A new font selection scheme for TEX macro packages (Basic Macros)*

Frank Mittelbach Rainer Schöpf

March 27, 2018

This file is maintained by the LaTeX Project team. Bug reports can be opened (category `latex`) at `https://latex-project.org/bugs.html`.

This file contains the main implementation of the 'low level' font selection commands. See other parts of the LaTeX distribution, or *The LaTeX Companion* for higher level documentation of the LaTeX 'New' Font Selection Scheme.

> **Warning:** The macro documentation is still basically the documentation from the first NFSS release and therefore in some cases probably not completely accurate.

The '2ekernel' code ensures that a `\usepackage{autofss1}` is essentially ignored if a 'full' format is being used that has picture mode already in the format.

Note the `autofss2` loading is currently disabled.

```
1 ⟨2ekernel⟩\expandafter\let\csname ver@autofss1.sty\endcsname\fmtversion
```

1 Preliminary macros

We define a number of macros that will be used later.

`\@nomath` `\@nomath` is used by most macros that will have no effect in math mode. It issues a warning message.

```
2 ⟨*2ekernel⟩
3 \def\@nomath#1{\relax\ifmmode
4    \@font@warning{Command \noexpand#1invalid in math mode}\fi}
```

*This file has version number v3.2a dated 2017/01/10

\no@alphabet@error The macro \no@alphabet@error is called whenever the user requests a math *alphabet* that is not available in the current *version*. In math mode an error message is produced otherwise the command keeps silent. The argument is the name of the control sequence that identifies the math *alphabet*. The \relax at the beginning is necessary to prevent TEX from scanning too far in certain situations.

```
\gdef\no@alphabet@error#1{\relax \ifmmode
    \@latex@error{Math\space alphabet\space identifier\space
          \noexpand#1is\space undefined\space in\space math\space
           version\space `\math@version'}%
        {Your\space requested\space math\space alphabet\space
         is\space undefined\space in\space the\space current\space
          math\space version.^^JCheck\space the\space spelling\space
          or\space use\space the\space \noexpand\SetMathAlphabet\space
          command.}
     \fi}
```

\new@mathgroup \mathgroup We also give a new name to \newfam and \fam to avoid verbal confusion (see the introduction).[1]

```
%\def\new@mathgroup{\alloc@8\mathgroup\chardef\sixt@@n}
\let\mathgroup\fam
%\let\newfam\new@mathgroup
\@onlypreamble\new@mathgroup
```

2 Macros for setting up the tables

\DeclareFontShape The macro \DeclareFontShape takes 6 arguments:

```
\def\DeclareFontShape{\begingroup
```

First we restore the catcodes of all characters used in the syntax.

```
   \nfss@catcodes
```

We use \expandafter \endgroup to restore catcode in case something goes wrong with the argument parsing (suggested by Tim Van Zandt)

\DeclareFontShape

```
   \expandafter\endgroup
   \DeclareFontShape@}
\def\DeclareFontShape@#1#2#3#4#5#6{%
   \expandafter\ifx\csname #1+#2\endcsname\relax
     \@latex@error{Font family `#1+#2' unknown}\@eha
   \else
     \expandafter
       \xdef\csname#1/#2/#3/#4\endcsname{\expandafter\noexpand
                                    \csname #5\endcsname}%
     \def\reserved@a{#6}%
     \global
```

[1] For the same reason it seems advisable to \let\fam and \newfam equal to \relax, but this is commented out to retain compatibility to existing style files.

```
     \expandafter\let\csname#5\expandafter\endcsname
        \ifx\reserved@a\@empty
          \@empty
        \else
          \reserved@a
        \fi
   \fi
  }
```

`\DeclareFixedFont` Define a direct font switch that avoids all overhead.

```
\def\DeclareFixedFont#1#2#3#4#5#6{%
   \begingroup
      \math@fontsfalse
      \every@math@size{}%
      \fontsize{#6}\z@
      \usefont{#2}{#3}{#4}{#5}%
      \global\expandafter\let\expandafter#1\the\font
   \endgroup
  }
```

`\do@subst@correction`

```
\def\do@subst@correction{%
       \xdef\subst@correction{%
          \font@name
          \global\expandafter\font
            \csname \curr@fontshape/\f@size\endcsname
            \noexpand\fontname\font
           \relax}%
```

Calling `\subst@correction` after the current group means calling it after we have loaded the substitution font which is done inside a group.

```
       \aftergroup\subst@correction
}
```

