relenc.dtx

Lars Hellström*

1999/01/23

1 Implementation

1.1 Initial stuff

The obligatory header code for a LaTeX 2$_\varepsilon$ package.

```
1 ⟨*package⟩
2 \NeedsTeXFormat{LaTeX2e}[1994/06/01]
3 \ProvidesPackage{relenc}[1999/01/23]
```

At the moment, there are no options for the relenc package.

1.2 Variables and similar things

`\RE@temptoks` `\RE@temptoks` is used by the basic search mechaism to store the part of the search path that has not yet been used.

```
4 \newtoks\RE@temptoks
```

`\RE@garbage` The `\RE@garbage` macro is used as a target for some assignments that really shouldn't be made, but where a mechanism for detecting this would not be worth the cost. It is also used as a temporary storage by some definition commands.

`\RE@first@search@item` During a search, the `\RE@first@search@item` macro will expand the first item in the search path. This is needed for the define first mechanism to work.

`\RE@define@first` The `\RE@define@first` macro works like a switch. If it is `\RE@active@define@first` then the define first mechanism is active: If the search does not find anything on the first try but finds something later then the first command in the search path gets `\let` to whatever the search finds. If `\RE@define@first` is `\@gobbletwo` then the define first mechanism is inactive and nothing happens. This construction allows for other possibilities as well.

The initial setting is `\@gobbletwo`, i.e., off.

```
5 \let\RE@define@first\@gobbletwo
```

*E-mail: Lars.Hellstrom@math.umu.se

1.3 Search mechanisms for variable commands

Variable commands are intended to be used as definitions of encoding-specific commands or compositions of such. Thus variable commands are actually instances of some user level command, which is not the same TEX control sequence as the variable command itself.

A variable command is defined to be a parameterless macro whose expansion consists of two tokens. The first is `\RE@text@variable` if the command is not a composition and it is `\RE@text@comp@variable` if the command is a composition. The second token is the user level command itself; this command will not be expanded, its name is merely used as a word-stem from which names of possible variants are formed. Thus if the definition of the user level command `\foo` in the `T1R` encoding (which would be stored in the control sequence `\T1R\foo`) would be a variable command, that definition would be

```
\RE@text@variable\foo
```

1.3.1 Identifying the search path and initiating the search

`\RE@text@variable`
`\RE@text@comp@variable`
`\RE@spath@unavailable`
`\RE@spath@available`

The first thing `\RE@text@variable` must do is to find the search path to use. It might be either a family-specific search path, in which case it is stored in the macro `\`⟨*encoding*⟩`/`⟨*family*⟩`-path`, or the general search path for the entire encoding, in which case it is stored in the macro `\`⟨*encoding*⟩`-path`. If the family-specific search path exists, it is chosen. If neither exists, this is an error which is reported.

The second thing `\RE@text@variable` does is setting up the search along the search path. This is by expanding to

`\RE@first@read@spath` ⟨*search path*⟩ `\RE@read@spath`

which starts the next phase of the search.

```
\def\RE@text@variable{%
  \expandafter\ifx \csname\cf@encoding/\f@family-path\endcsname \relax
    \expandafter\ifx \csname\cf@encoding-path\endcsname \relax
      \RE@spath@unavailable
    \else
      \RE@spath@available\RE@first@read@spath\RE@read@spath
    \fi
  \else
    \expandafter\expandafter \expandafter\RE@first@read@spath
       \csname\cf@encoding/\f@family-path\expandafter\endcsname
       \expandafter\RE@read@spath
  \fi
}
```

Hackers may find a few things in the definition a little strange. To begin with, I use `\cf@encoding` for the name of the current encoding, although [3] says that `\f@encoding` is defined as precisely that. I too find this a little odd, but the corresponding routines in the LATEX 2_ε kernel use `\cf@encoding` (see [1]) so I suppose that is safer, for some reason. I believe the only time at which `\cf@encoding` and

`\f@encoding` are different is during a font change, but I might be mistaken about that.

The usage of `\RE@spath@unavailable` and `\RE@spath@available` might also be confusing, but the alternative to this is heavily nested `\expandafter` sequences, and those things are not easy to read. Besides, `\RE@spath@unavailable` must do something with the initial command token anyway, because `\RE@text@variable` does not move it.

`\RE@spath@available` have two real arguments: The initial search command header (which can be `\RE@first@read@spath` or `\RE@first@comp@read@spath`) and the initial search command tail (which can be `\RE@read@spath` or `\RE@comp@read@spath`). The other arguments are "dummys" used to remove unvanted tokens.

All arguments of `\RE@spath@unavailable` are such dummys.

```
\def\RE@spath@available#1#2#3\fi#4\fi{\fi\fi
   \expandafter\expandafter \expandafter#1%
      \csname\cf@encoding-path\endcsname#2%
}
\def\RE@spath@unavailable#1\fi#2\fi#3{%
   \fi\fi
   \PackageError{relenc}{%
      There is no search path for relaxed encoding \cf@encoding%
   }\@eha
}
```

Example error message:

```
! Package relenc Error: There is no search path for relaxed encoding XX.

See the relenc package documentation for explanation.
Type H <return> for immediate help
 ...

I.100


? h
Your command was ignored.
Type  I <command> <return>  to replace it with another command,
or  <return>  to continue without it.
```

The reason that `\RE@text@comp@variable` and `\RE@text@variable` exist as separate macros is that there are separate commands `\RE@first@comp@read@spath` and `\RE@first@read@spath` at the next stage of the search process.

```
\def\RE@text@comp@variable{%
  \expandafter\ifx \csname\cf@encoding/\f@family-path\endcsname \relax
    \expandafter\ifx \csname\cf@encoding-path\endcsname \relax
      \RE@spath@unavailable
```

```
    \else
      \RE@spath@available\RE@first@comp@read@spath\RE@comp@read@spath
    \fi
  \else
    \expandafter\expandafter \expandafter\RE@first@comp@read@spath
       \csname\cf@encoding/\f@family-path\expandafter\endcsname
       \expandafter\RE@comp@read@spath
  \fi
}
```

1.3.2 Performing the search

Between the steps of the second stage of the search, the initial font-dependent command (`\foo` say) will have expanded to

⟨*search header*⟩ ⟨*remaining search path*⟩ ⟨*search tail*⟩ `\foo`

The ⟨*search header*⟩ can be one of `\RE@read@spath`, `\RE@first@read@spath`, `\RE@comp@read@spath`, and `\RE@first@comp@read@spath`. The ⟨*search tail*⟩ can be either `\RE@read@spath` or `\RE@comp@read@spath`.

