 \def\Uvarii{Uvarii}
77 \let\uvarii\EncodedUvarii
78
79 % |Y IS RESERVED TO BE ASSIGNED TO AN ARBITRARY TOKEN
80 \def\UvariiiCode{Y}
81 \def\EncodedUvariii{\EscapeChar\UvariiiCode}
82 \def\Uvariii{Uvariii}
83 \let\uvariii\EncodedUvariii
84
85 % |2 IS AN ENCODED ^ FOR USE IN TEXT MODE, ACCESSED VIA \carat
86 \def\CaratCode{2}
87 \def\EncodedCarat{\EscapeChar\CaratCode}
88 \def\Carat{\symbol{94}}
89 \let\carat\EncodedCarat
90
91 % |4 IS AN ENCODED \{
92 \def\LeftBraceCode{4}
93 \def\EncodedLeftBrace{\EscapeChar\LeftBraceCode}
94 % THE FOLLOWING IS NEEDED TO KEEP OT1 ENCODING FROM BREAKING;
95 % IT PROVIDES AN ADEQUATE BUT NOT IDEAL ENVIRONMENT FOR T1 ENCODING
96 \def\LeftBrace{\symbol{123}}
97 % THE FOLLOWING IS BETTER FOR T1 ENCODING, BUT BREAKS OT1 ENCODING
98 %\def\LeftBrace{\SaveLeftBrace}
99
100 % |5 IS AN ENCODED \}
101 \def\RightBraceCode{5}
102 \def\EncodedRightBrace{\EscapeChar\RightBraceCode}
103 % THE FOLLOWING IS NEEDED TO KEEP OT1 ENCODING FROM BREAKING;
104 % IT PROVIDES AN ADEQUATE BUT NOT IDEAL ENVIRONMENT FOR T1 ENCODING
105 \def\RightBrace{\symbol{125}}
106 % THE FOLLOWING IS BETTER FOR T1 ENCODING, BUT BREAKS OT1 ENCODING
107 %\def\RightBrace{\SaveRightBrace}
108
109 % |6 IS AN ENCODED \_
110 \def\UnderscoreCode{6}
111 \def\EncodedUnderscore{\EscapeChar\UnderscoreCode}
112 \def\Underscore{\symbol{95}}
113 %\def\Underscore{\SaveUnderscore}
114
115 % |7 IS AN ENCODED \^
116 \def\CircumflexCode{7}
117 \def\EncodedCircumflex{\EscapeChar\CircumflexCode}
118 \def\Circumflex{\noexpand\SaveCircumflex}
119
120 % |8 IS AN ENCODED \~
121 \def\TildeCode{8}
122 \def\EncodedTilde{\EscapeChar\TildeCode}
123 \def\Tilde{\noexpand\SaveTilde}
124
```

```
125 % |" IS AN ENCODED \"
126 \def\UmlautCode{"}
127 \def\EncodedUmlaut{\EscapeChar\UmlautCode}
128 \def\Umlaut{\noexpand\SaveUmlaut}
129
130 % |' IS AN ENCODED \'
131 \def\GraveCode{'}
132 \def\EncodedGrave{\EscapeChar\GraveCode}
133 \def\Grave{\noexpand\SaveGrave}
134
135 % |' IS AN ENCODED \'
136 \def\AcuteCode{'}
137 \def\EncodedAcute{\EscapeChar\AcuteCode}
138 \def\Acute{\noexpand\SaveAcute}
139
140 % |= IS AN ENCODED \=
141 \def\MacronCode{=}
142 \def\EncodedMacron{\EscapeChar\MacronCode}
143 \def\Macron{\noexpand\SaveMacron}
144
145 % |. IS AN ENCODED \.
146 \def\OverdotCode{.}
147 \def\EncodedOverdot{\EscapeChar\OverdotCode}
148 \def\Overdot{\noexpand\SaveOverdot}
149
150 % |u IS AN ENCODED \u
151 \def\BreveCode{u}
152 \def\EncodedBreve{\EscapeChar\BreveCode}
153 \def\Breve{\noexpand\SaveBreve}
154
155 % |v IS AN ENCODED \v
156 \def\CaronCode{v}
157 \def\EncodedCaron{\EscapeChar\CaronCode}
158 \def\Caron{\noexpand\SaveCaron}
159
160 % |H IS AN ENCODED \H
161 \def\DoubleAcuteCode{H}
162 \def\EncodedDoubleAcute{\EscapeChar\DoubleAcuteCode}
163 \def\DoubleAcute{\noexpand\SaveDoubleAcute}
164
165 % |c IS AN ENCODED \c
166 \def\CedillaCode{c}
167 \def\EncodedCedilla{\EscapeChar\CedillaCode}
168 \def\Cedilla{\noexpand\SaveCedilla}
169
170 % |d IS AN ENCODED \d
171 \def\UnderdotCode{d}
172 \def\EncodedUnderdot{\EscapeChar\UnderdotCode}
173 \def\Underdot{.}% CANNOT \edef \d
174
```

```
175 % |t IS AN ENCODED \t
176 \def\ArchJoinCode{t}
177 \def\EncodedArchJoin{\EscapeChar\ArchJoinCode}
178 \def\ArchJoin{.}% CANNOT \edef \t
179
180 % |b IS AN ENCODED \b
181 \def\LineUnderCode{b}
182 \def\EncodedLineUnder{\EscapeChar\LineUnderCode}
183 \def\LineUnder{.}% CANNOT \edef \b
184
185 % |C IS AN ENCODED \copyright
186 \def\CopyrightCode{C}
187 \def\EncodedCopyright{\EscapeChar\CopyrightCode}
188 \def\Copyright{.}% CANNOT \edef \copyright
189
190 % |p IS AN ENCODED \pounds
191 \def\PoundsCode{p}
192 \def\EncodedPounds{\EscapeChar\PoundsCode}
193 \def\Pounds{\SavePounds}
194
195 % |[ IS AN ENCODED {
196 \def\LBCode{[}
197 \def\EncodedLB{\EscapeChar\LBCode}
198 \def\UnencodedLB{.}
199 \def\LB{\EncodedLB}
200
201 % |] IS AN ENCODED }
202 \def\RBCode{]}
203 \def\EncodedRB{\EscapeChar\RBCode}
204 \def\UnencodedRB{.}
205 \def\RB{\EncodedRB}
206
207 % |z IS AN ENCODED \dag
208 \def\DaggerCode{z}
209 \def\EncodedDagger{\EscapeChar\DaggerCode}
210 \def\Dagger{.}% CANNOT \edef \dag
211
212 % |Z IS AN ENCODED \ddag
213 \def\DoubleDaggerCode{Z}
214 \def\EncodedDoubleDagger{\EscapeChar\DoubleDaggerCode}
215 \def\DoubleDagger{.}% CANNOT \edef \ddag
216
217 % |S IS AN ENCODED \S
218 \def\SectionSymbolCode{S}
219 \def\EncodedSectionSymbol{\EscapeChar\SectionSymbolCode}
220 \def\SectionSymbol{\SaveSectionSymbol}
221
222 % |P IS AN ENCODED \P
223 \def\PilcrowCode{P}
224 \def\EncodedPilcrow{\EscapeChar\PilcrowCode}
```

```
225 \def\Pilcrow{.}% CANNOT \edef \P
226
227 % |E IS AN ENCODED \AE
228 \def\AEscCode{E}
229 \def\EncodedAEsc{\EscapeChar\AEscCode}
230 \def\AEsc{\SaveAEsc}
231
232 % |e IS AN ENCODED \ae
233 \def\aescCode{e}
234 \def\Encodedaesc{\EscapeChar\aescCode}
235 \def\aesc{\Saveaesc}
236
237 % |O IS AN ENCODED \OE
238 \def\OEthelCode{O}
239 \def\EncodedOEthel{\EscapeChar\OEthelCode}
240 \def\OEthel{\SaveOEthel}
241
242 % |o IS AN ENCODED \oe
243 \def\oethelCode{o}
244 \def\Encodedoethel{\EscapeChar\oethelCode}
245 \def\oethel{\Saveoethel}
246
247 % |A IS AN ENCODED \AA
248 \def\AngstromCode{A}
249 \def\EncodedAngstrom{\EscapeChar\AngstromCode}
250 \def\Angstrom{\SaveAngstrom}
251
252 % |a IS AN ENCODED \aa
253 \def\angstromCode{a}
254 \def\Encodedangstrom{\EscapeChar\angstromCode}
255 \def\angstrom{\Saveangstrom}
256
257 % |Q IS AN ENCODED \O
258 \def\SlashedOCode{Q}
259 \def\EncodedSlashedO{\EscapeChar\SlashedOCode}
260 \def\SlashedO{\SaveSlashedO}
261
262 % |q IS AN ENCODED \o
263 \def\SlashedoCode{q}
264 \def\EncodedSlashedo{\EscapeChar\SlashedoCode}
265 \def\Slashedo{\SaveSlashedo}
266
267 % |L IS AN ENCODED \L
268 \def\BarredLCode{L}
269 \def\EncodedBarredL{\EscapeChar\BarredLCode}
270 \def\BarredL{\SaveBarredL}
271
272 % |l IS AN ENCODED \l
273 \def\BarredlCode{l}
274 \def\EncodedBarredl{\EscapeChar\BarredlCode}
```

```
275 \def\Barredl{\SaveBarredl}
276
277 % |s IS AN ENCODED \ss
278 \def\EszettCode{s}
279 \def\EncodedEszett{\EscapeChar\EszettCode}
280 \def\Eszett{\SaveEszett}
281
282 \newcounter{@letterindex}
283 \newcounter{@@letterindex}
284 \newcounter{@@@letterindex}
285 \newcounter{@wordindex}
286 \newcounter{@iargc}
287 \newcounter{@gobblesize}
288 \newcounter{@maxrotation}
289 \newcounter{@stringsize}
290 \newcounter{@@stringsize}
291 \newcounter{@@@stringsize}
292 \newcounter{@revisedstringsize}
293 \newcounter{@gobbleindex}
294 \newcounter{@charsfound}
295 \newcounter{@alph}
296 \newcounter{@alphaindex}
297 \newcounter{@capstrigger}
298 \newcounter{@fromindex}
299 \newcounter{@toindex}
300 \newcounter{@previousindex}
301 \newcounter{@flag}
302 \newcounter{@matchloc}
303 \newcounter{@matchend}
304 \newcounter{@matchsize}
305 \newcounter{@matchmax}
306 \newcounter{@skipped}
307 \newcounter{@lcwords}

308 %%%% CONFIGURATION COMMANDS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

\defaultTreatments   This command can be used to restore the default string treatments, prior to calling
\substring. The default treatments leave all symbol types intact and unaltered.

```
309 \newcommand\defaultTreatments{%
310   \def\EncodingTreatment{v}% <--Set=v to decode special chars (vs. q,e)
311   \def\AlphaCapsTreatment{1}% <--Set=1 to retain uppercase (vs. 0,2)
312   \def\AlphaTreatment{1}% <--Set=1 to retain lowercase (vs. 0,2)
313   \def\PunctuationTreatment{1}% <--Set=1 to retain punctuation (vs. 0)
314   \def\NumeralTreatment{1}% <--Set=1 to retain numerals (vs. 0)
315   \def\SymbolTreatment{1}% <--Set=1 to retain special chars (vs. 0)
316   \def\BlankTreatment{1}% <--Set=1 to retain blanks (vs. 0)
317   \def\CapitalizeString{0}%     <--Set=0 for no special action (vs. 1,2)
318   \def\SeekBlankSpace{0}%       <--Set=0 for no special action (vs. 1,2)
319 }
```

```
320 \defaultTreatments
```

\Treatments This command allows the user to specify the desired character class treatments, prior to a call to **\substring**. Unfortunately for the user, I have specified which character class each symbol belongs to. Therefore, it is not easy if the user decides that he wants a cedilla, for example, to be treated like an alphabetic character rather than a symbol.

```
321 % QUICK WAY TO SET UP TREATMENTS BY WHICH \@rotate HANDLES VARIOUS
322 % CHARACTERS
323 \newcommand\Treatments[6]{%
324   \def\AlphaCapsTreatment{#1}%  <--Set=0 to remove uppercase
325 %                                    =1 to retain uppercase
326 %                                    =2 to change UC to lc
327   \def\AlphaTreatment{#2}%     <--Set=0 to remove lowercase
328 %                                    =1 to retain lowercase
329 %                                    =2 to change lc to UC
330   \def\PunctuationTreatment{#3}%<--Set=0 to remove punctuation
331 %                                    =1 to retain punctuation
332   \def\NumeralTreatment{#4}%   <--Set=0 to remove numerals
333 %                                    =1 to retain numerals
334   \def\SymbolTreatment{#5}%    <--Set=0 to remove special chars
335 %                                    =1 to retain special chars
336   \def\BlankTreatment{#6}%     <--Set=0 to remove blanks
337 %                                    =1 to retain blanks
338 }
```

\+ This command (\+) is used to enact the stringstrings encoding. Key symbols are redefined, and any **\edef** which occurs while this command is active will adopt these new definitions.

```
339 % REENCODE MULTIBYTE SYMBOLS USING THE stringstrings ENCODING METHOD
340 \newcommand\+{%
341   \def\${\EncodedDollar}%
342   \def~{\EncodedBlankSpace}%
343   \def\^{\EncodedCircumflex}%
344   \def\~{\EncodedTilde}%
345   \def\"{\EncodedUmlaut}%
346   \def\'{\EncodedGrave}%
347   \def\'{\EncodedAcute}%
348   \def\={\EncodedMacron}%
349   \def\.{\EncodedOverdot}%
350   \def\u{\EncodedBreve}%
351   \def\v{\EncodedCaron}%
352   \def\H{\EncodedDoubleAcute}%
353   \def\c{\EncodedCedilla}%
354   \def\d{\EncodedUnderdot}%
355   \def\t{\EncodedArchJoin}%
356   \def\b{\EncodedLineUnder}%
357   \def\copyright{\EncodedCopyright}%
```

```
358    \def\pounds{\EncodedPounds}%
359    \def\{{\EncodedLeftBrace}%
360    \def\}{\EncodedRightBrace}%
361    \def\_{\EncodedUnderscore}%
362    \def\dag{\EncodedDagger}%
363    \def\ddag{\EncodedDoubleDagger}%
364    \def\S{\EncodedSectionSymbol}%
365    \def\P{\EncodedPilcrow}%
366    \def\AE{\EncodedAEsc}%
367    \def\ae{\Encodedaesc}%
368    \def\OE{\EncodedOEthel}%
369    \def\oe{\Encodedoethel}%
370    \def\AA{\EncodedAngstrom}%
371    \def\aa{\Encodedangstrom}%
372    \def\O{\EncodedSlashedO}%
373    \def\o{\EncodedSlashedo}%
374    \def\L{\EncodedBarredL}%
375    \def\l{\EncodedBarredl}%
376    \def\ss{\EncodedEszett}%
377 }
```