`\DeclareFontFamily`

```
\def\DeclareFontFamily#1#2#3{%
```

If we want fast checking for the encoding scheme we can just check for `\T@..` being defined.

```
% \@tempswafalse
% \def\reserved@b{#1}%
% \def\cdp@elt##1##2##3##4{\def\reserved@c{##1}%
%      \ifx\reserved@b\reserved@c \@tempswatrue\fi}%
% \cdp@list
% \if@tempswa
 \@ifundefined{T@#1}%
    {%
     \@latex@error{Encoding scheme  `#1' unknown}\@eha
    }%
    {%
```

Now we have to define the macro \⟨*#1*⟩+⟨*#2*⟩ to contain `#3`. But since most of the time `#3` will be empty we use `\let` in a tricky way rather than a simple `\def` since this will save internal memory. We store the argument `#3` in a temporary macro `\reserved@a`.

```
\def\reserved@a{#3}%
```

We compare `\reserved@a` with `\@empty` If these two are the same we `\let` the 'extra' macro equal to `\@empty` which is not the same a doing a `\let` to `\reserved@a` — the latter would blow one extra memory location rather then reusing the one from `\@empty`.

```
    \global
    \expandafter\let\csname #1+#2\expandafter\endcsname
          \ifx \reserved@a\@empty
            \@empty
          \else \reserved@a
          \fi
  }%
}
```

`\cdp@list` We initialize the code page list to be empty.

```
\let\cdp@list\@empty
\@onlypreamble\cdp@list
```

`\cdp@elt`

```
\let\cdp@elt\relax
\@onlypreamble\cdp@elt
```

`\DeclareFontEncoding`

```
\def\DeclareFontEncoding{%
```

First we start with ignoring all blanks and newlines since every surplus space in the second or third argument will come out in a weird place in the document.

```
  \begingroup
  \nfss@catcodes
  \expandafter\endgroup
  \DeclareFontEncoding@}
\@onlypreamble\DeclareFontEncoding

\def\DeclareFontEncoding@#1#2#3{%
 \expandafter
 \ifx\csname T@#1\endcsname\relax
    \def\cdp@elt{\noexpand\cdp@elt}%
    \xdef\cdp@list{\cdp@list\cdp@elt{#1}%
                   {\default@family}{\default@series}%
                   {\default@shape}}%
```

To support encoding dependent commands (like accents) we initialise the command \⟨*encoding*⟩`-cmd` to be `\@changed@cmd`. (See `ltoutenc.dtx` for details.)

```
    \expandafter\let\csname#1-cmd\endcsname\@changed@cmd
 \else
```

```
    \@font@info{Redeclaring font encoding #1}%
  \fi
  \global\@namedef{T@#1}{#2}%
  \global\@namedef{M@#1}{\default@M#3}%
```

Keep a record of the last encoding being declared:

```
  \xdef\LastDeclaredEncoding{#1}%
  }
\@onlypreamble\DeclareFontEncoding@
```

`\LastDeclaredEncoding` The last encoding being declared by `\DeclareFontEncoding`.

```
\def\LastDeclaredEncoding{}
```

`\DeclareFontSubstitution`

```
\def\DeclareFontSubstitution#1#2#3#4{%
  \expandafter
  \ifx\csname T@#1\endcsname\relax
    \@latex@error{Encoding scheme  `#1' unknown}\@eha
  \else
    \begingroup
```

We loop through the `\cdp@list` and rebuild it anew in `\toks@` thereby replacing the defaults for the encoding in question with the new defaults. It is important to store the encoding to test against expanded in `\reserved@a` since it might just be `\LastDeclaredEncoding` that is passed as `#1`.

```
      \edef\reserved@a{#1}%
      \toks@{}%
      \def\cdp@elt##1##2##3##4{%
         \def\reserved@b{##1}%
         \ifx\reserved@a\reserved@b
```

Here we use the new defaults but we use `##1` (i.e., the encoding name already stored previously) since we know that it is expanded.

```
            \addto@hook\toks@{\cdp@elt{##1}{#2}{#3}{#4}}%
         \else
```