The ⟨*search path*⟩ consists mainly of a sequence of blocks that looks as follows

`{` ⟨*character tokens, macros*⟩ `}`

The macros should also expand to character tokens. At each step, the first of these blocks is extracted as the `#1` argument of the search header. The remaining blocks are put in the `#2` argument and if no definition is found then these will be the ⟨*remaining search path*⟩ to use when setting up for the next step. For the moment, it is put away in the token list register `\RE@temptoks`.

Argument `#1` is first checked for whether the control sequence formed by expanding

`\csname` `#1` `\string` `#3` `\endcsname`

(`#3` is the control sequence named `\foo` above) exists. If it does, it is assumed to be the definition of `\foo` for the current font, otherwise everything is set up for the next step using the ⟨*remaining search path*⟩ as search path.

There is no check to see if `#2` is empty, so if the entire search path consisted of blocks like the one above, using a font-dependent command for which no definition has been given would cause an infinite loop. There is however a hack which prevents this: When a search path is set using the proper command, the control sequence `\RE@var@not@found` is appended to the search path. Primarily, this macro is a hack that prevents the search loop from going haywire, but it also has the advantage of stopping the expansion of the search header from stripping off the braces around the last group prematurely (when

`\RE@read@spath` `{a}` `{b}` `{cd}` `\RE@read@spath` `\foo`

is expanded `#1` will be `a` and `#2` will be `{b}{cd}`, but when

```
\RE@read@spath {b} {cd} \RE@read@spath \foo
```

is expanded `#1` will be `b` and `#2` will be `cd`, so the next time round, `#1` will be `c` and `#2` will be `d`—the last group has been split—but if the last item in the search path is not a group, but a single token, there is no group to split).

`\RE@read@spath`
`\RE@first@read@spath`
`\RE@comp@read@spath`
`\RE@first@comp@read@spath`

Considering their usage only, the difference between `\RE@read@spath` and `\RE@comp@read@spath` is that `\RE@read@spath` is used for actual font-dependent commands while `\RE@comp@read@spath` is used for compositions of a font-dependent command and its argument. In that case, the third argument will not be `\foo` but rather something like `\\foo-a` (a single control sequence, the first backslash is the escape character and the second is part of the actual name); this particular case occurs when the user has written `\foo {a}`).

If, on the other hand, their definitions are looked at, the difference is that while `\RE@read@spath` checks if the expansion of

```
\csname #1 \string #3 \endcsname
```

is defined, `\RE@comp@read@spath` checks if the expansion of

```
\csname #1 \@empty\ string #3 \endcsname
```

is defined. In most cases, these expand to the same thing (since `\@empty` is a parameterless macro which expands to nothing), but sometimes one needs to write a `#1` which gives somewhat different results depending on whether `#3` is a command or a composition. These blocks in the search path can now simply test whether the token after the parameter is `\@empty` or `\string`. The two macros that currently do this are `\RE@var@not@found` and `\RE@convert@nfss`.

The difference between `\RE@read@spath` and `\RE@first@read@spath` is that the latter is used in the first step, while the former is used in all the other steps. This difference exists because of the define first mechanism: If `\RE@first@read@spath` finds a definition in the first step, there is no need to call the mechanism no matter what the value of `\RE@define@first` is, but if `\RE@first@read@spath` does not find a definition then it must store the first block of the search path for a possible future use by `\RE@active@define@first`. `\RE@read@spath`, on the other hand, should never store a search path block, but if it finds a definition then it should call `\RE@define@first` since the define first mechanism might be active.

Quite naturally, `\RE@comp@read@spath` relates to `\RE@first@comp@read@spath` as `\RE@read@spath` to `\RE@first@read@spath` and `\RE@first@comp@read@spath` relates to `\RE@first@read@spath` as `\RE@comp@read@spath` to `\RE@read@spath` [phew!].

```
\def\RE@read@spath#1#2\RE@read@spath#3{%
  \RE@temptoks={#2}%
  \expandafter\ifx \csname#1\string#3\endcsname \relax
    \expandafter\RE@read@spath \the\expandafter\RE@temptoks
       \expandafter\RE@read@spath \expandafter#3%
  \else
    \RE@define@first{#1}{#3}%
```

```
    \csname#1\string#3\expandafter\endcsname
  \fi
}
\def\RE@first@read@spath#1#2\RE@read@spath#3{%
  \RE@temptoks={#2}%
  \expandafter\ifx \csname#1\string#3\endcsname \relax
    \def\RE@first@search@item{#1}%
    \expandafter\RE@read@spath \the\expandafter\RE@temptoks
       \expandafter\RE@read@spath \expandafter#3%
  \else
    \csname#1\string#3\expandafter\endcsname
  \fi
}
\def\RE@comp@read@spath#1#2\RE@comp@read@spath#3{%
  \RE@temptoks={#2}%
  \expandafter\ifx \csname#1\@empty\string#3\endcsname \relax
    \expandafter\RE@comp@read@spath \the\expandafter\RE@temptoks
       \expandafter\RE@comp@read@spath \expandafter#3%
  \else
    \RE@define@first{#1\@empty}{#3}%
    \csname#1\@empty\string#3\expandafter\endcsname
  \fi
}
\def\RE@first@comp@read@spath#1#2\RE@comp@read@spath#3{%
  \RE@temptoks={#2}%
  \expandafter\ifx \csname#1\@empty\string#3\endcsname \relax
    \def\RE@first@search@item{#1\@empty}%
    \expandafter\RE@comp@read@spath \the\expandafter\RE@temptoks
       \expandafter\RE@comp@read@spath \expandafter#3%
  \else
    \csname#1\@empty\string#3\expandafter\endcsname
  \fi
}
```