\? The command \? reverts the character encodings back to the standard LaTeX definitions. The command effectively undoes a previously enacted \+.

```
378 % WHEN TASK IS DONE, REVERT ENCODING TO STANDARD ENCODING METHOD
379 \newcommand\?{%
380    \let\$\SaveDollar%
381    \let~\SaveHardspace%
382    \let\^\SaveCircumflex%
383    \let\~\SaveTilde%
384    \let\"\SaveUmlaut%
385    \let\`\SaveGrave%
386    \let\'\SaveAcute%
387    \let\=\SaveMacron%
388    \let\.\SaveOverdot%
389    \let\u\SaveBreve%
390    \let\v\SaveCaron%
391    \let\H\SaveDoubleAcute%
392    \let\c\SaveCedilla%
393    \let\d\SaveUnderdot%
394    \let\t\SaveArchJoin%
395    \let\b\SaveLineUnder%
396    \let\copyright\SaveCopyright%
397    \let\pounds\SavePounds%
398    \let\{\SaveLeftBrace%
399    \let\}\SaveRightBrace%
400    \let\_\SaveUnderscore%
401    \let\dag\SaveDagger%
402    \let\ddag\SaveDoubleDagger%
403    \let\S\SaveSectionSymbol%
```

```
404    \let\P\SavePilcrow%
405    \let\AE\SaveAEsc%
406    \let\ae\Saveaesc%
407    \let\OE\SaveOEthel%
408    \let\oe\Saveoethel%
409    \let\AA\SaveAngstrom%
410    \let\aa\Saveangstrom%
411    \let\O\SaveSlashedO%
412    \let\o\SaveSlashedo%
413    \let\L\SaveBarredL%
414    \let\l\SaveBarredl%
415    \let\ss\SaveEszett%
416 }
```

\encodetoken   The command \encodetoken assigns the supplied token to one of three reserved
stringstrings user variables (the optional argument dictates which user variable).
Once encoded, the supplied token cannot be used in the normal way, but only in
stringstrings routines, unless and until it is decoded.

```
417 \newcommand\encodetoken[2][1]{%
418    \if 1#1%
419    \let\Uvari#2%
420    \let#2\uvari\else
421      \if 2#1%
422      \let\Uvarii#2%
423      \let#2\uvarii\else
424        \if 3#1%
425          \let\Uvariii#2%
426          \let#2\uvariii%
427        \fi
428      \fi
429    \fi
430 }
```

\decodetoken   The command \decodetoken deassigns the supplied token from the reserved
stringstrings user variables (the optional argument dictates which user variable),
so that the token may be used in the normal way again.

```
431 \newcommand\decodetoken[2][1]{%
432    \if 1#1%
433      \let#2\Uvari%
434      \def\Uvari{Uvari}\else
435      \if 2#1%
436        \let#2\Uvarii%
437        \def\Uvarii{Uvarii}\else
438        \if 3#1%
439          \let#2\Uvariii%
440          \def\Uvariii{Uvariii}%
441        \fi
442      \fi
```

27

```
443    \fi
444 }
```

```
445 %%%% COMMANDS TO MANIPULATE STRINGS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In the next group of commands, the result is always stored in an expandable string, \thestring. Expandable means that \thestring can be put into a subsequent \edef{} command. Additionally, the optional first argument can be used to cause three actions (verbose, encoded, or quiet):

=v \thestring is decoded (final result); print it immediately (default)

=e \thestring is encoded (intermediate result); don't print it

=q \thestring is decoded (final result), but don't print it

\substring   The command \substring is the brains of this package... It is used to acquire a substring from a given string, along with performing specified character manipulations along the way. Its strategy is fundamental to the stringstrings package: sequentially rotate the 1st character of the string to the end of the string, until the desired substring resides at end of rotated string. Then, gobble up the leading part of string until only the desired substring is left.

```
446 \newcommand\substring[4][v]{\+%
```

Obtain the string length of the string to be manipulated and store it in @stringsize.

```
447    \@getstringlength{#2}{@stringsize}%
```

First, \@decodepointer is used to convert indirect references like $ and $-3 into integers.

```
448    \@decodepointer{#3}%
449    \setcounter{@fromindex}{\@fromtoindex}%
450    \@decodepointer{#4}%
451    \setcounter{@toindex}{\@fromtoindex}%
```

Determine the number of characters to rotate to the end of the string and the number of characters to then gobble from it, in order to leave the desired substring.

```
452    \setcounter{@gobblesize}{\value{@stringsize}}%
453    \ifthenelse{\value{@toindex} > \value{@stringsize}}%
454      {\setcounter{@maxrotation}{\value{@stringsize}}}%
455      {\setcounter{@maxrotation}{\value{@toindex}}}%
456    \addtocounter{@gobblesize}{-\value{@maxrotation}}%
457    \addtocounter{@gobblesize}{\value{@fromindex}}%
458    \addtocounter{@gobblesize}{-1}%
```

Prepare for the string rotation by initializing counters, setting the targeted string into the working variable, \rotatingword, and set the encoding treatment specified.

```
459    \setcounter{@letterindex}{0}%
460    \edef\rotatingword{#2}%
461    \def\EncodingTreatment{#1}%
```

If capitalization (first character of string or of each word) was specified, the trigger for 1st-character capitalization will be set. However, the treatments for the alphabetic characters for the remainder of the string must be saved and reinstituted after the first character is capitalized.

```
462    \if 0\CapitalizeString%
463 %   DO NOT SET CAPITALIZE TRIGGER FOR FIRST CHARACTER
464      \setcounter{@capstrigger}{0}%
465    \else
466 %   SAVE CERTAIN TREATMENTS FOR LATER RESTORATION
467      \let\SaveAlphaTreatment\AlphaTreatment%
468      \let\SaveAlphaCapsTreatment\AlphaCapsTreatment%
469 %   SET CAPITALIZE TRIGGER FOR FIRST CHARACTER
470      \setcounter{@capstrigger}{1}%
471      \@forcecapson%
472    \fi
```

The command \@defineactions looks at the defined treatments and specifies how each of the stringstrings encoded characters should be handled (*i.e.*, left alone, removed, modified, *etc.*).

```
473 \@defineactions%
```

Here begins the primary loop of \substring in which characters of \rotatingword are successively moved (and possibly manipulated) from the first character of the string to the last. @letterindex is the running index defining how many characters have been operated on.

```
474    \whiledo{\value{@letterindex} < \value{@maxrotation}}{%
475      \addtocounter{@letterindex}{1}%
```

When \CapitalizeString equals 1, only the first character of the string is capitalized. When it equals 2, every word in the string is capitalized. When equal to 2, this bit of code looks for the blankspace that follows the end of a word, and uses it to reset the capitalization trigger for the next non-blank character.

```
476 %   IF NEXT CHARACTER BLANK WHILE \CapitalizeString=2,
477 %   SET OR KEEP ALIVE TRIGGER.
478      \if 2\CapitalizeString%
479        \isnextbyte[q]{\EncodedBlankSpace}{\rotatingword}%
480        \if F\theresult\isnextbyte[q]{\BlankSpace}{\rotatingword}\fi%
481        \if T\theresult%
482          \if 0\arabic{@capstrigger}%
483            \@forcecapson%
484            \@defineactions%
485          \fi
486          \setcounter{@capstrigger}{2}%
```

29

```
487      \fi
488    \fi
```

Is the next character an encoded symbol? If it is a normal character, simply rotate it to the end of the string. If it is an encoded symbol however, its treatment will depend on whether it will be gobbled away or end up in the final substring. If it will be gobbled away, leave it encoded, because the gobbling routine knows how to gobble encoded characters. If it will end up in the substring, manipulate it according to the encoding rules set in **\@defineactions** and rotate it.

```
489 %    CHECK IF NEXT CHARACTER IS A SYMBOL
490      \isnextbyte[q]{\EscapeChar}{\rotatingword}%
491      \ifthenelse{\value{@letterindex} < \value{@fromindex}}%
492        {%
493 %      THIS CHARACTER WILL EVENTUALLY BE GOBBLED
494        \if T\theresult%
495 %        ROTATE THE ESCAPE CHARACTER, WHICH WILL LEAVE THE SYMBOL ENCODED
496 %        FOR PROPER GOBBLING (ESCAPE CHARACTER DOESN'T COUNT AS A LETTER)
497        \edef\rotatingword{\@rotate{\rotatingword}}%
498        \addtocounter{@letterindex}{-1}%
499        \else
500 %        NORMAL CHARACTER OR SYMBOL CODE... ROTATE IT
501        \edef\rotatingword{\@rotate{\rotatingword}}%
502      \fi
503      }%
504        {%
505 %      THIS CHARACTER WILL EVENTUALLY MAKE IT INTO SUBSTRING
506        \if T\theresult%
507 %        ROTATE THE SYMBOL USING DEFINED TREATMENT RULES
508        \edef\rotatingword{\ESCrotate{\expandafter\@gobble\rotatingword}}%
509        \else
510 %        NORMAL CHARACTER... ROTATE IT
511        \edef\rotatingword{\@rotate{\rotatingword}}%
512      \fi
513      }%
```

Here, the capitalization trigger persistently tries to turn itself off with each loop through the string rotation. Only if the earlier code found the rotation to be pointing to the blank character(s) between words while **\CapitalizeString** equals 2 will the trigger be prevented from extinguishing itself.

```
514 %    DECREMENT CAPITALIZATION TRIGGER TOWARDS 0, EVERY TIME THROUGH LOOP
515      \if 0\arabic{@capstrigger}%
516      \else
517        \addtocounter{@capstrigger}{-1}%
518        \if 0\arabic{@capstrigger}\@relaxcapson\fi
519      \fi
```

In addition to the standard **\substring** calls in which fixed substring limits are specified (which in turn fixes the number of character rotations to be executed), some stringstrings commands want the rotations to continue until a blankspace

is located. This bit of code looks for that blank space, if that was the option requested. Once found, the rotation will stop. However, depending on the value of \SeekBlankSpace, the remainder of the string may either be retained or discarded.

```
520 %    IF SOUGHT SPACE IS FOUND, END ROTATION OF STRING
521    \if 0\SeekBlankSpace\else
522      \isnextbyte[q]{\EncodedBlankSpace}{\rotatingword}%
523      \if F\theresult\isnextbyte[q]{\BlankSpace}{\rotatingword}\fi%
524      \if T\theresult%
525        \if 1\SeekBlankSpace%
526 %        STOP ROTATION, KEEP REMAINDER OF STRING
527          \setcounter{@maxrotation}{\value{@letterindex}}%
528        \else
529 %        STOP ROTATION, THROW AWAY REMAINDER OF STRING
530          \addtocounter{@gobblesize}{\value{@maxrotation}}%
531          \setcounter{@maxrotation}{\value{@letterindex}}%
532          \addtocounter{@gobblesize}{-\value{@maxrotation}}%
533        \fi
534      \fi
535    \fi
536  }%
```

The loop has ended.
Gobble up the first @gobblesize characters (not bytes!) of the string, which should leave the desired substring as the remainder. If the mode is verbose, print out the resulting substring.

```
537 % GOBBLE AWAY THAT PART OF STRING THAT ISN'T PART OF REQUESTED SUBSTRING
538    \@gobblearg{\rotatingword}{\arabic{@gobblesize}}%
539    \edef\thestring{\gobbledword}%
540    \if v#1\thestring\fi%
541 \?}
```

Many of the following commands are self-expanatory. The recipe they follow is to use \Treatments to specify how different character classes are to be manipulated, and then to call upon \substring to effect the desired manipulation. Treatments are typically re-defaulted at the conclusion of the command, which is why the user, if desiring special treatments, should specify those treatments immediately before a call to \substring.

\caseupper

```
542 % Convert Lower to Uppercase; retain all symbols, numerals,
543 % punctuation, and blanks.
544 \newcommand\caseupper[2][v]{%
545    \Treatments{1}{2}{1}{1}{1}{1}%
546    \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
547    \defaultTreatments%
548 }
```

`\caselower`

```
549 % Convert Upper to Lowercase; retain all symbols, numerals,
550 % punctuation, and blanks.
551 \newcommand\caselower[2][v]{%
552   \Treatments{2}{1}{1}{1}{1}{1}%
553   \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
554   \defaultTreatments%
555 }
```

`\solelyuppercase`

```
556 % Convert Lower to Uppercase; discard symbols, numerals, and
557 % punctuation, but keep blanks.
558 \newcommand\solelyuppercase[2][v]{%
559   \Treatments{1}{2}{0}{0}{0}{1}%
560   \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
561   \defaultTreatments%
562 }
```

`\solelylowercase`

```
563 % Convert Upper to Lowercase; discard symbols, numerals, and
564 % punctuation, but keep blanks.
565 \newcommand\solelylowercase[2][v]{%
566   \Treatments{2}{1}{0}{0}{0}{1}%
567   \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
568   \defaultTreatments%
569 }
```

`\changecase`

```
570 % Convert Lower to Uppercase & Upper to Lower; retain all symbols, numerals,
571 % punctuation, and blanks.
572 \newcommand\changecase[2][v]{%
573   \Treatments{2}{2}{1}{1}{1}{1}%
574   \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
575   \defaultTreatments%
576 }
```

`\noblanks`

```
577 % Remove blanks; retain all else.
578 \newcommand\noblanks[2][v]{%
579   \Treatments{1}{1}{1}{1}{1}{0}%
580   \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
581   \defaultTreatments%
582 }
```

`\nosymbolsnumerals`

```
583 % Retain case; discard symbols & numerals; retain
584 % punctuation & blanks.
585 \newcommand\nosymbolsnumerals[2][v]{%
586     \Treatments{1}{1}{1}{0}{0}{1}%
587     \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
588     \defaultTreatments%
589 }
```

\alphabetic

```
590 % Retain case; discard symbols, numerals &
591 % punctuation; retain blanks.
592 \newcommand\alphabetic[2][v]{%
593     \Treatments{1}{1}{0}{0}{0}{1}%
594     \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
595     \defaultTreatments%
596 }
```

\capitalize   The command \CapitalizeString is not set by \Treatments, but only in
\capitalize or in \capitalizewords.