If `\reserved@a` and `\reserved@b` differ then we simply copy from the old list to the new.

```
            \addto@hook\toks@{\cdp@elt{##1}{##2}{##3}{##4}}%
         \fi}%
       \cdp@list
       \xdef\cdp@list{\the\toks@}%
    \endgroup
    \global
    \@namedef{D@#1}{%
          \def\default@family{#2}%
          \def\default@series{#3}%
          \def\default@shape{#4}%
          }%
  \fi
```

```
}
\@onlypreamble\DeclareFontSubstitution

\DeclareFontEncodingDefaults
\def\DeclareFontEncodingDefaults#1#2{%
  \ifx\relax#1\else
    \ifx\default@T\@empty\else
      \@font@info{Overwriting encoding scheme text defaults}%
    \fi
    \gdef\default@T{#1}%
  \fi
  \ifx\relax#2\else
    \ifx\default@M\@empty\else
      \@font@info{Overwriting encoding scheme math defaults}%
    \fi
    \gdef\default@M{#2}%
  \fi
}
\@onlypreamble\DeclareFontEncodingDefaults

\default@T
\default@M
\let\default@T\@empty
\let\default@M\@empty

\DeclarePreloadSizes
\def\DeclarePreloadSizes#1#2#3#4#5{%
 \@ifundefined{T@#1}%
   {\@latex@error{Encoding scheme  `#1' unknown}\@eha}%
   {%
```

Don't know at the moment what this group here does!

```
   \begingroup
```

We define a macro `\reserved@f`[2] that grabs the next *size* and loads the corresponding font. This is done by delimiting `\reserved@f`'s only argument by the token `,` (comma).

```
    \def\reserved@f##1,{%
```

The end of the list will be detected when there are no more elements, i.e. when `\reserved@f`'s argument is empty. The trick used here is explained in Appendix D of the TEXbook: if the argument is empty the `\if` will select the first clause and `\let \reserved@f` equal to `\relax`. (We use the `>` character here since it cannot appear in font file names.)

```
      \if>##1>%
        \let\reserved@f\relax
      \else
```

[2] We cannot use `\@tempa` since it is needed in `\pickup@font`.

Otherwise, we define `\font@name` appropriately and call `\pickup@font` to do the work. Note that the requested `\curr@fontshape` combination must have been defined, or you will get an error. The definition of `\font@name` is carried out globally to be consistent with the rest of the code in this file.

```
            \xdef\font@name{\csname#1/#2/#3/#4/##1\endcsname}%
            \pickup@font
```

Now we forget the name of the font just loaded. More precisely, we set the corresponding control sequence to `\relax`. This means that later on, when the font is first used, the macro `\define@newfont` is called again to execute the 'extra' macro for this font.

```
            \global\expandafter\let\font@name\relax
          \fi
```

Finally we call `\reserved@f` again to process the next *size*. If `\reserved@f` was `\let` equal to `\relax` this will end the macro.

```
          \reserved@f}%
```

We finish with reinserting the list of sizes after the `\reserved@f` macro and appending an empty element so that the end of the list is recognized properly.

```
      \reserved@f#5,,%
    \endgroup
    }%
}
\@onlypreamble\DeclarePreloadSizes
```

`\ifmath@fonts` We need a switch to decide if we have to switch math fonts. For this purpose we provide `\ifmath@fonts` that can be set to true or false by the `\S@...` macros depending on if math fonts are provided for this size or not. The default is of course to switch all fonts.

```
\newif\ifmath@fonts \math@fontstrue
```

`\DeclareMathSizes` `\DeclareMathSizes*` `\DeclareMathSizes` takes the text size, math text size, math script size, and math scriptscript size as arguments and defines the right `\S@...` macro.

```
\def\DeclareMathSizes{%
  \@ifstar{\@DeclareMathSizes\math@fontsfalse}%
          {\@DeclareMathSizes{}}}
\@onlypreamble\DeclareMathSizes
```