The definition of `\RE@read@spath` is not quite trivial. It must not leave any `\fi` up ahead, since everything should eventually expand to the actual definition of the searched-for command, and this is very likely to have parameters. There must be no surplus tokens between this command and its arguments; this is why

```
\def\RE@read@spath#1#2\RE@read@spath#3{%
  \RE@temptoks={#2}%
  \expandafter\ifx \csname#1\string#3\endcsname \relax
    \expandafter\RE@read@spath\the\RE@temptoks\RE@read@spath#3%
  \else
    \RE@define@first{#1}{#3}%
    \csname#1\string#3\endcsname
  \fi
}
```

would not do as a definition of `\RE@read@spath`.

A definition that would work is

```
\def\RE@read@spath#1#2\RE@read@spath#3{%
  \expandafter\ifx \csname#1\string#3\endcsname \relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi{%
    \RE@read@spath#2\RE@read@spath#3%
  }{%
    \RE@define@first{#1}{#3}%
    \csname#1\string#3\endcsname
  }%
}
```

but I prefer the "save away" definition since it makes a few additional features possible; the most important perhaps being that it gives a chance to peek in afterwards and see where the search ended (using `\ShowVariantSearchResult`).

1.3.3 Miscellaneous search-related macros

`\RE@active@define@first`
`\RE@again@read@spath`

`\RE@active@define@first` is the macro which performs the actual assignments done by the define first mechanism, if it is active. The macro simply expands to the suitable global assignment; `#1` is the block from the search path that found a definition (with `\@empty` added if the search was for a composition) and `#2` is the command or composition whose definition was searched.

```
\def\RE@active@define@first#1#2{%
   \global\expandafter\let
      \csname\RE@first@search@item\string#2\expandafter\endcsname
      \csname#1\string#2\endcsname
}
```

In some cases, the scan of the search path is resumed after a variant has been found. This is made possible by the fact that the remaining serach path is stored in `\RE@temptoks`, but the define first mechanism adds a complication. It cannot be allowed to copy a variant other than the first in the search path as that could actually change the effect of a command.

The typical case here (the only one possible to define using user level commands of relenc) is that a command has two variants, one of which is composed and one which is not. The composed occurs before the noncomposed and the default definition of the composed is to resume the scan of the search path. If the command is executed for an argument for which there is no composition, an active define first mechanism will hide all composite definitions!

Thus the define first mechanism must be temporarily disabled. One way to do this is to put a dummy value in `\RE@first@search@item`, and this solution is implemented in `\RE@again@read@spath`.

```
\def\RE@again@read@spath{%
   \def\RE@first@search@item##1\expandafter\endcsname{%
```

```
      RE@garbage\expandafter\endcsname
   }%
   \expandafter\RE@read@spath \the\RE@temptoks \RE@read@spath
}
```

\RE@var@not@found
\RE@gobble@readspath
\RE@text@comp@unavail

These macros put a graceful end to a search that has found nothing. When `\RE@var@not@found`, which should be the very last token in every search path (it is automatically added by `\SetEncodingSearchPath` and `\SetFamilySearchPath`), is expanded it should appear in the following context

> `\expandafter` `\ifx` `\csname` `\RE@var@not@found` ⟨*optional* `\@empty`⟩
> `\string` ⟨*command or composition*⟩ `\endcsname`

As `\RE@var@not@found` inserts an additional `\endcsname` and `\fi` when expanded by the `\csname`, the entire `\ifx` is finished before TEX gets to `\RE@gobble@readspath`. There are however a lot of mismatched tokens left over by `\RE@var@not@found`, some of which will be used in composing the error message.

```
\def\RE@var@not@found{relax\endcsname\relax\fi
   \RE@gobble@readspath
}
```

When `\RE@gobble@readspath` is expanded, it is in the context

> `\RE@gobble@readspath` ⟨*optional* `\@empty`⟩ `\string`
> ⟨*command or composition*⟩ `\endcsname` ⟨*some text*⟩ `\fi`

The `\fi` is the `\fi` that originally matched the `\ifx` mensioned above.

The `\@empty` is there iff ⟨*command or composition*⟩ is a composition. If it is, this composition is `\string`ed and the first backslash is thrown away. Then the remaining 'other' tokens are given to `\RE@text@comp@unavail` which will figure out what is the original command and what is its argument. If it is not a composition, the command is given to `\TextSymbolUnavailable`. In any case, everything up to and including the `\fi` disappears.

```
\def\RE@gobble@readspath#1\string#2\endcsname#3\fi{%
   \ifx\@empty#1%
```

If `#2` is a composition then

```
      \expandafter\expandafter \expandafter\RE@text@comp@unavail
        \expandafter\@gobble \string#2\RE@text@comp@unavail
   \else
      \TextSymbolUnavailable{#2}%
   \fi
}
```

As there is a hyphen between the original command and its argument, `\RE@text@comp@unavail` splits the name of the composition at the first hyphen. This is the right thing to do as long as `-` is not used as part of the name of the original command (unlikely, as LATEX does not define `\-` as an encoding-specific command).

```
\def\RE@text@comp@unavail#1-#2\RE@text@comp@unavail{%
   \PackageError{relenc}{%
      The composition of command #1 with #2\MessageBreak is declared %
      in encoding \cf@encoding,\MessageBreak but no definition could %
      be found%
   }\@eha
}
```

Example error text:

```
! Package relenc Error: The composition of command \foo with A
(relenc)                is declared in encoding XXX,
(relenc)                but no definition could be found.

See the relenc package documentation for explanation.
Type H <return> for immediate help
 ...

I.100


? h
Your command was ignored.
Type  I <command> <return>  to replace it with another command,
or  <return>  to continue without it.
```