```
597 % Capitalize first character of string,
598 \newcommand\capitalize[2][v]{%
599     \defaultTreatments%
600     \def\CapitalizeString{1}%
601     \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
602     \def\CapitalizeString{0}%
603 }
```

\capitalizewords

```
604 % Capitalize first character of each word in string,
605 \newcommand\capitalizewords[2][v]{%
606     \defaultTreatments%
607     \def\CapitalizeString{2}%
608     \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
609     \def\CapitalizeString{0}%
610 }
```

\reversestring   Reverses a string from back to front. To do this, a loop is set up in which characters
are grabbed one at a time from the end of the given string, working towards
the beginning of the string. The grabbed characters are concatenated onto the
end of the working string, \@reversedstring. By the time the loop is complete
\@reversedstring fully represents the reversed string. The result is placed into
\thestring.

```
611 % REVERSES SEQUENCE OF CHARACTERS IN STRING
612 \newcommand\reversestring[2][v]{%
613     \def\@reversedstring{}%
614     \+\@getstringlength{#2}{@@stringsize}\?%
```

33

```
615    \setcounter{@@@letterindex}{\the@@stringsize}%
616    \whiledo{\the@@@letterindex > 0}{%
617      \if e#1%
618        \substring[e]{#2}{\the@@@letterindex}{\the@@@letterindex}%
619      \else
620        \substring[q]{#2}{\the@@@letterindex}{\the@@@letterindex}%
621      \fi
622      \edef\@reversedstring{\@reversedstring\thestring}%
623      \addtocounter{@@@letterindex}{-1}%
624    }%
625    \edef\thestring{\@reversedstring}%
626    \if v#1\thestring\fi%
627 }
```

\convertchar Takes a string, and replaces each occurance of a specified character with a replacement string. The only complexity in the logic is that a separate replacement algorithm exists depending on whether the specified character to be replaced is a normal character or an encoded character.

```
628 % TAKES A STARTING STRING #2 AND SUBSTITUTES A SPECIFIED STRING #4
629 % FOR EVERY OCCURANCE OF A PARTICULAR GIVEN CHARACTER #3.  THE
630 % CHARACTER TO BE CONVERTED MAY BE EITHER A PLAIN CHARACTER OR
631 % AN ENCODABLE SYMBOL.
632 \newcommand\convertchar[4][v]{%
633    \+%
634    \edef\encodedstring{#2}%
635    \edef\encodedfromarg{#3}%
636    \edef\encodedtoarg{#4}%
637    \?%
638    \isnextbyte[q]{\EscapeChar}{\encodedfromarg}%
639    \if F\theresult%
640 %   PLAIN "FROM" ARGUMENT
641      \@convertbytetostring[#1]{\encodedstring}{#3}{\encodedtoarg}%
642    \else
643 %   ENCODABLE "FROM" ARGUMENT
644      \@convertsymboltostring[#1]{\encodedstring}%
645        {\expandafter\@gobble\encodedfromarg}{\encodedtoarg}%
646    \fi
647 }
```

\convertword Takes a string, a replaces each occurance of a specified string with a replacement string.

```
648 \newcounter{@@matchloc}
649 % LIKE \convertchar, EXCEPT FOR WORDS
650 \newcommand\convertword[4][v]{%
651    \+\edef\@@teststring{#2}%
652    \edef\@fromstring{#3}%
653    \edef\@tostring{#4}\?%
654    \edef\@@@teststring{\@@teststring}%
```

```
655    \def\@buildfront{}%
656    \edef\@buildstring{\@@teststring}%
657    \setcounter{@charsfound}{0}%
658    \whiledo{\the@charsfound > -1}{%
```

Seek occurance of \@fromstring in larger \@@teststring

```
659        \whereisword[q]{\@@teststring}{\@fromstring}%
660        \setcounter{@matchloc}{\theresult}%
661        \ifthenelse{\the@matchloc = 0}%
662          {%
```

Not found. Done.

```
663          \setcounter{@charsfound}{-1}%
664        }%
665        {%
```

Potential matchstring.

```
666        \addtocounter{@charsfound}{1}%
```

Grab current test string from beginning to point just prior to potential match.

```
667        \addtocounter{@matchloc}{-1}%
668        \substring[e]{\@@@teststring}{1}{\the@matchloc}%
```

The string \@buildfront is the total original string, with string substitutions, from character 1 to current potential match.

```
669        \edef\@buildfront{\@buildfront\thestring}%
```

See if potential matchstring takes us to end-of-string...

```
670        \addtocounter{@matchloc}{1}%
671        \addtocounter{@matchloc}{\the@matchsize}%
672        \ifthenelse{\the@matchloc > \the@@@stringsize}%
673          {%
```

...if so, then match is last one in string. Tack on replacement string to \@buildfront to create final string. Exit.

```
674          \setcounter{@charsfound}{-1}%
675          \edef\@buildstring{\@buildfront\@tostring}%
676        }%
677        {%
```

...if not, redefine current teststring to begin at point following the current substitution. Make substitutions to current \@buildstring and \@buildfront. Loop through logic again on new teststring.

```
678        \substring[e]{\@@@teststring}{\the@matchloc}{\@MAXSTRINGSIZE}%
679        \edef\@@teststring{\thestring}%
680        \edef\@@@teststring{\@@teststring}%
```

```
681          \edef\@buildstring{\@buildfront\@tostring\@@@teststring}%
682          \edef\@buildfront{\@buildfront\@tostring}%
683        }%
684      }%
685    }%
686    \substring[#1]{\@buildstring}{1}{\@MAXSTRINGSIZE}%
687 }
```

\resetlcwords   Removes all words from designated "lower-case words" list. This can be useful
because large lists of lower-case words can significantly slow-down the function of
\capitalizetitle.

```
688 \setcounter{@lcwords}{0}
689 % RESET LOWER-CASE WORD COUNT; START OVER
690 \newcommand\resetlcwords[0]{%
691    \setcounter{@lcwords}{0}%
692 }
```

\addlcwords   Add words to the list of designated "lower-case words" which will not be capitalized
by \capitalizetitle. The input should consist of space-separated words, which
are, in turn, passed on to \addlcword.

```
693 % PROVIDE LIST OF SPACE-SEPARATED WORDS TO REMAIN LOWERCASE IN TITLES
694 \newcommand\addlcwords[1]{%
695    \getargs{#1}%
696    \setcounter{@wordindex}{0}%
697    \whiledo{\value{@wordindex} < \narg}{%
698      \addtocounter{@wordindex}{1}%
699      \addlcword{\csname arg\roman{@wordindex}\endcsname}%
700    }
701 }
```

\addlcword   Add a word to the list of designated "lower-case words" which will not be capital-
ized by \capitalizetitle.

```
702 % PROVIDE A SINGLE WORD TO REMAIN LOWERCASE IN TITLES
703 \newcommand\addlcword[1]{%
704    \addtocounter{@lcwords}{1}%
705    \expandafter\edef\csname lcword\roman{@lcwords}\endcsname{#1}
706 }
```

\capitalizetitle   Makes every word of a multi-word input string capitalized, except for specifi-
cally noted "lower-case words" (examples might include prepositions, conjunc-
tions, *etc.*). The first word of the input string is capitalized, while lower-case
words, previously designated with \addlcword and \addlcwords, are left in lower
case.

```
707 % CAPITALIZE TITLE, EXCEPT FOR DESIGNATED "LOWER-CASE" WORDS
708 \newcommand\capitalizetitle[2][v]{%
```

```
709 % First, capitalize every word (save in encoded form, not printed)
710   \capitalizewords[e]{#2}%
711 % Then lowercase words that shouldn't be capitalized, like articles,
712 % prepositions, etc. (save in encoded form, not printed)
713   \setcounter{@wordindex}{0}%
714   \whiledo{\value{@wordindex} < \value{@lcwords}}{%
715     \addtocounter{@wordindex}{1}%
716     \edef\mystring{\thestring}%
717     \edef\lcword{\csname lcword\roman{@wordindex}\endcsname}%
718     \capitalize[e]{\lcword}%
719     \edef\ucword{\thestring}%
720     \convertword[e]{\mystring}{\ucword~}{\lcword~}%
721   }
722 % Finally, recapitalize the first word of the Title, and print it.
723   \capitalize[#1]{\thestring}%
724 }
```

\rotateword   Moves first word of given string **#2** to end of string, including leading and trailing blank spaces.

```
725 \newcommand\rotateword[2][v]{%
726   \+\edef\thestring{#2}\?%
```

Rotate leading blank spaces to end of string

```
727   \@treatleadingspaces[e]{\thestring}{}%
```

Define end-of-rotate condition for **\substring** as next blank space

```
728   \def\SeekBlankSpace{1}%
```

Leave rotated characters alone

```
729   \Treatments{1}{1}{1}{1}{1}{1}%
```

Rotate to the next blank space or the end of string, whichever comes first.

```
730   \substring[e]{\thestring}{1}{\@MAXSTRINGSIZE}%
```

Rotate trailing spaces.

```
731   \@treatleadingspaces[#1]{\thestring}{}%
732   \defaultTreatments%
733 }
```

\removeword   Remove the first word of given string **#2**, including leading and trailing spaces. Note that logic is identical to **\rotateword**, except that affected spaces and characters are removed instead of being rotated.

```
734 \newcommand\removeword[2][v]{%
735   \+\edef\thestring{#2}\?%
```

The **{x}** final argument indicates to delete leading spaces.

```
736    \@treatleadingspaces[e]{\thestring}{x}%
737    \def\SeekBlankSpace{1}%
```

The Treatments are specified to remove all characters.

```
738    \Treatments{0}{0}{0}{0}{0}{0}%
739    \substring[e]{\thestring}{1}{\@MAXSTRINGSIZE}%
```

Trailing spaces are also deleted.

```
740    \@treatleadingspaces[#1]{\thestring}{x}%
741    \defaultTreatments%
742 }
```

\getnextword   A special case of \getaword, where word-to-get is specified as "1".

```
743 % GETS NEXT WORD FROM STRING #2.
744 % NOTE: ROTATES BACK TO BEGINNING, AFTER STRING OTHERWISE EXHAUSTED
745 \newcommand\getnextword[2][v]{%
746    \getaword[#1]{#2}{1}%
747 }
```

\getaword   Obtain a specified word number (#3) from string #2. Logic: rotate leading spaces
to end of string; then loop #3 – 1 times through \rotateword. Finally, get next
word.

```
748 % GETS WORD #3 FROM STRING #2.
749 % NOTE: ROTATES BACK TO BEGINNING, AFTER STRING OTHERWISE EXHAUSTED
750 \newcommand\getaword[3][v]{%
751    \setcounter{@wordindex}{1}%
752    \+\edef\thestring{#2}\?%
753    \@treatleadingspaces[e]{\thestring}{}%
754    \whiledo{\value{@wordindex} < #3}{%
755       \rotateword[e]{\thestring}%
756       \addtocounter{@wordindex}{1}%
757    }%
758    \@getnextword[#1]{\thestring}%
759 }
```

\rotateleadingspaces   Rotate leading spaces of string #2 to the end of string.

```
760 \newcommand\rotateleadingspaces[2][v]{%
761    \@treatleadingspaces[#1]{#2}{}%
762 }
```

\removeleadingspaces   Remove leading spaces from string #2.

```
763 \newcommand\removeleadingspaces[2][v]{%
764    \@treatleadingspaces[#1]{#2}{x}%
765 }
```

\stringencode

```
766 % ENCODE STRING; UNLIKE OTHER COMMANDS, DEFAULT IS NO PRINT
767 \newcommand\stringencode[2][e]{%
768   \defaultTreatments%
769   \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
770 }
```

\stringdecode

```
771 % DECODE STRING
772 \newcommand\stringdecode[2][v]{%
773   \defaultTreatments%
774   \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
775 }
```

\gobblechar   Remove first character (not byte!) from string #2. Unlike just about all other stringstrings commands, result is retokenized and not expanded.

```
776 % SINGLE-CHARACTER VERSION OF \gobblechars.  IN THIS CASE, TWO-BYTE
777 % ESCAPE SEQUENCES, WHEN ENCOUNTERED, COUNT AS A SINGLE GOBBLE.
778 \newcommand\gobblechar[2][q]{\+%
779   \@gobblearg{#2}{1}%
780   \?\retokenize[#1]{\gobbledword}%
781 }
```

\gobblechars   Remove first #3 characters (not bytes!) from string #2. Unlike just about all other stringstrings commands, result is retokenized and not expanded.

```
782 % USER CALLABLE VERSION OF \@gobblearg.  TURNS ON REENCODING.
783 % GOBBLE FIRST #3 CHARACTERS FROM STRING #2. TWO-BYTE
784 % ESCAPE SEQUENCES, WHEN ENCOUNTERED, COUNT AS A SINGLE GOBBLE.
785 \newcommand\gobblechars[3][q]{\+%
786   \@gobblearg{#2}{#3}%
787   \?\retokenize[#1]{\gobbledword}%
788 }
```

\retokenize   One of the key stringstrings routines that provides several indispensible functions. Its function is to take an encoded string #2 that has been given, and repopulate the string with its LaTeX tokens in a \def form (not an expanded \edef form). It is useful if required to operate on a string outside of the stringstrings library routines, following a stringstrings manipulation. It is also useful to display certain tokens which cannot be manipulated in expanded form. See Table 2 for a list of tokens that will only work when the resulting string is retokenized (and not expanded).

Logic: Loop through each character of given string #2. Each successive character of the string is retokenized as \inexttoken, \iinexttoken, \iiinexttoken, \ivnexttoken, *etc.*, respectively. Then a series of strings are formed as

```
\def\buildtoken{}
\def\buildtokeni{\builtoken\inexttoken}
\def\buildtokenii{\builtokeni\iinexttoken}
\def\buildtokeniii{\builtokenii\iiinexttoken}
\def\buildtokeniv{\builtokeniii\ivnexttoken}
```

The last in the sequence of `\builtoken...` strings (renamed `\buildtokenq`) is the retokenized version of string `#2`.