`\@DeclareMathSizes` This modification by Michael J. Downes on comp.text.tex on 2002/10/17 allows the user to have settings such as `\DeclareMathSizes{9.5dd}{9.5dd}{7.4dd}{6.6dd}`.

```
⟨/2ekernel⟩
⟨latexrelease⟩\IncludeInRelease{2015/01/01}{\@DeclareMathSizes}%
⟨latexrelease⟩                    {Arbitrary units in \DeclareMathSizes}%
⟨*2ekernel | latexrelease⟩
\def\@DeclareMathSizes #1#2#3#4#5{%
  \@defaultunits\dimen@ #2pt\relax\@nnil
  \if $#3$%
```

```
    \expandafter\let\csname S@\strip@pt\dimen@\endcsname\math@fontsfalse
  \else
    \@defaultunits\dimen@ii #3pt\relax\@nnil
    \@defaultunits\@tempdima #4pt\relax\@nnil
    \@defaultunits\@tempdimb #5pt\relax\@nnil
    \toks@{#1}%
    \expandafter\xdef\csname S@\strip@pt\dimen@\endcsname{%
      \gdef\noexpand\tf@size{\strip@pt\dimen@ii}%
      \gdef\noexpand\sf@size{\strip@pt\@tempdima}%
      \gdef\noexpand\ssf@size{\strip@pt\@tempdimb}%
      \the\toks@
    }%
  \fi
}%
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}{\@DeclareMathSizes}%
⟨latexrelease⟩                  {Arbitrary units in \DeclareMathSizes}%
⟨latexrelease⟩\def\@DeclareMathSizes#1#2#3#4#5{%
⟨latexrelease⟩     \@defaultunits\dimen@#2pt\relax\@nnil
⟨latexrelease⟩     \if$#3$%
⟨latexrelease⟩       \expandafter \let
⟨latexrelease⟩         \csname S@\strip@pt\dimen@\endcsname
⟨latexrelease⟩         \math@fontsfalse
⟨latexrelease⟩     \else
⟨latexrelease⟩       \expandafter \gdef
⟨latexrelease⟩       \csname S@\strip@pt\dimen@\endcsname
⟨latexrelease⟩            {\gdef\tf@size{#3}\gdef\sf@size{#4}%
⟨latexrelease⟩                                \gdef\ssf@size{#5}%
⟨latexrelease⟩             #1%
⟨latexrelease⟩                                }%
⟨latexrelease⟩     \fi}%
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩

\@onlypreamble\@DeclareMathSizes
```

3 Selecting a new font

3.1 Macros for the user

`\fontencoding` `\f@encoding`

As we said in the introduction a font is described by four parameters. We first define macros to specify the wanted *family*, *series*, or *shape*. These are simply recorded in internal macros `\f@family`, `\f@series`, and `\f@shape`, resp. We use `\edef`'s so that the arguments can also be macros.

```
\DeclareRobustCommand\fontencoding[1]{%
    \expandafter\ifx\csname T@#1\endcsname\relax
      \@latex@error{Encoding scheme `#1' unknown}\@eha
    \else
```

```
      \edef\f@encoding{#1}%
      \ifx\cf@encoding\f@encoding
```

If the new encoding is the same as the old encoding we have nothing to do. However, in case we had a sequence of several encoding changes without a `\selectfont` in-between we can save processing by making sure that `\enc@update` is `\relax`.

```
        \let\enc@update\relax
      \else
```

If current and new encoding differ we define the macro `\enc@update` to contain all updates necessary at `\selectfont` time.

```
        \let\enc@update\@@enc@update
      \fi
    \fi
}
```

`\@@enc@update`

```
\def\@@enc@update{%
```

When `\@@enc@update` is executed `\f@encoding` holds the encoding name for the new encoding and `\cf@encoding` the name of the last active encoding.

We start by setting the init command for encoding dependent macros to `\@changed@cmd`.

```
          \expandafter
          \let
            \csname\cf@encoding -cmd\endcsname
            \@changed@cmd
```

Then we turn the one for the new encoding to `\@current@cmd` (see `ltoutenc.dtx` for further explanations).

```
          \expandafter
          \let
            \csname\f@encoding-cmd\endcsname
            \@current@cmd
```

We execute the default settings `\default@T`, followed by the one for the new encoding.

```
        \default@T
        \csname T@\f@encoding\endcsname
```

Finally we change the default substitution values, disable `\enc@update` and make `\f@encoding` officially the current encoding.

```
        \csname D@\f@encoding\endcsname
        \let\enc@update\relax
        \let\cf@encoding\f@encoding
}
```

`\enc@update` The default action in `\selectfont` is to do nothing.

```
\let\enc@update\relax
```

\fontfamily
\f@family
\fontseries
\f@series
\fontshape
\f@shape

```
\DeclareRobustCommand\fontfamily[1]{\edef\f@family{#1}}
\DeclareRobustCommand\fontseries[1]{\edef\f@series{#1}}
\DeclareRobustCommand\fontshape [1]{\edef\f@shape{#1}}
```

Some handy abbreviation if you want to get some particular font in the current size. If also the size should change one has to issue a `\fontsize` command first.

```
\def\usefont#1#2#3#4{\fontencoding{#1}\fontfamily{#2}%
             \fontseries{#3}\fontshape{#4}\selectfont
             \ignorespaces}
```

\linespread

The command `\linespread` changes the current `\baselinestretch` by calling `\set@fontsize`. The values for `\f@size` and `\f@baselineskip` will be left unchanged.

```
\DeclareRobustCommand\linespread[1]
   {\set@fontsize{#1}\f@size\f@baselineskip}
```