1.4 Making variable text commands

1.4.1 Miscellaneous support macros

`\RE@empty@is@qmark` Expands to its argument if that is nonempty, otherwise it expands to `?`. This macro is primarily meant to be used inside a `\csname` ... `\endcsname` block. As always, an ingenious fool might break it, but a category 3 `^^M` is extremely unlikely (and hard to type), so it should work for any reasonable argument.

```
\begingroup
   \lccode`\$=13\relax
   \lowercase{%
      \gdef\RE@empty@is@qmark#1{\ifx$#1$?\else#1\fi}%
   }
\endgroup
```

`\RE@font@spec` This macro concatenates the four parameters ⟨*encoding*⟩, ⟨*family*⟩, ⟨*series*⟩, and ⟨*shape*⟩ into relenc's name for this font, i.e., puts slashes between the arguments and replaces every empty argument with a question mark.

```
\def\RE@font@spec#1#2#3#4{%
   \RE@empty@is@qmark{#1}/\RE@empty@is@qmark{#2}/%
   \RE@empty@is@qmark{#3}/\RE@empty@is@qmark{#4}%
}
```

\RE@bsl@string

One thing which complicates the definition of the definition commands (which must work even if they appear in a font definition file) is that LaTeX has the naughty habit of changing the TeX parameter `\escapechar` when it is defing a new font (loading the corresponding font definition file). The `\escapechar` parameter decides which character, if any, should be put in front of a control sequence name which is `\string`ed or written to the log file as part of some `\tracing`... operation. (To see this effect yourself, set `\tracingmacros` positive just before a font change that loads a new font definition file (better reset it to zero after the font change though) and have a look at the log file—throughout a large chunk of logged macro expansions there are no backslashes on the command names!) The reason I don't like this is that (i) it doesn't achieve anything that couldn't just as well be achieved by a combination of `\expandafter` and gobbling of unwanted tokens and (ii) some font definition files are read by LaTeX having `\escapechar` set to its normal value, so you can't trust it either way!

Still, that is how LaTeX does it and this must somehow be coped with. The `\RE@bsl@string` macro acts like `\string` but inserts a backslash in front of control sequence names even when `\string` wouldn't insert anything there. It is used instead of `\string` in all defining commands.

```
\def\RE@bsl@string{%
   \ifnum \escapechar<\z@ \@backslashchar \fi
   \string
}
```

1.4.2 Declaration commands

\DeclareTextVariableCommand
\DeclareTextVariableCommandNoDefault
\DeclareTextVariableSymbol

Two of these are the obvious counterparts of `\DeclareTextCommand` and `\DeclareTextSymbol`. The parameters are the same, and the actual declaring (from the LaTeX 2_ε kernel's point of view) is done by `\DeclareTextCommand`.

`\DeclareTextVariableCommandNoDefault` is a new possibility that exists since a relaxed encoding definition file need not actually specify the definition of any font-dependent command. It can, for example, specify that "Unless otherwise stated, use the definition from encoding ... ".

As the ordinary counterparts of these commands are preamble commands, I have made these preamble commands as well.

```
\newcommand\DeclareTextVariableCommand{\RE@dec@text@varcmd\newcommand}
\@onlypreamble\DeclareTextVariableCommand

\newcommand\DeclareTextVariableCommandNoDefault{%
   \RE@dec@text@varcmd\@gobble
}
\@onlypreamble\DeclareTextVariableCommandNoDefault

\newcommand\DeclareTextVariableSymbol[3]{%
   \RE@dec@text@varcmd\chardef#1{#2}#3\relax
}
\@onlypreamble\DeclareTextVariableSymbol

\def\RE@dec@text@varcmd#1#2#3{%
```

```
    \DeclareTextCommand{#2}{#3}{\RE@text@variable#2}%
    \expandafter#1\csname#3/?/?/?\string#2\endcsname
}
```

\ProvideTextVariableCommand

This is the variable counterpart of \ProvideTextCommand, and it has the same set of parameters. Its definition is a bit more complicated than that of the earlier commands for declaring variable font-dependent commands, partly because \ProvideTextCommand does not tell whether the definition it provided was used, but also becuase this command need not be meaningless even if the font-dependent command it should declare already was declared.

If the font-dependent command is not declared (the \⟨*encoding*⟩\⟨*command*⟩ control sequence is \relax), \ProvideTextCommand is used to declare it and \providecommand is used to give an encoding level definition. If the font-dependent command was declared and variable, then \providecommand is still called, to give an encoding-level definition of the command in case there wasn't already one. Otherwise it must have been declared as a non-variable font-dependent command and the following info message is given:

```
Package relenc Info: You have provided a declaration of \foo in
(relenc)             encoding T1R as a variable command, but it was
(relenc)             already declared as a non-variable command.
(relenc)             Your declaration has been ignored.
```

In this case, the slight problem of the left-over definition remains. This is taken care of by giving it to \providecommand as a definition of \RE@garbage (which was introduced for this kind of tasks).

\ProvideTextCommand is not a preamble-only command, so I have left \ProvideTextVariableCommand free to use anywhere too, although I cannot see why it should be needed after \begin{document}.

```
\newcommand\ProvideTextVariableCommand[2]{%
   \expandafter\ifx \csname#2\string#1\endcsname \relax
      \ProvideTextCommand#1#2{\RE@text@variable#1}%
      \expandafter\providecommand
         \csname#2/?/?/?\string#1\expandafter\endcsname
   \else
      \long\def\RE@garbage{\RE@text@variable#1}%
      \expandafter\ifx \csname#2\string#1\endcsname \RE@garbage
         \expandafter\providecommand
            \csname#2/?/?/?\string#1\expandafter\expandafter
            \expandafter\endcsname
      \else
         \PackageInfo{relenc}{You have provided a declaration of
             \protect#1 in\MessageBreak encoding #2 as a variable
             command, but it was\MessageBreak already declared as a
             non-variable command.\MessageBreak Your declaration has
             been ignored}%
         \expandafter\providecommand
            \csname RE@garbage\expandafter\expandafter
               \expandafter\endcsname
```

```
159       \fi
160    \fi
161 }
```