```
789 % CONVERTS ENCODED STRING BACK INTO TOKENIZED FORM (i.e., def'ED).
790 \newcommand\retokenize[2][q]{\+%
791   \edef\@svstring{#2}%
792   \edef\buildtoken{}%
793   \@getstringlength{#2}{@@stringsize}\?%
794   \setcounter{@@letterindex}{0}%
795   \whiledo{\the@@letterindex < \the@@stringsize}{%
796     \setcounter{@previousindex}{\the@@letterindex}%
797     \addtocounter{@@letterindex}{1}%
798     \substring[e]{\@svstring}{\the@@letterindex}{\the@@letterindex}%
799     \@retokenizechar{\thestring}{\roman{@@letterindex}nexttoken}%
800     \expandafter\def\csname buildtoken\roman{@@letterindex}%
801      \expandafter\endcsname\expandafter%
802       {\csname buildtoken\roman{@previousindex}\expandafter\endcsname%
803        \csname\roman{@@letterindex}nexttoken\endcsname}%
804   }%
805   \expandafter\def\expandafter\buildtokenq\expandafter{%
806     \csname buildtoken\roman{@@letterindex}\endcsname}%
807   \def\thestring{\buildtokenq}%
808   \if v#1\thestring\fi
809 }
```

```
810 % %%%% COMMANDS TO EXTRACT STRING INFORMATION %%%%%%%%%%%%%%%%%%%%%%%%
```

The following group of commands extract information about a string, and store the result in a string called `\theresult`. Since the result is not a substring, a *mode* of `[e]` carries no meaning. Only `[v]` and `[q]` modes apply here.

`\stringlength` Returns the length of the given string in *characters*, not *bytes*.

```
811 % USER CALLABLE VERSION of \@getstringlength:
812 % GET'S STRING LENGTH OF [#2], PUTS RESULT IN \theresult.  PRINTS RESULT
813 % UNLESS IN QUIET [q] MODE.
814 \newcommand\stringlength[2][v]{\+%
815   \@getstringlength{#2}{@@stringsize}%
816   \edef\theresult{\arabic{@@stringsize}}%
817   \if v#1\theresult\fi%
818 \?}
```

`\findchars` Find number of occurances of character `#3` in string `#2`.

```
819 % CHECKS TO SEE IF THE CHARACTER [#3] APPEARS ANYWHERE IN STRING [#2].
820 % THE NUMBER OF OCCURANCES IS PRINTED OUT, EXCEPT WHEN [#1]=q, QUIET
821 % MODE.  RESULT IS ALSO STORED IN \theresult .  TO FIND SPACES, ARG3
822 % SHOULD BE SET TO {~}, NOT { }.
823 \newcommand\findchars[3][v]{\+%
824     \@getstringlength{#2}{@@stringsize}%
825     \setcounter{@charsfound}{0}%
826     \setcounter{@@letterindex}{0}%
```

Loop through each character of #2.

```
827     \whiledo{\value{@@letterindex} < \value{@@stringsize}}{%
828         \addtocounter{@@letterindex}{1}%
```

Test if the `@@letterindex` character of string #2 equals #3. If so, add to tally.

```
829         \testmatchingchar{#2}{\arabic{@@letterindex}}{#3}%
830         \ifmatchingchar\addtocounter{@charsfound}{1}\fi
831     }%
832     \edef\theresult{\arabic{@charsfound}}%
833     \if q#1\else\theresult\fi%
834 \?}
```

\whereischar   Similar to \findchars, but instead finds first occurance of match character #3
               within #2 and returns its location within #2.

```
835 % CHECKS TO FIND LOCATION OF FIRST OCCURANCE OF [#3] IN STRING [#2].
836 % THE LOCATION IS PRINTED OUT, EXCEPT WHEN [#1]=q, QUIET
837 % MODE.  RESULT IS ALSO STORED IN \theresult .  TO FIND SPACES, ARG3
838 % SHOULD BE SET TO {~}, NOT { }.
839 \newcommand\whereischar[3][v]{\+%
840     \@getstringlength{#2}{@@stringsize}%
841     \edef\@theresult{0}%
842     \setcounter{@@letterindex}{0}%
```

Loop through characters of #2 sequentially.

```
843     \whiledo{\value{@@letterindex} < \value{@@stringsize}}{%
844         \addtocounter{@@letterindex}{1}%
```

Look for match. If found, save character-location index, and reset loop index to
break from loop.

```
845         \testmatchingchar{#2}{\arabic{@@letterindex}}{#3}%
846         \ifmatchingchar%
847             \edef\@theresult{\the@@letterindex}%
848             \setcounter{@@letterindex}{\the@@stringsize}%
849         \fi
850     }%
851     \edef\theresult{\@theresult}%
852     \if q#1\else\theresult\fi%
853 \?}
```

\whereisword   Finds location of specified word (#3) in string #2.

```
854 % LIKE \whereischar, EXCEPT FOR WORDS
855 \newcommand\whereisword[3][v]{\+%
856   \setcounter{@skipped}{0}%
```

\@@@@teststring initially contains #2. As false alarms are located, the string will be redefined to lop off initial characters of string.

```
857   \edef\@@@@teststring{#2}%
858   \edef\@matchstring{#3}%
859   \@getstringlength{#2}{@@stringsize}%
860   \setcounter{@@@stringsize}{\value{@@stringsize}}
861   \@getstringlength{#3}{@matchsize}%
862   \setcounter{@matchmax}{\the@@stringsize}%
863   \addtocounter{@matchmax}{-\the@matchsize}%
864   \addtocounter{@matchmax}{1}%
865   \setcounter{@flag}{0}%
```

Define **\matchchar** as the first character of the match string (#3).

```
866   \substring[e]{#3}{1}{1}%
867   \edef\matchchar{\thestring}%
868   \whiledo{\the@flag = 0}{%
```

Look for first character of match string within **\@@@@teststring**.

```
869     \whereischar[q]{\@@@@teststring}{\matchchar}%
870     \setcounter{@matchloc}{\theresult}%
871     \ifthenelse{\equal{0}{\value{@matchloc}}}%
```

If none found, we are done.

```
872     {%
873       \setcounter{@flag}{1}%
874     }%
```

If **\matchchar** is found, must determine if it is the beginning of the match string, or just an extraneous match (*i.e.*, false alarm). Extract substring of **\@@@@teststring**, of a size equal to the match string. Compare this extracted string with the match string.

```
875     {%
876       \setcounter{@matchend}{\theresult}%
877       \addtocounter{@matchend}{\value{@matchsize}}%
878       \addtocounter{@matchend}{-1}%
879       \substring[e]{\@@@@teststring}{\the@matchloc}{\the@matchend}%
880       \ifthenelse{\equal{\thestring}{\@matchstring}}%
```

Found a match! Save the match location

```
881       {%
882         \setcounter{@flag}{1}%
883         \addtocounter{@matchloc}{\the@skipped}%
```

```
884        \edef\theresult{\the@matchloc}%
885     }%
```

False alarm. Determine if lopping off the leading characters of `\@@@@teststring`
(to discard the false-alarm occurance) is feasible. If lopping would take one past
the end of the string, then no match is possible. If lopping permissible, rede-
fine the string `\@@@@teststring`, keeping track of the total number of lopped-off
characters in the counter `@skipped`.

```
886     {%
887       \addtocounter{@skipped}{\the@matchloc}%
888       \addtocounter{@matchloc}{1}%
889       \ifthenelse{\value{@matchloc} > \value{@matchmax}}%
890       {%
891         \setcounter{@flag}{1}%
892         \edef\theresult{0}%
893       }%
894       {%
895         \substring[e]{\@@@@teststring}{\the@matchloc}{\@MAXSTRINGSIZE}%
896         \edef\@@@@teststring{\thestring}%
897       }%
898     }%
899   }%
900   }%
901   \if q#1\else\theresult\fi%
902 \?}
```

`\findwords`   Finds the number of occurances of a word within the provided string

```
903 % LIKE \findchar, EXCEPT FOR WORDS
904 \newcommand\findwords[3][v]{%
905   \+\edef\@@teststring{#2}\?%
906   \edef\@@@teststring{\@@teststring}%
907   \setcounter{@charsfound}{0}%
908   \whiledo{\the@charsfound > -1}{%
```

Seek occurance of #3 in the string to be tested

```
909     \whereisword[q]{\@@teststring}{#3}%
910     \setcounter{@matchloc}{\theresult}%
911     \ifthenelse{\the@matchloc = 0}%
912       {%
```

None found. Break from loop.

```
913         \edef\theresult{\the@charsfound}%
914         \setcounter{@charsfound}{-1}%
915     }%
916       {%
```

Found. Increment count.

```
917        \addtocounter{@charsfound}{1}%
918        \addtocounter{@matchloc}{\the@matchsize}%
919        \ifthenelse{\the@matchloc > \the@@stringsize}%
920        {%
```

This "find" takes us to the end-of-string. Break from loop now.

```
921           \edef\theresult{\the@charsfound}%
922           \setcounter{@charsfound}{-1}%
923        }%
924        {%
```

More string to search. Lop off what has been searched from string to be tested, and re-loop for next search.

```
925           \substring[e]{\@@@teststring}{\the@matchloc}{\@MAXSTRINGSIZE}%
926           \edef\@@teststring{\thestring}%
927           \edef\@@@teststring{\@@teststring}%
928        }%
929      }%
930   }%
931   \if q#1\else\theresult\fi%
932 }
```

\wordcount   Counts words (space-separated text) in a string. Simply removes one word at a
             time, counting the words as it goes. With each removal, checks for non-zero string
             size remaining.

```
933 % WORD COUNT
934 \newcommand\wordcount[2][v]{\+%
935   \edef\@argv{#2}
936   \@getstringlength{\@argv}{@stringsize}
937   \setcounter{@iargc}{0}
938   \whiledo{\value{@stringsize} > 0}{%
939     \addtocounter{@iargc}{1}%
940     \removeword[e]{\@argv}%
941     \edef\@argv{\thestring}%
942     \@getstringlength{\@argv}{@stringsize}%
943   }
944   \edef\theresult{\arabic{@iargc}}%
945   \if v#1\theresult\fi%
946 \?}
```

\getargs   Parse a string of arguments in Unix-like manner. Define \argv as #2. Grabs
           leading word from \argv and puts it in \argi. Increment argument count; remove
           leading word from \argv. Repeat this process, with each new argument being
           placed in \argii, \argiii, \argiv, *etc.* Continue until size of \argv is exhausted.

```
947 % OBTAINS ARGUMENTS (WORDS) IN #1 ALA UNIX getarg COMMAND
948 % narg CONTAINS NUMBER OF ARGUMENTS.  ARGUMENTS CONTAINED IN
949 % argi, argii, argiii, argiv, ETC.
```

```
950 % v mode disabled
951 \newcommand\getargs[2][q]{\+%
952   \if v#1\def\@mode{q}\else\def\@mode{#1}\fi
953   \edef\@argv{#2}%
954   \@getstringlength{\@argv}{@stringsize}%
955   \setcounter{@iargc}{0}%
956   \whiledo{\value{@stringsize} > 0}{%
957     \addtocounter{@iargc}{1}%
958     \getaword[\@mode]{\@argv}{1}%
959     \expandafter\edef\csname arg\roman{@iargc}\endcsname{\thestring}%
960     \removeword[e]{\@argv}%
961     \edef\@argv{\thestring}%
962     \@getstringlength{\@argv}{@stringsize}%
963   }
964   \edef\narg{\arabic{@iargc}}%
965 \?}

966 %%%%% COMMANDS TO TEST STRINGS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The following group of commands test for various alphanumeric string conditions.

\isnextbyte   This routine performs a simple test to determine if the first byte of string #3 matches the byte given by #2. The only problem is that the test can produce a false negative if the first byte of the test string equals the match byte and the second byte of the test string equals the SignalChar (defined below).

   To resolve this possibility, the test is performed twice with two different values for \SignalChar, only one of which can produce a false negative for a given test string. If the two results match, then that result gives the correct answer to the question of whether the first byte of #3 equals #2. If, however, the two results fail to match, then one can assume that one of the results is a false negative, and so a "true" condition results.

The following two "signal characters," used for the two tests, can be any two distinct characters. They are used solely by \isnextbyte.

```
967 \def\PrimarySignalChar{@}
968 \def\SecondarySignalChar{'}
969
970 % \isnextbyte NEEDS TO OPERATE IN RAW (SINGLE BYTE) MODE SO AS TO
971 % PERFORM TESTS FOR PRESENCE OF \EscapeChar
```

Incidentally, \isnextbyte can and is used by stringstrings to detect multi-byte characters in a manner which may also be employed by the user. To do this: First, the string to be tested should be encoded. Then, \isnextbyte may be used to check for \EscapeChar which is how every multi-byte character will begin its encoding by the stringstrings package. If \EscapeChar is detected as the next character, then the string to test may have its leading byte gobbled and the next

character (called the Escape Code) may be tested, and compared against the known stringstrings escape codes. The combination of Escape-Character/Escape-Code is how all multi-byte characters are encoded by the stringstrings package.

```
972 \newcommand\isnextbyte[3][v]{%
```

Here's the first test...

```
973   \let\SignalChar\PrimarySignalChar%
974   \edef\@x{\if #2#3\else\SignalChar\fi}%
975   \edef\@x{\if \SignalChar\@x F\else T\fi}%
```

...and the second

```
976   \let\SignalChar\SecondarySignalChar%
977   \edef\@y{\if #2#3\else\SignalChar\fi}%
978   \edef\@y{\if \SignalChar\@y F\else T\fi}%
```

If the two tests produced the same result, then a comparison of `\@x\@y` and `\@y\@x` will show it.

```
979 % BECAUSE THE METHOD ONLY PRODUCES FALSE NEGATIVES, IF RESULTS DON'T
980 % AGREE FROM USING TWO DIFFERENT SIGNAL CHARACTERS, RESULT MUST BE TRUE.
981   \ifthenelse{\equal{\@x\@y}{\@y\@x}}
982     {\edef\theresult{\@x}}%
983 % CORRECT THE FALSE NEGATIVE
984     {\edef\theresult{T}}%
985   \if q#1\else\theresult\fi
986 }
```

\testmatchingchar This routine checks for a specified match-character within a target string. Unlike `\isnextbyte`, this routine checks for characters (single- or multi-byte) and not just individual bytes. Additionally, rather than testing the match-character against the first byte of the test-string, the user specifies (through #2) which byte of the test-string should be compared to the match-character.