\fontsize

We also define a macro that allows to specify a size. In this case, however, we also need the value of `\baselineskip`. As the first argument to `\set@fontsize` we pass the current value of `\baselinestretch`. This will either match the internal value (in which case nothing changes, or it will be an updated value due to a user change of that macro using `\renewcommand`. If we would pass the internal `\f@linespread` such a change would be effectively overwritten by a size change.

```
\DeclareRobustCommand\fontsize[2]
   {\set@fontsize\baselinestretch{#1}{#2}}
```

\f@linespread

This macro holds the current internal value for `\baselinestretch`.

```
\let\f@family\@empty
\let\f@series\@empty
\let\f@shape\@empty
\let\f@size\@empty
\let\f@baselineskip\@empty
\let\f@linespread\@empty
```

\cf@encoding

```
\let\f@encoding\@empty
\let\cf@encoding\@empty
```

\@defaultunits

The function `\@defaultunits` when wrapped around a dimen or skip assignment supplies default units. Usage:

`\@defaultunits\dimen@=#1pt\relax\@nnil`

Note: the `\relax` is *important*. Other units can be substituted for the 'pt' if desired.

We use `\remove@to@nnil` as an auxiliary macros for `\@defaultunits`. It just has to gobble the supplied default unit 'pt' or whatever, if it wasn't used in the assignment.

```
\def\@defaultunits{\afterassignment\remove@to@nnil}
```

\strip@pt
\rem@pt

This macro strips the characters `pt` produced by using `\the` on a dimen register.

```
\begingroup
  \catcode`P=12
  \catcode`T=12
  \lowercase{
    \def\x{\def\rem@pt##1.##2PT{##1\ifnum##2>\z@.##2\fi}}}
  \expandafter\endgroup\x
\def\strip@pt{\expandafter\rem@pt\the}
```

\mathversion
\math@version

`\mathversion` takes the math *version* name as argument, defines `\math@version` appropriately and switches to the font selected forcing a call to `\glb@settings` if the *version* is known to the system.

```
\DeclareRobustCommand\mathversion[1]
         {\@nomath\mathversion
          \expandafter\ifx\csname mv@#1\endcsname\relax
          \@latex@error{Math version `#1' is not defined}\@eha\else
          \edef\math@version{#1}%
```

We need to force a math font setup both now and at the point where we return to the previous math version. Forcing a math font setup can simply be done by setting `\glb@currsize` to an invalid value since this will trigger the setup when the formula starts.

```
          \gdef\glb@currsize{}%
```

When the scope of the current `\mathversion` ends we need to restore the old setup. However this time we need to force it directly at least if we are inside math, otherwise we could wait. Another way to enhance this code here is todo the setting only if the version really has changed after all. This might be interesting in case of `amstext` and `boldsymbol`.

```
          \aftergroup\glb@settings
          \fi}
```

If TEX would support a hook just before the end of a formula (opposite of `\everymath` so to speak) the implementation of the algorithm would be much simpler because in that case we would set up the correct math fonts at this point without having to worry about incorrect settings due to nesting. The same would be true if in LATEX the use of `$` (as the primitive TEX command) would be impossible and instead only a higher-level interface would be available. Note that this does not mean that a `$` couldn't be the short-hand for starting and stopping that higher-level interface, it only means that the direct TEX function must be hidden.

Anyway, since we don't have this and won't have it in LATEX 2_ε we need to implement it in a somewhat slower way.

We test for the current math font setup on entry of a formula, i.e., on the hooks `\everymath` and `\everydisplay`. But since these hooks may contain user data we provide ourselves with an internal version of these hooks which stays frozen.

\frozen@everymath
\frozen@everydisplay

New internal names for `\everymath` and `\everydisplay`.

```
\let\frozen@everymath\everymath
\let\frozen@everydisplay\everydisplay
```

\everymath
\everydisplay

Now we provide now user hooks that will be called in the frozen internals.

```
\newtoks\everymath
\newtoks\everydisplay
```

\frozen@everymath

Now we define the behaviour of the frozen hooks: first check the math setup then call the user hook.

```
\frozen@everymath = {\check@mathfonts
                     \the\everymath}
```

\frozen@everydisplay

Ditto for the display hook.

```
\frozen@everydisplay = {\check@mathfonts
                        \the\everydisplay}
```

\curr@math@size

This holds locally the current math size.

```
\let\curr@math@size\@empty
```

3.2 Macros for loading fonts

\pickup@font

The macro \pickup@font which is used in \selectfont is very simple: if the font name is undefined (i.e. not known yet) it calls \define@newfont to load it.

```
\def\pickup@font{%
    \expandafter \ifx \font@name \relax
       \define@newfont
    \fi}
```