`\DeclareTextVariableAccent`

This is the variable counterpart of `\DeclareTextAccent`, and again this counterpart has the same parameters as its non-variable model. Also, it is a preamble-only command, just as its model.

```
162 \newcommand{\DeclareTextVariableAccent}[3]{%
163    \DeclareTextCommand{#1}{#2}{\RE@text@variable#1}%
164    \expandafter\newcommand \csname#2/?/?/?\string#1\endcsname
165       {\add@accent{#3}}%
166 }
167 \@onlypreamble\DeclareTextVariableAccent
```

1.4.3 Definition commands

There is an important difference between the declaration commands and the definition commands, and this has to do with when they are executed. A definition command usually appears in a `.fd` file, which is read in inside a group (usually several, but that is irrelevant), but the definitions it makes should be in force after the group has ended. Therefore the definitions must be global, but the `\newcommand` family of LaTeX definition commands only make local definitions. Because of this, I have chosen to primarily use the TeX primitive `\gdef` and its relatives in the definition commands, although in several cases LaTeX-style alternatives are available as well.

`\DefineTextCommandVariant`

The basic defintion command, `\gdef` style. As simple as it can be, but note that the ⟨*parameter text*⟩ does not become one of the arguments of the macro. This has some consequences for when spaces are ignored.

```
168 \newcommand{\DefineTextCommandVariant}[5]{%
169    \expandafter\gdef
170       \csname\RE@font@spec{#2}{#3}{#4}{#5}\RE@bsl@string#1\endcsname
171 }
```

`\DefineTextSymbolVariant`

The basic command for defining a text symbol command.

```
172 \newcommand{\DefineTextSymbolVariant}[6]{%
173    \global\expandafter\chardef
174       \csname\RE@font@spec{#2}{#3}{#4}{#5}\RE@bsl@string#1\endcsname
175       =#6\relax
176 }
```

`\DefineTextAccentVariant`

The basic command for defining an accent command. Very much a special case of `\DefineTextCommandVariant`.

```
177 \newcommand{\DefineTextAccentVariant}[6]{%
178    \expandafter\gdef
179       \csname\RE@font@spec{#2}{#3}{#4}{#5}\RE@bsl@string#1\endcsname
180       {\add@accent{#6}}%
181 }
```

\NewTextCommandVariant
\RenewTextCommandVariant
\ProvideTextCommandVariant
\RE@make@text@cmd@variant
\RE@make@text@cmd@var@x

These commands are the LaTeX-style commands to define variable font-dependent commands. As the LaTeX commands they are built on is not really meant to be used for global definitions, this involves some hacking (using the LaTeX core's private control sequences to do things they are not meant to do, for example). Everything would have been much simpler if the autoload format hook `\aut@global` had been available in all LaTeX formats, but it is not. The purpose of `\RE@make@text@cmd@variant` is to save some `\csname` ... `\endcsname` pairs, the real defining is done by `\RE@make@text@cmd@var@x`.

```
\CheckCommand*{\newcommand}{\@star@or@long\new@command}
\newcommand{\NewTextCommandVariant}
   {\RE@make@text@cmd@variant\new@command}

\CheckCommand*{\renewcommand}{\@star@or@long\renew@command}
\newcommand{\RenewTextCommandVariant}
   {\RE@make@text@cmd@variant\renew@command}

\CheckCommand*{\providecommand}{\@star@or@long\provide@command}
\newcommand{\ProvideTextCommandVariant}
   {\RE@make@text@cmd@variant\provide@command}

\def\RE@make@text@cmd@variant#1#2#3#4#5#6{%
   \expandafter\RE@make@text@cmd@var@x
      \csname\RE@font@spec{#3}{#4}{#5}{#6}\RE@bsl@string#2\endcsname
      {#1}%
}
```

Instead of `\aut@global`, `\l@ngrel@x` is used as a hook. `\l@ngrel@x` is defined to be a macro whose last action is to execute `\global`. As `\l@ngrel@x` appears just before the central `\def`, this makes that definition global in exactly those cases where the definition is made. This requires that the topmost layer of `\newcommand` (and its cousins), the one in which the `*`-processing takes place, is stepped over. Otherwise the definition of `\l@ngrel@x` would be reset.

If the command has an optional argument, however, a simple

```
\def\l@ngrel@x{\global}
```

is not sufficient. In that case the command the user issues is only a preparation that inserts the default optional argument if none is present and issues the real command (the name of the real command is the name of the user command with an extra backslash prepended to it).

The real command will be globally defined, thanks to the `\global` at the end of `\l@ngrel@x`, but the preparation command is at the time of `\l@ngrel@x` only locally defined. This is why `\l@ngrel@x` `\let`s the preparation command to itself globally, this preserves the definition but makes it global. If the command does not have an optional argument, this does no harm as the incorrect definition is overwritten with the correct just after `\l@ngrel@x`.

Finally, `\l@ngrel@x` resets itself to `\relax`, in case something expects it to be `\long` or `\relax`. As `\l@ngrel@x` is not reset if no command definition takes place, there is a slight chance of error in that case, but at least all LaTeX core user commands reset it before using so the hack should not have any side-effects.

```
\def\RE@make@text@cmd@var@x#1#2{%
   \def\l@ngrel@x{%
      \global\let#1#1%
      \let\l@ngrel@x\relax
      \global
   }%
   #2#1%
}
```

To examine sometime: Can `\globaldefs=1` be used instead here? As that makes all assignments global, there is a definite chance it might break something else.

1.5 Making compositions of font-dependent commands

`\RE@if@composed` This is a general test to see if a macro is a command that is set up for having compositions, i.e., if the expansion of the macro begins with `\@text@composite`. This is the same test as is used in `\DeclareTextCompositeCommand`.