This routine is not as efficient as `\isnextbyte`, but much more versatile.

```
987 % CHECKS TO SEE IF [#2]'th CHARACTER IN STRING [#1] EQUALS [#3]
988 % RESULT STORED IN BOOLEAN \ifmatchingchar
989 \newif\ifmatchingchar
990 \newcommand\testmatchingchar[3]{%
```

Extract desired character from test string

```
991   \substring[e]{#1}{#2}{#2}\+%
```

Determine if the match-character is a multi-byte symbol.

```
992   \isnextbyte[q]{\EscapeChar}{#3}%
993   \if T\theresult%
```

Is the tested character also a multi-byte symbol?

```
994      \isnextbyte[q]{\EscapeChar}{\thestring}%
995      \if T\theresult%
```

Yes it is. . . Therefore, compare codes following the escape character

```
996        \edef\@testcode{\expandafter\@DiscardNextChar\expandafter{#3}}%
997        \edef\@teststring{\@DiscardNextChar{\thestring}}%
998        \if \@teststring\@testcode\matchingchartrue\else\matchingcharfalse\fi
999      \else
```

No, we are comparing a normal character against a multi-byte symbol (apples and oranges), a false comparison.

```
1000        \matchingcharfalse%
1001      \fi
1002    \else
```

No, we are comparing two normal one-byte characters, not a mult-byte character.

```
1003      \if \thestring#3\matchingchartrue\else\matchingcharfalse\fi
1004    \fi
1005 \?}
```

\testcapitalized   This routine checks to see if first character of string is capitalized. The only quirk is that the routine must ascertain whether that character is a single-byte character or a multi-byte character.

```
1006 \newif\ifcapitalized
1007 \newcommand\testcapitalized[1]{\+%
1008    \isnextbyte[q]{\EscapeChar}{#1}%
1009    \if T\theresult%
1010      \def\EncodingTreatment{e}%
1011      \edef\rotatingword{#1}%
```

Rotate the first [multi-byte] character of the string to the end of the string, lowering its case. Store as \@stringA.

```
1012      \def\AlphaCapsTreatment{2}%
1013      \@defineactions%
1014      \edef\@stringA{\ESCrotate{\expandafter\@gobble\rotatingword}}%
```

Rotate the first [multi-byte] character of the string to the end of the string, retaining its case. Store as \@stringB.

```
1015      \def\AlphaCapsTreatment{1}%
1016      \@defineactions%
1017      \edef\@stringB{\ESCrotate{\expandafter\@gobble\rotatingword}}%
1018    \else
```

. . . or, if the first character is a normal one-byte character. . . Rotate the first [normal] character of the string to the end of the string, lowering its case. Store as \@stringA.

```
1019      \def\AlphaCapsTreatment{2}%
1020      \edef\@stringA{\@rotate{#1}}%
```

Rotate the first [normal] character of the string to the end of the string, retaining its case. Store as `\@stringB`.

```
1021      \def\AlphaCapsTreatment{1}%
1022      \edef\@stringB{\@rotate{#1}}%
1023   \fi
```

Compare strings A and B, to see if changing the case of first letter altered the string

```
1024   \ifthenelse{\equal{\@stringA}{\@stringB}}%
1025   {\capitalizedfalse}{\capitalizedtrue}\?%
1026   \defaultTreatments%
1027 }
```

`\testuncapitalized` This routine is the complement of `\testcapitalized`. The only difference is that the `\@stringA` has its case made upper for the comparison, instead of lowered.

```
1028 \newif\ifuncapitalized
1029 \newcommand\testuncapitalized[1]{\+%
1030   \isnextbyte[q]{\EscapeChar}{#1}%
1031   \if T\theresult%
1032     \def\EncodingTreatment{e}%
1033     \edef\rotatingword{#1}%
1034     \def\AlphaTreatment{2}%
1035     \@defineactions%
1036     \edef\@stringA{\ESCrotate{\expandafter\@gobble\rotatingword}}%
1037     \def\AlphaTreatment{1}%
1038     \@defineactions%
1039     \edef\@stringB{\ESCrotate{\expandafter\@gobble\rotatingword}}%
1040   \else
1041     \def\AlphaTreatment{2}%
1042     \edef\@stringA{\@rotate{#1}}%
1043     \def\AlphaTreatment{1}%
1044     \edef\@stringB{\@rotate{#1}}%
1045   \fi
1046   \ifthenelse{\equal{\@stringA}{\@stringB}}%
1047   {\uncapitalizedfalse}{\uncapitalizedtrue}\?%
1048   \defaultTreatments%
1049 }
```

`\testleadingalpha` Test if the leading character of the string is alphabetic. This is simply accomplished by checking whether the string is either capitalized or uncapitalized. If non-alphabetic, it will show up as false for both those tests.

```
1050 \newif\ifleadingalpha
1051 \newcommand\testleadingalpha[1]{%
1052   \testcapitalized{#1}%
```

```
1053  \ifcapitalized
1054    \leadingalphatrue%
1055  \else
1056    \testuncapitalized{#1}%
1057    \ifuncapitalized
1058      \leadingalphatrue%
1059    \else
1060      \leadingalphafalse%
1061    \fi
1062  \fi
1063 }
```

\testuppercase   Checks to see if all alphabetic characters in a string are uppercase. Non-alphabetic characters don't affect the result, unless the string is composed solely of nonalphabetic characters, in which case the test results is false.

```
1064 \newif\ifuppercase
1065 \newcommand\testuppercase[1]{%
```

Strip all non-alphabetic characters. Save as **\@stringA**.

```
1066    \Treatments{1}{1}{0}{0}{0}{0}%
1067    \substring[e]{#1}{1}{\@MAXSTRINGSIZE}%
1068    \edef\@stringA{\thestring}%
```

Lower the case of all uppercase characters in **\@stringA**. Save as **\@stringB**. Compare these two strings.

```
1069    \def\AlphaTreatment{2}%
1070    \substring[e]{#1}{1}{\@MAXSTRINGSIZE}%
1071    \edef\@stringB{\thestring}%
1072    \ifthenelse{\equal{\@stringA}{\@stringB}}%
1073    {%
```

If the strings are equal, then all the alphabetic characters in the original string were uppercase. Need only check to make sure at least one alphabetic character was present in the original string.

```
1074      \@getstringlength{\@stringA}{@stringsize}%
1075      \ifthenelse{\value{@stringsize} = 0}%
1076      {\uppercasefalse}{\uppercasetrue}%
1077    }%
```

If strings are not equal, then the alphabetic characters of the original string were not all uppercase. Test false.

```
1078    {\uppercasefalse}%
1079    \defaultTreatments%
1080 }
```

\ifsolelyuppercase   Compare the original string to one made solely uppercase. If they are equal (and not composed solely of blankspaces), then the original string was solely uppercase

49

to begin with.

```
1081 \newif\ifsolelyuppercase
1082 \newcommand\testsolelyuppercase[1]{%
1083   \stringencode{#1}%
1084   \edef\@stringA{\thestring}%
1085   \solelyuppercase[e]{#1}%
1086   \edef\@stringB{\thestring}%
1087   \ifthenelse{\equal{\@stringA}{\@stringB}}%
1088   {%
1089     \noblanks[q]{\@stringA}%
1090     \@getstringlength{\thestring}{@stringsize}%
1091     \ifthenelse{\value{@stringsize} = 0}%
1092     {\solelyuppercasefalse}{\solelyuppercasetrue}%
1093   }%
1094   {\solelyuppercasefalse}%
1095   \defaultTreatments%
1096 }
```

\testlowercase   This routine is the complement to \testuppercase, with corresponding logic.

```
1097 \newif\iflowercase
1098 \newcommand\testlowercase[1]{%
1099   \Treatments{1}{1}{0}{0}{0}%
1100   \substring[e]{#1}{1}{\@MAXSTRINGSIZE}%
1101   \edef\@stringA{\thestring}%
1102   \def\AlphaCapsTreatment{2}%
1103   \substring[e]{#1}{1}{\@MAXSTRINGSIZE}%
1104   \edef\@stringB{\thestring}%
1105   \ifthenelse{\equal{\@stringA}{\@stringB}}%
1106   {%
1107     \@getstringlength{\@stringA}{@stringsize}%
1108     \ifthenelse{\value{@stringsize} = 0}%
1109     {\lowercasefalse}{\lowercasetrue}%
1110   }%
1111   {\lowercasefalse}%
1112   \defaultTreatments%
1113 }
```

\testsolelylowercase   This routine is the complement to \testsolelyuppercase, with corresponding
logic.

```
1114 \newif\ifsolelylowercase
1115 \newcommand\testsolelylowercase[1]{%
1116   \stringencode{#1}%
1117   \edef\@stringA{\thestring}%
1118   \solelylowercase[e]{#1}%
1119   \edef\@stringB{\thestring}%
1120   \ifthenelse{\equal{\@stringA}{\@stringB}}%
1121   {%
```

```
1122      \noblanks[q]{\@stringA}%
1123      \@getstringlength{\thestring}{@stringsize}%
1124      \ifthenelse{\value{@stringsize} = 0}%
1125      {\solelylowercasefalse}{\solelylowercasetrue}%
1126    }%
1127    {\solelylowercasefalse}%
1128    \defaultTreatments%
1129 }
```

\testalphabetic   Comparable to \testsolelyuppercase and \testsolelylowercase in its logic,
this routine tests whether the string is purely alphabetic or not.

```
1130 \newif\ifalphabetic
1131 \newcommand\testalphabetic[1]{%
1132    \stringencode{#1}%
1133    \edef\@stringA{\thestring}%
1134    \alphabetic[e]{#1}%
1135    \edef\@stringB{\thestring}%
1136    \ifthenelse{\equal{\@stringA}{\@stringB}}%
1137    {%
1138      \noblanks[q]{\@stringA}%
1139      \@getstringlength{\thestring}{@stringsize}%
1140      \ifthenelse{\value{@stringsize} = 0}%
1141      {\alphabeticfalse}{\alphabetictrue}%
1142    }%
1143    {\alphabeticfalse}%
1144    \defaultTreatments%
1145 }

1146 %
1147 %%%% SUPPORT ROUTINES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1148 %
```

The following routines support the execution of the stringstrings package.

\ESCrotate   After the escape character has been ascertained as the next character, this routine
operates on the subsequent escape code to rotate the symbol to end of string, in
the fashion of macro \@rotate.

```
1149 \newcommand\ESCrotate[1]{%
1150      \if\UvariCode#1\@uvariaction\else
1151      \if\UvariiCode#1\@uvariiaction\else
1152    \if\UvariiiCode#1\@uvariiiaction\else
1153    \if\@fromcode#1\@tostring\else
1154    \if\PipeCode#1\@pipeaction\else
1155      \if\DollarCode#1\@dollaraction\else
1156      \if\CaratCode#1\@carataction\else
1157      \if\CircumflexCode#1\@circumflexaction\else
1158        \if\TildeCode#1\@tildeaction\else
1159        \if\UmlautCode#1\@umlautaction\else
```

51

```
1160        \if\GraveCode#1\@graveaction\else
1161         \if\AcuteCode#1\@acuteaction\else
1162          \if\MacronCode#1\@macronaction\else
1163           \if\OverdotCode#1\@overdotaction\else
1164            \if\LeftBraceCode#1\@leftbraceaction\else
1165             \if\RightBraceCode#1\@rightbraceaction\else
1166              \if\UnderscoreCode#1\@underscoreaction\else
1167               \if\DaggerCode#1\@daggeraction\else
1168                \if\DoubleDaggerCode#1\@doubledaggeraction\else
1169                 \if\SectionSymbolCode#1\@sectionsymbolaction\else
1170                  \if\PilcrowCode#1\@pilcrowaction\else
1171                   \if\LBCode#1\@lbaction\else
1172                    \if\RBCode#1\@rbaction\else
1173  \if\BreveCode#1\@breveaction\else
1174   \if\CaronCode#1\@caronaction\else
1175    \if\DoubleAcuteCode#1\@doubleacuteaction\else
1176     \if\CedillaCode#1\@cedillaaction\else
1177      \if\UnderdotCode#1\@underdotaction\else
1178       \if\ArchJoinCode#1\@archjoinaction\else
1179        \if\LineUnderCode#1\@lineunderaction\else
1180         \if\CopyrightCode#1\@copyrightaction\else
1181          \if\PoundsCode#1\@poundsaction\else
1182           \if\AEscCode#1\@AEscaction\else
1183            \if\aescCode#1\@aescaction\else
1184             \if\OEthelCode#1\@OEthelaction\else
1185              \if\oethelCode#1\@oethelaction\else
1186               \if\AngstromCode#1\@Angstromaction\else
1187                \if\angstromCode#1\@angstromaction\else
1188                 \if\SlashedOCode#1\@slashedOaction\else
1189                  \if\SlashedoCode#1\@slashedoaction\else
1190                   \if\BarredlCode#1\@barredlaction\else
1191                    \if\BarredLCode#1\@barredLaction\else
1192                     \if\EszettCode#1\@eszettaction\else
1193                        \expandafter\@gobble#1\undecipherable%
1194                     \fi
1195                    \fi
1196                   \fi
1197                  \fi
1198                 \fi
1199                \fi
1200               \fi
1201              \fi
1202             \fi
1203            \fi
1204           \fi
1205          \fi
1206         \fi
1207        \fi
1208       \fi
1209      \fi
```

```
1210        \fi
1211       \fi
1212      \fi
1213     \fi
1214                            \fi
1215                           \fi
1216                          \fi
1217                         \fi
1218                        \fi
1219                       \fi
1220                      \fi
1221                     \fi
1222                    \fi
1223                   \fi
1224                  \fi
1225                 \fi
1226               \fi
1227              \fi
1228             \fi
1229            \fi
1230          \fi
1231        \fi
1232      \fi
1233    \fi
1234     \fi
1235      \fi
1236        \fi
1237 }
```