\split@name

\pickup@font assumes that \font@name is set but it is sometimes called when \f@family, \f@series, \f@shape, or \f@size may have the wrong settings (see, e.g., the definition of \getanddefine@fonts). Therefore we need a macro to extract font *family*, *series*, *shape*, and *size* from the font name. To this end we define \split@name which takes the font name as a list of characters of \catcode 12 (without the backslash at the beginning) delimited by the special control sequence \@nil. This is not very complicated: we first ensure that / has the right \catcode

```
{\catcode`\/=12
```

and define \split@name so that it will define our private \f@encoding, \f@family, \f@series, \f@shape, and \f@size macros.

```
\gdef\split@name#1/#2/#3/#4/#5\@nil{\def\f@encoding{#1}%
                                    \def\f@family{#2}%
                                    \def\f@series{#3}%
                                    \def\f@shape{#4}%
                                    \def\f@size{#5}}}
```

\curr@fontshape

Abbreviation which may get removed again for speed.

```
\def\curr@fontshape{\f@encoding/\f@family/\f@series/\f@shape}
```

\define@newfont

Now we can tackle the problem of defining a new font.

```
\def\define@newfont{%
```

We have already mentioned that the token list that \split@name will get as argument must not start with a backslash. To reach this goal we will set the \escapechar to −1 so that the \string primitive will not generate an escape character. To keep this change local we open a group. We use \begingroup for this purpose since \define@newfont might be called in math mode, and an empty \bgroup... \egroup would add an empty Ord atom to the math list and thus affect the spacing.

Also locally redefine \typeout so that 'No file ...fd' Warnings become Font Info message just sent to the log file.

```
\begingroup
  \let\typeout\@font@info
  \escapechar\m@ne
```

Then we extract *encoding scheme*, *family*, *series*, *shape*, and *size* from the font name. Note the four \expandafter's so that \font@name is expanded first, then \string, and finally \split@name.

```
  \expandafter\expandafter\expandafter
     \split@name\expandafter\string\font@name\@nil
```

If the \curr@fontshape combination is not available, (i.e. undefined) we call the macro \wrong@fontshape to take care of this case. Otherwise \extract@font will load the external font for us.

```
% \expandafter\ifx
% \csname\curr@fontshape\endcsname \relax
  \try@load@fontshape % try always
% \fi
  \expandafter\ifx
     \csname\curr@fontshape\endcsname \relax
    \wrong@fontshape\else
```

To allow substitution we call the curr@fontshape macro which usually will expand to \relax but may hold code for substitution (see \subst@fontshape definition).

```
% \csname\curr@fontshape\endcsname
  \extract@font\fi
```

We are nearly finished and must only restore the \escapechar by closing the group.

```
\endgroup}
```

```
\def\try@load@fontshape{%
  \expandafter
  \ifx\csname \f@encoding+\f@family\endcsname\relax
    \@font@info{Try loading font information for
                \f@encoding+\f@family}%
```

We predefine this combination to be \@empty which means that next time we don't try again unnecessary in case we don't find a .fd file. If the file contains a \DeclareFontFamily command than this setting will be overwritten.

```
  \global\expandafter\let
     \csname\f@encoding+\f@family\endcsname\@empty
```

Set the catcodes used in the syntax, but do it only once (this will be restored at the end of the font loading group).

```
\nfss@catcodes
\let\nfss@catcodes\relax
```

For increased portability make the external filename monocase, but look for the (old style) mixed case filename if the first attempt fails.

On any monocase system this means that the file is looked for twice which takes up time and string space, but at least for this release Check for both names to give people time to re-install their private fd files with lowercase names.

```
\edef\reserved@a{%
  \lowercase{%
    \noexpand\InputIfFileExists{\f@encoding\f@family.fd}}}%
\reserved@a\relax
     {\@input@{\f@encoding\f@family.fd}}%
\fi}
```

`\nfss@catcodes` This macro should contain the standard `\catcode` assignments to all characters which are used in the commands found in an `.fd` file and which might have special `\catcode`s in the middle of a document. If necessary, this list can be extended in a package file using a suitable number of `\expandafter`, i.e.,

```
\expandafter\def\expandafter\nfss@catcodes
      \expandafter{\nfss@catcodes <additional settings>}
```

Note, that this macro might get executed several times since it is also called by `\DeclareFontShape`, thus it probably should not be misused as a general purpose hook.

```
\def\nfss@catcodes{%
```