The syntax is as follows:

`\RE@if@composed` ⟨*command*⟩ `{`⟨*then-text*⟩`}{`⟨*else-text*⟩`}`

expands to ⟨*then-text*⟩ if ⟨*command*⟩ can have compositions and to ⟨*else-text*⟩ otherwise.

```
\def\RE@if@composed#1{%
   \expandafter\expandafter \expandafter\ifx
      \expandafter\@car #1\relax\relax\@nil \@text@composite
```

If it is composite then ...

```
      \expandafter\@firstoftwo
   \else
      \expandafter\@secondoftwo
   \fi
}
```

1.5.1 Commands for compositions that are variable

The LaTeX 2ε kernel's method of forming the names of compositions is to `\string` the name of the base command and append `-`⟨*argument*⟩ to that, all of which is then `\csname`d. This generates names like

```
\\T1\foo-a
```

for the command `\foo`'s composition with the argument `a` in the `T1` encoding. As the LaTeX 2ε kernel's mechanism for checking for compositions is used for compositions that have variable definitions as well, the names of these commands are formed using the same procedure.

Should then the names of the variants of the composition be generated from names as the above, then these names would be things like

```
\T1R/zcm/m/n\\T1R\foo-a
```

which is a bit longish, even for my taste. Therefore the names of the variants are formed from a base consisting only of the name of the font-dependent command and its argument, which is `\\foo-a` in the above example. This reduces the above to

```
\T1R/zcm/m/n\\foo-a
```

saving two to four characters. This difference in naming does however have some consequences for the usage of definitions from a non-relaxed encoding.

If, for example, `{T1}` is included in the current search path and a definition of the composition `\foo{a}` is looked for, a possible definition of this in the `T1` encoding will not be found! This is because the name tried is `\T1\\foo-a`, but the command is called `\\T1\foo-a`. This is the reason for the macro `\RE@convert@nfss`, which looks ahead to see if a composition or a plain command is currently looked for. Had instead `{\RE@convert@nfss{T1}}` been included, `\T1\\foo-a` would have been changed to `\\T1\foo-a`.

`\DeclareVariableTextComposition`
`\RE@dec@var@text@comp`

This is the variable analog of LaTeX's `\DeclareTextCompositeCommand`, although it is somewhat less as it does not make a default definition for the composition. Thus it is also an analog of `\DeclareTextVariableCommandNoDefault` for compositions.

```
\newcommand{\DeclareVariableTextComposition}[3]{%
   \expandafter\RE@dec@var@text@comp
      \csname\string#1-#3\expandafter\endcsname
      \csname\@backslashchar#2\string#1-#3\endcsname
      {#1}{#2}{#3}%
}
```

As `\DeclareVariableTextComposition` merely make a few combinations of its arguments and then call `\RE@dec@var@text@comp`, the arguments of the latter are worth an explanation. Continuing the above example of `\foo{a}` in the `T1R` encoding, the arguments will be

```
#1 is \\foo-a   #2 is \\T1R\foo-a
#3 is \foo      #4 is T1R           #5 is a
```

Should the composition already have been declared, but the definition is not variable, that definition is made the encoding level definition and the composition is made variable. If the definition is variable, an info message is given and the definition is not changed.

```
\def\RE@dec@var@text@comp#1#2#3#4#5{%
  \ifx#2\relax
```

If this composition has not been declared then ...

```
     \DeclareTextCompositeCommand{#3}{#4}{#5}%
        {\RE@text@comp@variable#1}%
  \else
     \expandafter\expandafter \expandafter\ifx
        \expandafter\@car#2\@nil
        \RE@text@variable
```

If this composition is variable then ...

```
        \PackageInfo{relenc}{Redundant \protect
           \DeclareVariableTextComposition.\MessageBreak
           The composition of \protect#3 with #5 is\MessageBreak
           already declared as a variable command\MessageBreak
           in encoding #4%
        }%
     \else
        \expandafter\let \csname#4/?/?/?\string#1\endcsname #2
        \def#2{\RE@text@comp@variable#1}%
     \fi
  \fi
}
```

Example message:

```
Package relenc Info: Redundant \DeclareVariableTextComposition.
(relenc)             The composition of \foo with a is
(relenc)             already declared as a variable command
(relenc)             in encoding T1R.
```

\DefineTextCompositionVariant
\DefineTextCompositionVariantCommand
\RE@def@text@comp@var

These are the commands that actually define variants of compositions. The first makes the variant a chardef token, the second makes it a parameterless macro.

The arguments for `\RE@def@text@comp@var` should be the following things: The command that does the defining (`\chardef` or `\gdef`), the base user command, the relevant encoding, family, series, and shape, and finally the argument for the composition. Note that the actual definition (slot number or replacement text, respectively) is not among the arguments of `\RE@def@text@comp@var`.

```
\newcommand\DefineTextCompositionVariant[7]{%
   \global \RE@def@text@comp@var\chardef{#1}{#2}{#3}{#4}{#5}{#6}%
      #7\relax
}
\newcommand\DefineTextCompositionVariantCommand{%
   \RE@def@text@comp@var\gdef
}
\def\RE@def@text@comp@var#1#2#3#4#5#6#7{%
   \expandafter#1%
      \csname
         \RE@font@spec{#3}{#4}{#5}{#6}\@backslashchar
         \RE@bsl@string#1-#6%
      \endcsname
}
```

\DefineTextUncomposedVariant
\RE@def@text@uncmp
\RE@def@text@uncmp@x

`\DefineTextUncomposedVariant` extracts the definition used for arguments that does not have definitions, inserts the given argument, and makes the resulting token sequence the definition of the variant at hand. Doing this requires that the definition of the command that the LaTeX 2_ε kernel sees is taken apart, so it might be worth describing how it is contructed.

If `\foo` is composite in the `T1` encoding, `\T1\foo` is a macro which expands to

`\@text@composite \T1\foo #1 \@empty \@text@composite`
`{` ⟨*noncomposite definition*⟩ `}`

where ⟨*noncomposite definition*⟩ is what is needed here. The stuff before checks if the composition is defined, but in this case things are best if that part is never expanded further that this.