\@getnextword   A low-level routine designed to extract the next [space-delimited] word of the
                primary argument. It has several quirks: if the passed string has one leading
                space, it is included as part of next word. If it has two leading [hard]spaces,
                the 2nd hard space *is* the next word. Using the higher-level \getnextword deals
                automatically with these abberant possibilities.

```
1238 \newcommand\@getnextword[2][v]{%
1239   \defaultTreatments%
1240   \def\SeekBlankSpace{2}%
1241   \substring[#1]{#2}{1}{\@MAXSTRINGSIZE}%
1242   \def\SeekBlankSpace{0}%
1243 }
```

\@retokenizechar   This command is the guts of the retokenize command. It grabs the character
                   provided in string #1 and assigns it to a unique token whose name is created from
                   the string #2. The command has two primary \if branches. The first branch
                   is taken if the character is a special two-byte-encoded escape-sequence, while the
                   second branch is taken if the character is a &, %, #, a blankspace, or any simple
                   one-byte character.

```
1244 \newcommand\@retokenizechar[2]{%
1245   \isnextbyte[q]{\EscapeChar}{#1}%
1246   \if T\theresult%
1247   \edef\@ESCcode{\expandafter\@gobble#1}%
1248       \if\UvariCode\@ESCcode%
1249       \expandafter\def\csname#2\endcsname{\Uvari}\else
1250      \if\UvariiCode\@ESCcode%
1251      \expandafter\def\csname#2\endcsname{\Uvarii}\else
1252     \if\UvariiiCode\@ESCcode%
1253     \expandafter\def\csname#2\endcsname{\Uvariii}\else
1254    \if\PipeCode\@ESCcode%
1255    \expandafter\def\csname#2\endcsname{\Pipe}\else
1256     \if\DollarCode\@ESCcode%
1257    \expandafter\def\csname#2\endcsname{\$}\else
1258     \if\CaratCode\@ESCcode%
1259     \expandafter\def\csname#2\endcsname{\Carat}\else
1260      \if\CircumflexCode\@ESCcode%
1261     \expandafter\def\csname#2\endcsname{\^}\else
1262      \if\TildeCode\@ESCcode%
1263     \expandafter\def\csname#2\endcsname{\~}\else
1264      \if\UmlautCode\@ESCcode%
1265     \expandafter\def\csname#2\endcsname{\"}\else
1266      \if\GraveCode\@ESCcode%
1267     \expandafter\def\csname#2\endcsname{\'}\else
1268       \if\AcuteCode\@ESCcode%
1269      \expandafter\def\csname#2\endcsname{\'}\else
1270       \if\MacronCode\@ESCcode%
1271      \expandafter\def\csname#2\endcsname{\=}\else
1272       \if\OverdotCode\@ESCcode%
1273       \expandafter\def\csname#2\endcsname{\.}\else
1274        \if\LeftBraceCode\@ESCcode%
1275        \expandafter\def\csname#2\endcsname{\{}\else
1276         \if\RightBraceCode\@ESCcode%
1277        \expandafter\def\csname#2\endcsname{\}}\else
1278         \if\UnderscoreCode\@ESCcode%
1279        \expandafter\def\csname#2\endcsname{\_}\else
1280         \if\DaggerCode\@ESCcode%
1281        \expandafter\def\csname#2\endcsname{\dag}\else
1282         \if\DoubleDaggerCode\@ESCcode%
1283        \expandafter\def\csname#2\endcsname{\ddag}\else
1284          \if\SectionSymbolCode\@ESCcode%
1285          \expandafter\def\csname#2\endcsname{\S}\else
1286           \if\PilcrowCode\@ESCcode%
1287          \expandafter\def\csname#2\endcsname{\P}\else
1288            \if\LBCode\@ESCcode%
1289            \expandafter\def\csname#2\endcsname{\SaveLB}\else
1290             \if\RBCode\@ESCcode%
1291             \expandafter\def\csname#2\endcsname{\SaveRB}\else
1292 \if\BreveCode\@ESCcode\expandafter\def\csname#2\endcsname{\u}\else
1293  \if\CaronCode\@ESCcode\expandafter\def\csname#2\endcsname{\v}\else
```

```
1294    \if\DoubleAcuteCode\@ESCcode\expandafter\def\csname#2\endcsname{\H}\else
1295     \if\CedillaCode\@ESCcode\expandafter\def\csname#2\endcsname{\c}\else
1296     \if\UnderdotCode\@ESCcode\expandafter\def\csname#2\endcsname{\d}\else
1297     \if\ArchJoinCode\@ESCcode\expandafter\def\csname#2\endcsname{\t}\else
1298    \if\LineUnderCode\@ESCcode\expandafter\def\csname#2\endcsname{\b}\else
1299   \if\CopyrightCode\@ESCcode\expandafter\def\csname#2\endcsname{\copyright}\else
1300    \if\PoundsCode\@ESCcode\expandafter\def\csname#2\endcsname{\pounds}\else
1301     \if\AEscCode\@ESCcode\expandafter\def\csname#2\endcsname{\AE}\else
1302      \if\aescCode\@ESCcode\expandafter\def\csname#2\endcsname{\ae}\else
1303       \if\OEthelCode\@ESCcode\expandafter\def\csname#2\endcsname{\OE}\else
1304        \if\oethelCode\@ESCcode\expandafter\def\csname#2\endcsname{\oe}\else
1305         \if\AngstromCode\@ESCcode\expandafter\def\csname#2\endcsname{\AA}\else
1306          \if\angstromCode\@ESCcode\expandafter\def\csname#2\endcsname{\aa}\else
1307         \if\SlashedOCode\@ESCcode\expandafter\def\csname#2\endcsname{\O}\else
1308        \if\SlashedoCode\@ESCcode\expandafter\def\csname#2\endcsname{\o}\else
1309       \if\BarredlCode\@ESCcode\expandafter\def\csname#2\endcsname{\l}\else
1310      \if\BarredLCode\@ESCcode\expandafter\def\csname#2\endcsname{\L}\else
1311     \if\EszettCode\@ESCcode\expandafter\def\csname#2\endcsname{\ss}\else
1312    \expandafter\def\csname#2\endcsname{\undecipherable}%
1313     \fi
1314      \fi
1315       \fi
1316        \fi
1317         \fi
1318          \fi
1319         \fi
1320        \fi
1321       \fi
1322      \fi
1323     \fi
1324    \fi
1325   \fi
1326    \fi
1327     \fi
1328      \fi
1329      \fi
1330     \fi
1331    \fi
1332   \fi
1333                       \fi
1334                        \fi
1335                       \fi
1336                      \fi
1337                     \fi
1338                    \fi
1339                   \fi
1340                  \fi
1341                 \fi
1342                \fi
1343               \fi
```

```
1344              \fi
1345            \fi
1346          \fi
1347        \fi
1348      \fi
1349      \fi
1350      \fi
1351    \fi
1352    \fi
1353     \fi
1354      \fi
1355  \else
1356    \expandafter\ifx\expandafter\&#1%
1357                      \expandafter\def\csname#2\endcsname{\&}\else
1358      \expandafter\ifx\expandafter\%#1%
1359                      \expandafter\def\csname#2\endcsname{\%}\else
1360        \expandafter\ifx\expandafter\##1%
1361                      \expandafter\def\csname#2\endcsname{\#}\else
1362        \if\EncodedBlankSpace#1\expandafter\def\csname#2\endcsname{~}\else
1363          \expandafter\edef\csname#2\endcsname{#1}%
1364        \fi
1365      \fi
1366     \fi
1367    \fi
1368  \fi
1369 }
```

\@defineactions This routine defines how encoded characters are to be treated by the \ESCrotate routine, depending on the [encoding, capitalization, blank, symbol, *etc.*] treatments that have been *a priori* specified.

```
1370 % \@blankaction AND OTHER ...action'S ARE SET, DEPENDING ON VALUES OF
1371 % TREATMENT FLAGS.  CHARS ARE EITHER ENCODED, DECODED, OR REMOVED.
1372 \newcommand\@defineactions{%
1373 % SET UP TREATMENT FOR SPACES, ENCODED SPACES, AND [REENCODED] SYMBOLS
1374   \if e\EncodingTreatment%
1375 % ENCODE SPACES, KEEP ENCODED SPACES ENCODED, ENCODE SYMBOLS.
1376     \edef\@blankaction{\EncodedBlankSpace}%
1377     \def\@dollaraction{\EncodedDollar}%
1378     \def\@pipeaction{\EncodedPipe}%
1379     \def\@uvariaction{\EncodedUvari}%
1380     \def\@uvariiaction{\EncodedUvarii}%
1381     \def\@uvariiiaction{\EncodedUvariii}%
1382     \def\@carataction{\EncodedCarat}%
1383     \def\@circumflexaction{\EncodedCircumflex}%
1384     \def\@tildeaction{\EncodedTilde}%
1385     \def\@umlautaction{\EncodedUmlaut}%
1386     \def\@graveaction{\EncodedGrave}%
1387     \def\@acuteaction{\EncodedAcute}%
1388     \def\@macronaction{\EncodedMacron}%
```

```
1389      \def\@overdotaction{\EncodedOverdot}%
1390      \def\@breveaction{\EncodedBreve}%
1391      \def\@caronaction{\EncodedCaron}%
1392      \def\@doubleacuteaction{\EncodedDoubleAcute}%
1393      \def\@cedillaaction{\EncodedCedilla}%
1394      \def\@underdotaction{\EncodedUnderdot}%
1395      \def\@archjoinaction{\EncodedArchJoin}%
1396      \def\@lineunderaction{\EncodedLineUnder}%
1397      \def\@copyrightaction{\EncodedCopyright}%
1398      \def\@poundsaction{\EncodedPounds}%
1399      \def\@leftbraceaction{\EncodedLeftBrace}%
1400      \def\@rightbraceaction{\EncodedRightBrace}%
1401      \def\@underscoreaction{\EncodedUnderscore}%
1402      \def\@daggeraction{\EncodedDagger}%
1403      \def\@doubledaggeraction{\EncodedDoubleDagger}%
1404      \def\@sectionsymbolaction{\EncodedSectionSymbol}%
1405      \def\@pilcrowaction{\EncodedPilcrow}%
1406      \def\@eszettaction{\EncodedEszett}%
1407      \def\@lbaction{\EncodedLB}%
1408      \def\@rbaction{\EncodedRB}%
1409      \if 2\AlphaCapsTreatment%
1410        \def\@AEscaction{\Encodedaesc}%
1411        \def\@OEthelaction{\Encodedoethel}%
1412        \def\@Angstromaction{\Encodedangstrom}%
1413        \def\@slashedOaction{\EncodedSlashedo}%
1414        \def\@barredLaction{\EncodedBarredl}%
1415      \else
1416        \def\@AEscaction{\EncodedAEsc}%
1417        \def\@OEthelaction{\EncodedOEthel}%
1418        \def\@Angstromaction{\EncodedAngstrom}%
1419        \def\@slashedOaction{\EncodedSlashedO}%
1420        \def\@barredLaction{\EncodedBarredL}%
1421      \fi
1422      \if 2\AlphaTreatment%
1423        \def\@aescaction{\EncodedAEsc}%
1424        \def\@oethelaction{\EncodedOEthel}%
1425        \def\@angstromaction{\EncodedAngstrom}%
1426        \def\@slashedoaction{\EncodedSlashedO}%
1427        \def\@barredlaction{\EncodedBarredL}%
1428      \else
1429        \def\@aescaction{\Encodedaesc}%
1430        \def\@oethelaction{\Encodedoethel}%
1431        \def\@angstromaction{\Encodedangstrom}%
1432        \def\@slashedoaction{\EncodedSlashedo}%
1433        \def\@barredlaction{\EncodedBarredl}%
1434      \fi
1435    \else
1436 % EncodingTreatment=v or q:
1437 % LEAVE SPACES ALONE; RESTORE ENCODED SPACES AND SYMBOLS
1438      \def\@blankaction{\BlankSpace}%
```

```
1439       \def\@dollaraction{\Dollar}%
1440       \def\@pipeaction{\Pipe}%
1441       \def\@uvariaction{\Uvari}%
1442       \def\@uvariiaction{\Uvarii}%
1443       \def\@uvariiiaction{\Uvariii}%
1444       \def\@carataction{\Carat}%
1445       \def\@circumflexaction{\Circumflex}%
1446       \def\@tildeaction{\Tilde}%
1447       \def\@umlautaction{\Umlaut}%
1448       \def\@graveaction{\Grave}%
1449       \def\@acuteaction{\Acute}%
1450       \def\@macronaction{\Macron}%
1451       \def\@overdotaction{\Overdot}%
1452       \def\@breveaction{\Breve}%
1453       \def\@caronaction{\Caron}%
1454       \def\@doubleacuteaction{\DoubleAcute}%
1455       \def\@cedillaaction{\Cedilla}%
1456       \def\@underdotaction{\Underdot}%
1457       \def\@archjoinaction{\ArchJoin}%
1458       \def\@lineunderaction{\LineUnder}%
1459       \def\@copyrightaction{\Copyright}%
1460       \def\@poundsaction{\Pounds}%
1461       \def\@leftbraceaction{\LeftBrace}%
1462       \def\@rightbraceaction{\RightBrace}%
1463       \def\@underscoreaction{\Underscore}%
1464       \def\@daggeraction{\Dagger}%
1465       \def\@doubledaggeraction{\DoubleDagger}%
1466       \def\@sectionsymbolaction{\SectionSymbol}%
1467       \def\@pilcrowaction{\Pilcrow}%
1468       \def\@eszettaction{\Eszett}%
1469       \def\@lbaction{\UnencodedLB}%
1470       \def\@rbaction{\UnencodedRB}%
1471       \if 2\AlphaCapsTreatment%
1472         \def\@AEscaction{\aesc}%
1473         \def\@OEthelaction{\oethel}%
1474         \def\@Angstromaction{\angstrom}%
1475         \def\@slashedOaction{\Slashedo}%
1476         \def\@barredLaction{\Barredl}%
1477       \else
1478         \def\@AEscaction{\AEsc}%
1479         \def\@OEthelaction{\OEthel}%
1480         \def\@Angstromaction{\Angstrom}%
1481         \def\@slashedOaction{\SlashedO}%
1482         \def\@barredLaction{\BarredL}%
1483       \fi
1484       \if 2\AlphaTreatment%
1485         \def\@aescaction{\AEsc}%
1486         \def\@oethelaction{\OEthel}%
1487         \def\@angstromaction{\Angstrom}%
1488         \def\@slashedoaction{\SlashedO}%
```

```
1489        \def\@barredlaction{\BarredL}%
1490      \else
1491        \def\@aescaction{\aesc}%
1492        \def\@oethelaction{\oethel}%
1493        \def\@angstromaction{\angstrom}%
1494        \def\@slashedoaction{\Slashedo}%
1495        \def\@barredlaction{\Barredl}%
1496      \fi
1497    \fi
1498 % REMOVE SPACES AND ENCODED SPACES?
1499    \if 0\BlankTreatment%
1500      \edef\@blankaction{}%
1501    \fi
1502 % REMOVE ENCODED SYMBOLS?
1503    \if 0\SymbolTreatment%
1504      \def\@dollaraction{}%
1505      \def\@pipeaction{}%
1506      \def\@carataction{}%
1507      \def\@circumflexaction{}%
1508      \def\@tildeaction{}%
1509      \def\@umlautaction{}%
1510      \def\@graveaction{}%
1511      \def\@acuteaction{}%
1512      \def\@macronaction{}%
1513      \def\@overdotaction{}%
1514      \def\@breveaction{}%
1515      \def\@caronaction{}%
1516      \def\@doubleacuteaction{}%
1517      \def\@cedillaaction{}%
1518      \def\@underdotaction{}%
1519      \def\@archjoinaction{}%
1520      \def\@lineunderaction{}%
1521      \def\@copyrightaction{}%
1522      \def\@poundsaction{}%
1523      \def\@leftbraceaction{}%
1524      \def\@rightbraceaction{}%
1525      \def\@underscoreaction{}%
1526      \def\@daggeraction{}%
1527      \def\@doubledaggeraction{}%
1528      \def\@sectionsymbolaction{}%
1529      \def\@pilcrowaction{}%
1530      \def\@lbaction{}%
1531      \def\@rbaction{}%
1532    \fi
1533 % REMOVE ENCODED ALPHACAPS?
1534    \if 0\AlphaCapsTreatment%
1535      \def\@AEscaction{}%
1536      \def\@OEthelaction{}%
1537      \def\@Angstromaction{}%
1538      \def\@slashedOaction{}%
```

```
1539     \def\@barredLaction{}%
1540   \fi
1541 % REMOVE ENCODED ALPHA?
1542   \if 0\AlphaTreatment%
1543     \def\@aescaction{}%
1544     \def\@oethelaction{}%
1545     \def\@angstromaction{}%
1546     \def\@slashedoaction{}%
1547     \def\@barredlaction{}%
1548     \def\@eszettaction{}%
1549   \fi
1550 }
```