We start by making `@` a letter and ignoring all blanks and newlines.

```
\makeatletter
\catcode`\ 9%
\catcode`\^^I9%
\catcode`\^^M9%
```

Then we set up `\`, `{`, `}`, `#` and `%` in case an `.fd` file is loaded during a verbatim environment.

```
\catcode`\\\z@
\catcode`\{\@ne
\catcode`\}\tw@
\catcode`\#6%
\catcode`\^7%
\catcode`\%14%
```

The we make sure that the important syntax parts have the right `\catcode`.

```
\@makeother\<%
\@makeother\>%
\@makeother\*%
\@makeother\.%
\@makeother\-%
```

```
    \@makeother\/%
    \@makeother\[%
    \@makeother\]%
    \@makeother\‘%
    \@makeother\’%
    \@makeother\"%
}
```

`\DeclareErrorFont` Declare the last resort shape! We assume that in this fontshape there is a 10pt font but it doesn't really matter. We only loose one macro name if the assumption is false. But at least the font should be there!

```
\def\DeclareErrorFont#1#2#3#4#5{%
      \xdef\error@fontshape{%
          \noexpand\expandafter\noexpand\split@name\noexpand\string
          \expandafter\noexpand\csname#1/#2/#3/#4/#5\endcsname
          \noexpand\@nil}%
```

Initialize all those internal variables which may or may not have values in the first seconds of NFSS' bootstraping process. Later on such values will be updated when an encoding is selected, etc.

We definitely don't want to set `\f@encoding`; we can set all the others since if they are left "blank" any selection would grap "error default values" as well. However, this probably should go also.

```
%       \gdef\f@encoding{#1}%
      \gdef\default@family{#2}%
      \gdef\default@series{#3}%
      \gdef\default@shape{#4}%
      \global\let\f@family\default@family
      \global\let\f@series\default@series
      \global\let\f@shape\default@shape
      \gdef\f@size{#5}%
      \gdef\f@baselineskip{#5pt}%
}
\@onlypreamble\DeclareErrorFont
```

`\wrong@fontshape` Before we come to the macro `\extract@font` we have to take care of unknown `\curr@fontshape` combinations. The general strategy is to issue a warning and to try a default *shape*, then a default *series*, and finally a default *family*. If this last one also fails TEX will go into an infinite loop. But if the defaults are set incorrectly one deserves nothing else!

```
⟨/2ekernel⟩
⟨latexrelease⟩\IncludeInRelease{2015/01/01}{\wrong@fontshape}%
⟨latexrelease⟩                    {Font substitution in preamble}%
⟨*2ekernel | latexrelease⟩
\def\wrong@fontshape{%
    \csname D@\f@encoding\endcsname   % install defaults if in math
```

We remember the wanted `\curr@fontshape` combination which we will need in a moment.

```
    \edef\reserved@a{\csname\curr@fontshape\endcsname}%
  \ifx\last@fontshape\reserved@a
     \errmessage{Corrupted NFSS tables}%
     \error@fontshape
  \else
```

Then we warn the user about the mess and set the shape to its default.

```
    \let\f@shape\default@shape
```

If the combination is not known, try the default *series*.

```
    \expandafter\ifx\csname\curr@fontshape\endcsname\relax
       \let\f@series\default@series
```

If this is still undefined, try the default *family*. Otherwise give up. We never try to change the encoding scheme!

```
        \expandafter
          \ifx\csname\curr@fontshape\endcsname\relax
           \let\f@family\default@family
```

If we change the font family and we are in the preamble then the corresponding `.fd` file may not been loaded yet. Therefore we try this now. Otherwise equating the requested font shape with the finally selected fontshape below will fail and can result in "NFSS tables corruped". After begin document that will not happen as all `.fd` files involved in substituation are loaded at `\begin{document}`.

```
           \begingroup
              \try@load@fontshape
           \endgroup
        \fi \fi
  \fi
```

At this point a valid `\curr@fontshape` combination must have been found. We inform the user about this fact.

The `\expandafter\string` here stops TEX adding the space that it usually puts after command names in messages. The similar construction with `\@undefined` just produces 'undefined', but saves a few tokens.