The actual "work" is done by `\RE@def@text@uncmp@x`, as this macro contains the `\gdef` which is the only command in this that TEX's stomach sees; everything else just expands in one way or another (unless the warning is issued, that of course contains some stomach stuff too). `\DefineTextUncomposedVariant` makes two `\csname`s: The name of the macro from which the definition is to be extracted and the name of the macro to define.

`\RE@def@text@uncmp` checks if the macro to extract from really is composed, and if it is then it expands it *one level*, feeding it the ⟨*argument*⟩ as argument. Then `\RE@def@text@uncmp@x` is given this one level expansion, and the `\@text@composite`s and everything between them is thrown away. The ⟨*noncomposite definition*⟩ is made the replacement text of a `\gdef` defining the macro to be defined. If the macro to extract a definition from is not composed however, a warning is given.

As this command refers to a specific encoding's definition of a command, the ⟨*encoding*⟩ argument mustn't be empty.

```
\newcommand\DefineTextUncomposedVariant[6]{%
   \expandafter\RE@def@text@uncmp
      \csname#2\RE@bsl@string#1\expandafter\endcsname
      \csname#2/\RE@empty@is@qmark{#3}/\RE@empty@is@qmark{#4}/%
         \RE@empty@is@qmark{#5}\@backslashchar\RE@bsl@string#1-#6%
      \endcsname
      {#6}{#1}{#2}%
}
\def\RE@def@text@uncmp#1#2#3#4#5{%
   \RE@if@composed#1{%
      \expandafter\RE@def@text@uncmp@x #1{#3}{#2}%
   }{%
      \PackageWarning{relenc}{There are no compositions for %
         \protect#4 in\MessageBreak the #5 encoding. %
         \protect\DefineTextUncomposedVariant\MessageBreak
         makes no sense here%
      }%
   }%
}
\def\RE@def@text@uncmp@x\@text@composite#1\@text@composite#2#3{%
   \gdef#3{#2}%
}
```

Example warning:

```
Package relenc Warning: There are no compositions for \foo in
(relenc)                the T1 encoding. \DefineTextUncomposedVariant
(relenc)                makes no sense here.
```

1.5.2 Commands for variants with compositions

The relenc package offers another way in which a command can be defined relative to both argument and font, and that is to allow variants of font-dependent commands to have compositions. It is pretty similar to the usual making of composite commands in LaTeX.

\DefineTextVariantComposition
\DefineTextVariantCompositionCommand

These commands define the composition and make the variant a command with compositions, if it is not already. The mechanism used for compositions uses \@text@composite, exactly like the compositions made by commands in the LaTeX 2_ε kernel.

\DefineTextVariantComposition makes the final definition a chardef token. \DefineTextVariantCompositionCommand makes the final definition a macro without parameters.

An interesting point about these commands is that a variant need not be defined before a composition of it is made! If this happens then of course technicaly the variant gets defined, but its definition for the case when that composition is not found is simply to set things up for a continued scan of the search path. What makes this possible is that the remaining search path is stored in \RE@temptoks.

```
\newcommand\DefineTextVariantComposition[7]{%
   \RE@def@text@var@comp\chardef{#1}{#2}{#3}{#4}{#5}{#6}#7\relax
}
\newcommand\DefineTextVariantCompositionCommand{%
   \RE@def@text@var@comp\gdef
}
```

\RE@def@text@var@comp
\RE@make@text@comp

These are the macros that do the actual work. The parameters of \RE@def@text@var@comp are the user level command, the encoding, family, series, and shape in question, the argument for the composition, the assignment command that should define the composition, and finally the text that should be put after the control sequnce of the composition to complete the assignment. When the assignment command is \chardef, as is the case when \DefineTextVariantComposition makes the call, this is simply a ⟨*number*⟩—the slot. The extra \relax is to stop TeX from reading ahead too far when looking for the end of the ⟨*number*⟩.

The first thing \RE@def@text@var@comp does is to check if the variant the user wants do define a composition of is defined.

```
\def\RE@def@text@var@comp#1#2#3#4#5#6#7{%
   \expandafter\let \expandafter\RE@garbage
      \csname\RE@font@spec{#3}{#4}{#5}{#6}\RE@bsl@string#2\endcsname
   \ifx \RE@garbage\relax
```

This means the variant has not been defined. The default definition given sets up a continued scan along the search path. The key macro here is \RE@again@read@spath (the definition of which is found on page 7). It temporarily disables the define first mechanism and then it expands to

```
\expandafter \RE@read@spath \the \RE@temptoks \RE@read@spath
```

As the ##1 below has become #1 when \RE@make@text@comp is expanded, this will act as a normal #1 in the definition \RE@make@text@comp expands to.

```
      \expandafter\RE@make@text@comp\csname
         \RE@font@spec{#3}{#4}{#5}{#6}\RE@bsl@string#2%
         \endcsname {\RE@again@read@spath#2{##1}}%
```

Otherwise it must be checked if the variant is composed. If it is, then everything is fine, if it isn't, then the current definition is made the default definition.

```
   \else
      \RE@if@composed\RE@garbage{}{%
         \expandafter\RE@make@text@comp
            \csname
               \RE@font@spec{#3}{#4}{#5}{#6}\RE@bsl@string#2%
            \expandafter\endcsname
            \expandafter{\RE@garbage{##1}}%
      }%
   \fi
```

Then it is finally time to define the composition. This is pretty straight off.

```
   \global\expandafter#1\csname
      \@backslashchar\RE@font@spec{#3}{#4}{#5}{#6}\RE@bsl@string#2-#7%
   \endcsname
}
```

\RE@make@text@comp makes #1 a composed command, with #2 as the default definition. This is exactly what the corresponding command in standard LaTeX does, but for some reason LaTeX defines this command of the fly each time it is needed.

```
\def\RE@make@text@comp#1#2{%
   \gdef#1##1{\@text@composite#1##1\@empty\@text@composite{#2}}%
}
```

1.6 Miscellanenous commands

1.6.1 Search paths

A very important thing about search paths is that they are responsible for ending a search that has found nothing, as the macros which actually perform the search contain no mechanism for this. The easiest way to ensure this is to append the token \RE@var@not@found to every search path set, and this is exactly what the standard commands for setting search paths do. Anyone who does not use these commands for setting a search path must be aware that if a search path does not contain any block that ends the search then some tiny errors are likely to start infinite loops.