\@forcecapson   Force capitalization of strings processed by \substring for the time being.

```
1551 \newcommand\@forcecapson{%
1552   \def\AlphaTreatment{2}%
1553   \def\AlphaCapsTreatment{1}%
1554 }
```

\@relaxcapson   Restore prior treatments following a period of enforced capitalization.

```
1555 \newcommand\@relaxcapson{%
1556   \let\AlphaTreatment\SaveAlphaTreatment%
1557   \let\AlphaCapsTreatment\SaveAlphaCapsTreatment%
1558   \@defineactions%
1559 }
```

\@decodepointer   As pertains to arguments 3 and 4 of \substring, this routine implements use
of the $ character to mean END-OF-STRING, and $-{*integer*} for addressing
relative to the END-OF-STRING.

```
1560 \newcommand\@decodepointer[2][\value{@stringsize}]{%
1561   \isnextbyte[q]{$}{#2}%
1562   \if T\theresult%
1563     \isnextbyte[q]{-}{\expandafter\@gobble#2}%
1564     \if T\theresult%
1565       \setcounter{@@@letterindex}{#1}%
1566       \@gobblearg{#2}{2}%
1567       \addtocounter{@@@letterindex}{-\gobbledword}%
1568       \edef\@fromtoindex{\value{@@@letterindex}}%
1569     \else
1570       \edef\@fromtoindex{#1}%
1571     \fi
1572   \else
1573     \edef\@fromtoindex{#2}%
1574   \fi
1575 }
```

\@getstringlength   Get's string length of #1, puts result in counter #2.

```
1576 \newcommand\@getstringlength[2]{%
1577   \edef\@@teststring{#1\endofstring}%
1578   \ifthenelse{\equal{\@@teststring}{\endofstring}}%
1579   {\setcounter{#2}{0}}%
1580   {%
1581     \setcounter{@gobblesize}{1}%
1582     \whiledo{\value{@gobblesize} < \@MAXSTRINGSIZE}{%
1583 %
1584       \@gobblearg{\@@teststring}{1}%
1585       \edef\@@teststring{\gobbledword}%
1586       \ifthenelse{\equal{\@@teststring}{\endofstring}}%
1587         {\setcounter{#2}{\value{@gobblesize}}%
1588           \setcounter{@gobblesize}{\@MAXSTRINGSIZE}}%
1589         {\addtocounter{@gobblesize}{1}}%
1590     }%
1591   }%
1592 }
```

\@gobblearg Gobble first #2 characters from string #1. The result is stored in \gobbledword.
Two-byte escape sequences, when encountered, count as a single gobble.

```
1593 \newcommand\@gobblearg[2]{%
1594   \setcounter{@letterindex}{0}%
1595   \setcounter{@gobbleindex}{#2}%
1596   \edef\gobbledword{#1}%
1597   \whiledo{\value{@letterindex} < \value{@gobbleindex}}{%
1598     \isnextbyte[q]{\EscapeChar}{\gobbledword}%
1599     \if T\theresult%
1600 %     GOBBLE ESCAPE CHARACTER
1601       \edef\gobbledword{\@DiscardNextChar{\gobbledword}}%
1602     \fi
1603 %   GOBBLE NORMAL CHARACTER OR ESCAPE CODE
1604     \edef\gobbledword{\@DiscardNextChar{\gobbledword}}%
1605     \addtocounter{@letterindex}{1}%
1606   }%
1607 }
```

\@DiscardNextChar Remove the next character from the argument string. Since \@gobble skips
spaces, the routine must first look for the case of a leading blankspace. If none
is found, proceed with a normal \@gobble. Note: as per LaTeX convention,
\@DiscardNextChar treats double/multi-softspaces as single space.

```
1608 \newcommand\@DiscardNextChar[1]{%
1609   \expandafter\if\expandafter\BlankSpace#1\else
1610     \expandafter\@gobble#1%
1611   \fi
1612 }
```

\@convertsymboltostring Routine for converting an encodable symbol (#3) into string (#4), for every occu-

rance in the given string **#2**.

```
1613 \newcommand\@convertsymboltostring[4][v]{%
1614   \def\@fromcode{#3}%
1615   \def\@tostring{#4}%
1616   \def\EncodingTreatment{e}%
1617   \substring[e]{#2}{1}{\@MAXSTRINGSIZE}%
1618   \@convertoff%
1619   \if e#1\else\substring[#1]{\thestring}{1}{\@MAXSTRINGSIZE}\fi%
1620 }
```

\@convertbytetostring Routine for converting an plain byte (**#3**) into string (**#4**), for every occurance in the given string **#2**.

```
1621 \newcommand\@convertbytetostring[4][v]{%
1622   \def\@frombyte{#3}%
1623   \def\@tostring{#4}%
1624   \def\EncodingTreatment{e}%
1625   \substring[e]{#2}{1}{\@MAXSTRINGSIZE}%
1626   \@convertoff%
1627   \if e#1\else\substring[#1]{\thestring}{1}{\@MAXSTRINGSIZE}\fi%
1628 }
```

\@treatleadingspaces This routine will address the leading spaces of string **#2**. If argument **#3** is an 'x' character, those leading spaces will be deleted from the string. Otherwise, those leading spaces will be rotated to the end of the string.

```
1629 \newcommand\@treatleadingspaces[3][v]{\+%
1630   \defaultTreatments%
1631   \edef\thestring{#2}%
1632   \@getstringlength{\thestring}{@stringsize}%
1633   \setcounter{@maxrotation}{\value{@stringsize}}%
1634   \setcounter{@letterindex}{0}%
1635   \whiledo{\value{@letterindex} < \value{@maxrotation}}{%
1636     \addtocounter{@letterindex}{1}%
1637     \isnextbyte[q]{\EncodedBlankSpace}{\thestring}%
1638     \if F\theresult\isnextbyte[q]{\BlankSpace}{\thestring}\fi%
1639     \if T\theresult%
1640       \isnextbyte[q]{#3}{x}%
1641       \if F\theresult%
1642 %       NORMAL OR ENCODED BLANK... ROTATE IT
1643         \edef\thestring{\@rotate{\thestring}}%
1644       \else
1645 %       NORMAL OR ENCODED BLANK... DELETE IT (IF 3rd ARG=X)
1646         \@gobblearg{\thestring}{1}%
1647         \edef\thestring{\gobbledword}%
1648       \fi
1649     \else
1650       \setcounter{@maxrotation}{\value{@letterindex}}%
1651     \fi
```

```
1652    }\?%
1653    \substring[#1]{\thestring}{1}{\@MAXSTRINGSIZE}%
1654 }
```

\@convertoff This routine is an initialization routine to guarantee that there is no conversion of
\@frombyte to \@tostring, until further notice. It accomplishes this by setting
up such that subsequent \if\@frombyte and \if\@fromcode clauses will auto-
matically fail.

```
1655 \newcommand\@convertoff{\def\@frombyte{xy}\def\@tostring{}%
1656                        \def\@fromcode{xy}}
1657 \@convertoff
```

\@rotate The following code is the engine of the string manipulation routine. It is a tree
of successive LaTeX commands (each of which is composed of an \if... cascade)
which have the net effect of rotating the first letter of the string into the last po-
sition. Depending on modes set by \@defineactions and \defaultTreatments,
the leading character is either encoded, decoded, or removed in the process. Note:
\@rotate loses track of double/multi-spaces, per LaTeX convention, unless encoded
blanks (~) are used.

```
1658 \newcommand\@rotate[1]{%
1659 % CHECK BYTE CONVERSION TEST FIRST
1660  \if \@frombyte#1\@tostring\else
1661 % MUST CHECK FOR MULTI-BYTE CHARACTERS NEXT, SO THAT ENCODING CHARACTER
1662 % ISN'T MISTAKEN FOR A NORMAL CHARACTER LATER IN MACRO.
1663    \if 0\SymbolTreatment%
1664      \@removeExpandableSymbols{#1}%
1665    \else
1666      \@rotateExpandableSymbols{#1}%
1667    \fi
1668  \fi
1669 }
1670
1671 \newcommand\@rotateExpandableSymbols[1]{%
1672 % INDIRECT (EXPANDABLE) SYMBOLS
1673    \expandafter\ifx\expandafter\&#1\&\else
1674     \expandafter\ifx\expandafter\%#1\%\else
1675      \expandafter\ifx\expandafter\##1\#\else
1676       \@rotateBlankSpaces{#1}%
1677      \fi
1678     \fi
1679    \fi
1680 }
1681
1682 \newcommand\@removeExpandableSymbols[1]{%
1683 % INDIRECT (EXPANDABLE) SYMBOLS
1684    \expandafter\ifx\expandafter\&#1\else
1685     \expandafter\ifx\expandafter\%#1\else
```

63

```
1686      \expandafter\ifx\expandafter\##1\else
1687       \@rotateBlankSpaces{#1}%
1688      \fi
1689     \fi
1690    \fi
1691 }
1692
1693 \newcommand\@rotateBlankSpaces[1]{%
1694    \expandafter\ifx\expandafter$#1$\else% <---RETAIN GOING INTO/FROM MATH MODE
1695 % THE FOLLOWING FINDS TILDES, BUT MUST COME AFTER EXPANDABLE SYMBOL
1696 % SEARCH, OR ELSE IT FINDS THEM TOO, BY MISTAKE.
1697    \if \EncodedBlankSpace#1\@blankaction\else% <--- FINDS REENCODED TILDE
1698 %   THE FOLLOWING SHOULD FIND TILDES, BUT DOESN'T... THUS, COMMENTED OUT.
1699 %   \expandafter\ifx\expandafter\EncodedBlankSpace#1\@blankaction\else
1700     \if \BlankSpace#1\@blankaction\else
1701      \if 2\AlphaTreatment%
1702        \@chcaseAlpha{#1}%
1703      \else
1704       \if 0\AlphaTreatment%
1705         \@removeAlpha{#1}%
1706       \else
1707         \@rotateAlpha{#1}%
1708       \fi
1709      \fi
1710     \fi
1711 %   \fi
1712    \fi
1713   \fi
1714 }
1715
1716 \newcommand\@rotateAlpha[1]{%
1717 % LOWERCASE
1718    \if a#1a\else
1719     \if b#1b\else
1720      \if c#1c\else
1721       \if d#1d\else
1722        \if e#1e\else
1723         \if f#1f\else
1724          \if g#1g\else
1725           \if h#1h\else
1726            \if i#1i\else
1727             \if j#1j\else
1728              \if k#1k\else
1729               \if l#1l\else
1730                \if m#1m\else
1731                 \if n#1n\else
1732                  \if o#1o\else
1733                   \if p#1p\else
1734                    \if q#1q\else
1735                     \if r#1r\else
```