\SetEncodingSearchPath
\SetFamilySearchPath
\RE@set@spath
\RE@spath@catcodes

These are all fairly standard. \RE@set@spath does the actual setting of a search path while \SetEncodingSearchPath and \SetFamilySearchPath merely form the name of the macro that will store the search path and temporarily change

some \catcodes to make typing the search path somewhat easier. The \catcodes are changed by \RE@spath@catcodes.

```
\newcommand{\SetEncodingSearchPath}[1]{%
   \begingroup
   \RE@spath@catcodes
   \expandafter\RE@set@spath \csname#1-path\endcsname
}
\newcommand{\SetFamilySearchPath}[2]{%
   \begingroup
   \RE@spath@catcodes
   \expandafter\RE@set@spath \csname#1/#2-path\endcsname
}
\def\RE@set@spath#1#2{%
   \gdef#1{#2\RE@var@not@found}%
   \endgroup
}
\def\RE@spath@catcodes{%
   \catcode`\ =9\relax
   \catcode`\^^I=9\relax
   \catcode`\^^M=9\relax
   \catcode`\@=11\relax
   \catcode`\/=12\relax
   \catcode`\?=12\relax
}
```

\RE@convert@nfss

For some relaxed encodings, one wants the definition of a font-dependent command to be identical to the definition of the command in a non-relaxed encoding (this is the case with T1R, which is designed to be a relaxed version of T1). As there are slight differences in how the names of compositions are generated between the LaTeX 2_ε kernel and relenc, it would not be sufficient to simply include {T1} in the search path of the T1R encoding to make it use T1's definitions of compositions as variants of that composition. The purpose of \RE@convert@nfss is to overcome this by changing the names of the variants of compositions where required. Including \RE@convert@nfss{T1} in the search path of T1R gives the intended effect.

```
\def\RE@convert@nfss#1#2{%
   \ifx\@empty#2%
      \@backslashchar#1\expandafter\expandafter \expandafter\@gobble
   \else
      #1\expandafter#2%
   \fi
}
```

1.6.2 The define first mechanism

\ActivateDefineFirst
\DeactivateDefineFirst

These commands can be used to turn the define first mechanism on and off. I am not sure at the moment whether implementing it was a good idea, but as it exists it might as well stay so it can be evaluated.

It could be worth pointing out that these commands act locally. The define first mechanism, on the other hand, acts globally.

```
\newcommand\ActivateDefineFirst{%
   \let\RE@define@first\RE@active@define@first
}
\newcommand\DeactivateDefineFirst{%
   \let\RE@define@first\@gobbletwo
}
```

1.7 Debugging

\ShowVariantSearchResult

This command offers a way to peek into the search mechanism, and in particular to see where it stopped. This command has a potential to come in real handy, but note that it is the *remaining search path* that is shown. You do not see the block that was actually used, you see all blocks that come after it.

```
\newcommand{\ShowVariantSearchResult}{%
   \immediate\write\sixt@@n{Encoding: \cf@encoding}%
   \immediate\write\sixt@@n{Family:   \f@family}%
   \immediate\write\sixt@@n{Series:   \f@series}%
   \immediate\write\sixt@@n{Shape:    \f@shape}%
   \immediate\write\sixt@@n
      {Remaining search path:\MessageBreak\the\RE@temptoks}%
   \show\RE@first@search@item
}
```

1.8 Fix for a bug in LaTeX

The definition of the control sequence `\@text@composite@x` in LaTeXes released 1998 and earlier has a bug (number 2930 in the LaTeX bugs database). The definition in question is

```
\def\@text@composite@x#1#2{%
   \ifx#1\relax
      \expandafter#2%
   \else
      #1%
   \fi}
```

The problem with this definition is the `\expandafter`. According to [1], it was "added so that less chaos ensues from the misuse of this code with commands that take several arguments", but unfortunately, it was placed in a position where it can do no good. `#2` is in all reasonable cases a sequence of more than one token, hence the `\expandafter` will not expand the following `\else`, but some token in `#2` instead! I suspect that the `\expandafter` was really meant to be placed in front of the `#1`, which actually is just a single token.

As it happens, the above bug can cause some serious problems when the non-composite definition of a composite command happens to be variable, since the above `\expandafter` will then expand a control sequence which should *not* be expanded.

`\@text@composite@x` The following code fixes the above bug if the LaTeX used is one with that bug. This is done as a curtesy to all those LaTeX users which (like myself) are not always updating LaTeX as soon as a new release appears. The definition of `\@text@composite@x` is checked and, if it matches the above, replaced with the fixed definition.

```
\def\RE@garbage#1#2{%
   \ifx#1\relax
      \expandafter#2%
   \else
      #1%
   \fi}
\ifx \@text@composite@x\RE@garbage
   \def\@text@composite@x#1{%
      \ifx #1\relax
        \expandafter\@secondoftwo
      \else
        \expandafter\@firstoftwo
      \fi
      #1%
   }
\fi
\let\RE@garbage\relax
⟨/package⟩
```

References

[1] Johannes Braams, David Carlisle, Alan Jeffrey, Frank Mittelbach, Chris Rowley, Rainer Schöpf: `ltoutenc.dtx` (part of the LaTeX 2_ε base distribution).

[2] Alan Jeffrey, Rowland McDonnell (manual), Sebastian Rahtz, Ulrik Vieth: *The fontinst utility* (v 1.8), `fontinst.dtx`, in CTAN at `ftp://ftp.tex.ac.uk/tex-archive/fonts/utilities/fontinst/`...

[3] LaTeX3 Project Team: *LaTeX 2_ε font selection*, `fntguide.tex` (part of the LaTeX 2_ε base distribution).

[4] Frank Mittelbach [et al. ?]: `encguide.tex`. To appear as part of the LaTeX 2_ε base distribution. Sometime. Or at least, that is the intention.