64

```
1736                        \if s#1s\else
1737                         \if t#1t\else
1738                          \if u#1u\else
1739                           \if v#1v\else
1740                            \if w#1w\else
1741                             \if x#1x\else
1742                              \if y#1y\else
1743                               \if z#1z\else
1744                                \if 2\AlphaCapsTreatment%
1745                                   \@chcaseAlphaCaps{#1}%
1746                                \else
1747                                 \if 0\AlphaCapsTreatment%
1748                                    \@removeAlphaCaps{#1}%
1749                                 \else
1750                                    \@rotateAlphaCaps{#1}%
1751                                 \fi
1752                                \fi
1753                               \fi
1754                              \fi
1755                             \fi
1756                            \fi
1757                           \fi
1758                          \fi
1759                         \fi
1760                        \fi
1761                       \fi
1762                      \fi
1763                     \fi
1764                    \fi
1765                   \fi
1766                  \fi
1767                 \fi
1768                \fi
1769               \fi
1770              \fi
1771             \fi
1772            \fi
1773           \fi
1774          \fi
1775         \fi
1776        \fi
1777       \fi
1778      \fi
1779 }
1780
1781 \newcommand\@removeAlpha[1]{%
1782 % LOWERCASE
1783   \if a#1\else
1784    \if b#1\else
1785     \if c#1\else
```

```
1786          \if d#1\else
1787           \if e#1\else
1788            \if f#1\else
1789             \if g#1\else
1790              \if h#1\else
1791               \if i#1\else
1792                \if j#1\else
1793                 \if k#1\else
1794                  \if l#1\else
1795                   \if m#1\else
1796                    \if n#1\else
1797                     \if o#1\else
1798                      \if p#1\else
1799                       \if q#1\else
1800                        \if r#1\else
1801                         \if s#1\else
1802                          \if t#1\else
1803                           \if u#1\else
1804                            \if v#1\else
1805                             \if w#1\else
1806                              \if x#1\else
1807                               \if y#1\else
1808                                \if z#1\else
1809                                 \if 2\AlphaCapsTreatment%
1810                                   \@chcaseAlphaCaps{#1}%
1811                                 \else
1812                                   \if 0\AlphaCapsTreatment%
1813                                     \@removeAlphaCaps{#1}%
1814                                   \else
1815                                     \@rotateAlphaCaps{#1}%
1816                                   \fi
1817                                 \fi
1818                                \fi
1819                               \fi
1820                              \fi
1821                             \fi
1822                            \fi
1823                           \fi
1824                          \fi
1825                         \fi
1826                        \fi
1827                       \fi
1828                      \fi
1829                     \fi
1830                    \fi
1831                   \fi
1832                  \fi
1833                 \fi
1834                \fi
1835               \fi
```

```
1836            \fi
1837          \fi
1838        \fi
1839      \fi
1840     \fi
1841    \fi
1842   \fi
1843  \fi
1844 }
1845
1846 \newcommand\@chcaseAlpha[1]{%
1847 % LOWERCASE TO UPPERCASE
1848   \if a#1A\else
1849    \if b#1B\else
1850     \if c#1C\else
1851      \if d#1D\else
1852       \if e#1E\else
1853        \if f#1F\else
1854         \if g#1G\else
1855          \if h#1H\else
1856           \if i#1I\else
1857            \if j#1J\else
1858             \if k#1K\else
1859              \if l#1L\else
1860               \if m#1M\else
1861                \if n#1N\else
1862                 \if o#1O\else
1863                  \if p#1P\else
1864                   \if q#1Q\else
1865                    \if r#1R\else
1866                     \if s#1S\else
1867                      \if t#1T\else
1868                       \if u#1U\else
1869                        \if v#1V\else
1870                         \if w#1W\else
1871                          \if x#1X\else
1872                           \if y#1Y\else
1873                            \if z#1Z\else
1874                             \if 2\AlphaCapsTreatment%
1875                               \@chcaseAlphaCaps{#1}%
1876                             \else
1877                               \if 0\AlphaCapsTreatment%
1878                                 \@removeAlphaCaps{#1}%
1879                               \else
1880                                 \@rotateAlphaCaps{#1}%
1881                               \fi
1882                             \fi
1883                            \fi
1884                           \fi
1885                          \fi
```

```
1886                         \fi
1887                          \fi
1888                         \fi
1889                        \fi
1890                       \fi
1891                      \fi
1892                     \fi
1893                    \fi
1894                   \fi
1895                  \fi
1896                 \fi
1897                \fi
1898               \fi
1899              \fi
1900             \fi
1901            \fi
1902           \fi
1903          \fi
1904         \fi
1905        \fi
1906       \fi
1907     \fi
1908   \fi
1909 }
1910
1911 \newcommand\@rotateAlphaCaps[1]{%
1912 % UPPERCASE
1913   \if A#1A\else
1914    \if B#1B\else
1915     \if C#1C\else
1916      \if D#1D\else
1917       \if E#1E\else
1918        \if F#1F\else
1919         \if G#1G\else
1920          \if H#1H\else
1921           \if I#1I\else
1922            \if J#1J\else
1923             \if K#1K\else
1924              \if L#1L\else
1925               \if M#1M\else
1926                \if N#1N\else
1927                 \if O#1O\else
1928                  \if P#1P\else
1929                   \if Q#1Q\else
1930                    \if R#1R\else
1931                     \if S#1S\else
1932                      \if T#1T\else
1933                       \if U#1U\else
1934                        \if V#1V\else
1935                         \if W#1W\else
```

68

```
1936                        \if X#1X\else
1937                         \if Y#1Y\else
1938                          \if Z#1Z\else
1939                           \if O\NumeralTreatment%
1940                             \@removeNumerals{#1}%
1941                           \else
1942                             \@rotateNumerals{#1}%
1943                           \fi
1944                          \fi
1945                         \fi
1946                        \fi
1947                       \fi
1948                      \fi
1949                     \fi
1950                    \fi
1951                   \fi
1952                  \fi
1953                 \fi
1954                \fi
1955               \fi
1956              \fi
1957             \fi
1958            \fi
1959           \fi
1960          \fi
1961         \fi
1962        \fi
1963       \fi
1964      \fi
1965     \fi
1966    \fi
1967   \fi
1968  \fi
1969 \fi
1970 }
1971
1972 \newcommand\@removeAlphaCaps[1]{%
1973 % UPPERCASE
1974  \if A#1\else
1975   \if B#1\else
1976    \if C#1\else
1977     \if D#1\else
1978      \if E#1\else
1979       \if F#1\else
1980        \if G#1\else
1981         \if H#1\else
1982          \if I#1\else
1983           \if J#1\else
1984            \if K#1\else
1985             \if L#1\else
```

69

```
1986                  \if M#1\else
1987                   \if N#1\else
1988                    \if O#1\else
1989                     \if P#1\else
1990                      \if Q#1\else
1991                       \if R#1\else
1992                        \if S#1\else
1993                         \if T#1\else
1994                          \if U#1\else
1995                           \if V#1\else
1996                            \if W#1\else
1997                             \if X#1\else
1998                              \if Y#1\else
1999                               \if Z#1\else
2000                                \if 0\NumeralTreatment%
2001                                  \@removeNumerals{#1}%
2002                                \else
2003                                  \@rotateNumerals{#1}%
2004                                \fi
2005                               \fi
2006                              \fi
2007                             \fi
2008                            \fi
2009                           \fi
2010                          \fi
2011                         \fi
2012                        \fi
2013                       \fi
2014                      \fi
2015                     \fi
2016                    \fi
2017                   \fi
2018                  \fi
2019                 \fi
2020                \fi
2021               \fi
2022              \fi
2023             \fi
2024            \fi
2025           \fi
2026          \fi
2027         \fi
2028        \fi
2029       \fi
2030      \fi
2031 }
2032
2033 \newcommand\@chcaseAlphaCaps[1]{%
2034 % UPPERCASE TO LOWERCASE
2035   \if A#1a\else
```

```
2036     \if B#1b\else
2037      \if C#1c\else
2038       \if D#1d\else
2039        \if E#1e\else
2040         \if F#1f\else
2041          \if G#1g\else
2042           \if H#1h\else
2043            \if I#1i\else
2044             \if J#1j\else
2045              \if K#1k\else
2046               \if L#1l\else
2047                \if M#1m\else
2048                 \if N#1n\else
2049                  \if O#1o\else
2050                   \if P#1p\else
2051                    \if Q#1q\else
2052                     \if R#1r\else
2053                      \if S#1s\else
2054                       \if T#1t\else
2055                        \if U#1u\else
2056                         \if V#1v\else
2057                          \if W#1w\else
2058                           \if X#1x\else
2059                            \if Y#1y\else
2060                             \if Z#1z\else
2061                              \if O\NumeralTreatment%
2062                                \@removeNumerals{#1}%
2063                              \else
2064                                \@rotateNumerals{#1}%
2065                              \fi
2066                             \fi
2067                            \fi
2068                           \fi
2069                          \fi
2070                         \fi
2071                        \fi
2072                       \fi
2073                      \fi
2074                     \fi
2075                    \fi
2076                   \fi
2077                  \fi
2078                 \fi
2079                \fi
2080               \fi
2081              \fi
2082             \fi
2083            \fi
2084           \fi
2085          \fi
```

```
2086        \fi
2087       \fi
2088      \fi
2089     \fi
2090    \fi
2091   \fi
2092 }
2093
2094 \newcommand\@rotateNumerals[1]{%
2095 % NUMERALS
2096   \if 1#11\else
2097    \if 2#12\else
2098     \if 3#13\else
2099      \if 4#14\else
2100       \if 5#15\else
2101        \if 6#16\else
2102         \if 7#17\else
2103          \if 8#18\else
2104           \if 9#19\else
2105            \if 0#10\else
2106             \if 0\PunctuationTreatment%
2107               \@removePunctuation{#1}%
2108             \else
2109               \@rotatePunctuation{#1}%
2110             \fi
2111            \fi
2112           \fi
2113          \fi
2114         \fi
2115        \fi
2116       \fi
2117      \fi
2118     \fi
2119    \fi
2120   \fi
2121 }
2122
2123 \newcommand\@removeNumerals[1]{%
2124 % NUMERALS
2125   \if 1#1\else
2126    \if 2#1\else
2127     \if 3#1\else
2128      \if 4#1\else
2129       \if 5#1\else
2130        \if 6#1\else
2131         \if 7#1\else
2132          \if 8#1\else
2133           \if 9#1\else
2134            \if 0#1\else
2135             \if 0\PunctuationTreatment%
```

```latex
2136                 \@removePunctuation{#1}%
2137              \else
2138                \@rotatePunctuation{#1}%
2139            \fi
2140          \fi
2141         \fi
2142        \fi
2143       \fi
2144      \fi
2145     \fi
2146    \fi
2147   \fi
2148  \fi
2149 \fi
2150 }
2151
2152 \newcommand\@rotatePunctuation[1]{%
2153 % PUNCTUATION
2154   \if ;#1;\else
2155    \if :#1:\else
2156     \if '#1'\else
2157      \if "#1"\else
2158       \if ,#1,\else
2159        \if .#1.\else
2160         \if ?#1?\else
2161          \if '#1'\else
2162           \if !#1!\else
2163            \if 0\SymbolTreatment%
2164               \@removeDirectSymbols{#1}%
2165            \else
2166               \@rotateDirectSymbols{#1}%
2167            \fi
2168           \fi
2169          \fi
2170         \fi
2171        \fi
2172       \fi
2173      \fi
2174     \fi
2175    \fi
2176   \fi
2177 }
2178
2179 \newcommand\@removePunctuation[1]{%
2180 % PUNCTUATION
2181   \if ;#1\else
2182    \if :#1\else
2183     \if '#1\else
2184      \if "#1\else
2185       \if ,#1\else
```

73

```
2186        \if .#1\else
2187         \if ?#1\else
2188          \if `#1\else
2189           \if !#1\else
2190            \if 0\SymbolTreatment%
2191              \@removeDirectSymbols{#1}%
2192            \else
2193              \@rotateDirectSymbols{#1}%
2194            \fi
2195           \fi
2196          \fi
2197         \fi
2198        \fi
2199       \fi
2200      \fi
2201     \fi
2202    \fi
2203   \fi
2204 }
2205
2206 \newcommand\@rotateDirectSymbols[1]{%
2207 % DIRECT SYMBOLS
2208   \if /#1/\else
2209    \if @#1@\else
2210     \if *#1*\else
2211      \if (#1(\else
2212       \if )#1)\else
2213        \if -#1-\else
2214         \if _#1_\else
2215          \if =#1=\else
2216           \if +#1+\else
2217            \if [#1[\else
2218             \if ]#1]\else
2219              \if ^#1^\else% <--FOR SUPERSCRIPTS, NOT \^
2220               \if <#1<\else
2221                \if >#1>\else
2222                 \if |#1|\else
2223                  \if &#1&\else
2224                   \@rotateUndecipherable{#1}%
2225                  \fi
2226                 \fi
2227                \fi
2228               \fi
2229              \fi
2230             \fi
2231            \fi
2232           \fi
2233          \fi
2234         \fi
2235        \fi
```

```
2236        \fi
2237       \fi
2238      \fi
2239     \fi
2240    \fi
2241 }
2242
2243 \newcommand\@removeDirectSymbols[1]{%
2244 % DIRECT SYMBOLS
2245   \if /#1\else
2246    \if @#1\else
2247     \if *#1\else
2248      \if (#1\else
2249       \if )#1\else
2250        \if -#1\else
2251         \if _#1\else
2252          \if =#1\else
2253           \if +#1\else
2254            \if [#1\else
2255             \if ]#1\else
2256              \if ^#1\else% <--FOR SUPERSCRIPTS, NOT \^
2257               \if <#1\else
2258                \if >#1\else
2259                 \if |#1\else
2260                  \if &#1\else
2261                   \@rotateUndecipherable{#1}%
2262                  \fi
2263                 \fi
2264                \fi
2265               \fi
2266              \fi
2267             \fi
2268            \fi
2269           \fi
2270          \fi
2271         \fi
2272        \fi
2273       \fi
2274      \fi
2275     \fi
2276    \fi
2277   \fi
2278 }
2279
2280 \newcommand\@rotateUndecipherable[1]{%
2281 % REPLACE UNDECIPHERABLE SYMBOL WITH A TOKEN CHARACTER (DEFAULT .)
2282   \expandafter\@gobble#1\undecipherable%
2283 % DONE... CLOSE UP SHOP
2284 }
```

```
2285 \catcode`\&=4
2286 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2287 ⟨/package⟩
```