`\@wrong@font@char` is locally redefined in `\UseTextSymbol` from its normal (empty) definition, to report the symbol generating the font switch.

```
     \@font@warning{Font shape `\expandafter\string\reserved@a'
                      \expandafter\@gobble\string\@undefined\MessageBreak
                    using `\curr@fontshape' instead\@wrong@font@char}%
    \global\let\last@fontshape\reserved@a
```

We change `\@defaultsubs` to produce a warning at the end of the document. The macro `\@defaultsubs` is initially `\relax` but gets changed here if some default font substitution happens. It is then executed in `\enddocument`.

```
    \gdef\@defaultsubs{%
      \@font@warning{Some font shapes were not available, defaults
                       substituted.\@gobbletwo}%
```

If we substitute a `\curr@fontshape` combination by the default one we don't want the warning to be printed out whenever this (unknown) combination is used.

Therefore we globally `\let` the macro corresponding to the wanted combination equal to its substitution. This requires the use of four `\expandafter`'s since `\csname`...`\endcsname` has to be expanded before `\reserved@a` (i.e. the requested combination), and this must happen before the `\let` is executed.

```
    \global\expandafter\expandafter\expandafter\let
      \expandafter\reserved@a
          \csname\curr@fontshape\endcsname
```

Now we can redefine `\font@name` accordingly. This *must* be done globally since it might occur in the group opened by `\define@newfont`. If we would this definition were local the closing `\endgroup` there would restore the old meaning of `\font@name` and then switch to the wrong font at the end of `\selectfont` although the correct font was loaded.

```
    \xdef\font@name{%
      \csname\curr@fontshape/\f@size\endcsname}%
```

The last thing this macro does is to call `\pickup@font` again to load the font if it is not defined yet. At this point this code will loop endlessly if the defaults are not well defined.

```
    \pickup@font}
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}{\wrong@fontshape}%
⟨latexrelease⟩                  {Font substituation in preamble}%
⟨latexrelease⟩\def\wrong@fontshape{%
⟨latexrelease⟩    \csname D@\f@encoding\endcsname
⟨latexrelease⟩    \edef\reserved@a{\csname\curr@fontshape\endcsname}%
⟨latexrelease⟩  \ifx\last@fontshape\reserved@a
⟨latexrelease⟩    \errmessage{Corrupted NFSS tables}%
⟨latexrelease⟩    \error@fontshape
⟨latexrelease⟩  \else
⟨latexrelease⟩    \let\f@shape\default@shape
⟨latexrelease⟩    \expandafter\ifx\csname\curr@fontshape\endcsname\relax
⟨latexrelease⟩       \let\f@series\default@series
⟨latexrelease⟩        \expandafter
⟨latexrelease⟩          \ifx\csname\curr@fontshape\endcsname\relax
⟨latexrelease⟩            \let\f@family\default@family
⟨latexrelease⟩        \fi \fi
⟨latexrelease⟩  \fi
⟨latexrelease⟩    \@font@warning{Font shape
⟨latexrelease⟩           `\expandafter\string\reserved@a'
⟨latexrelease⟩           \expandafter\@gobble\string\@undefined
⟨latexrelease⟩           \MessageBreak
⟨latexrelease⟩           using `\curr@fontshape' instead\@wrong@font@char}%
⟨latexrelease⟩    \global\let\last@fontshape\reserved@a
⟨latexrelease⟩    \gdef\@defaultsubs{%
⟨latexrelease⟩      \@font@warning{Some font shapes were not available,
⟨latexrelease⟩                     defaults substituted.\@gobbletwo}}%
⟨latexrelease⟩    \global\expandafter\expandafter\expandafter\let
⟨latexrelease⟩       \expandafter\reserved@a
```

```
433 ⟨latexrelease⟩            \csname\curr@fontshape\endcsname
434 ⟨latexrelease⟩     \xdef\font@name{%
435 ⟨latexrelease⟩       \csname\curr@fontshape/\f@size\endcsname}%
436 ⟨latexrelease⟩     \pickup@font}
437 ⟨latexrelease⟩\EndIncludeInRelease
438 ⟨*2ekernel⟩
```

`\@wrong@font@char` Normally empty but redefined in `\UseTextSymbol` so that the Font shape undefined message can refer to the symbol causing the problem.

```
439 \let\@wrong@font@char\@empty
```

`\@@defaultsubs` See above.

`\@defaultsubs`

```
440 \let\@defaultsubs\relax
```

`\strip@prefix` In `\extract@font` we will need a way to recover the replacement text of a macro. This is done by the primitive `\meaning` together with the macro `\strip@prefix` (for the details see appendix D of the TEXbook, p. 382).

```
441 \def\strip@prefix#1>{}
```

4 Assigning math fonts to *versions*

`\install@mathalphabet` This is just another name for `\gdef` but we can redefine it if necessary later on.

```
442 \let\install@mathalphabet\gdef
```

`\math@fonts`

```
443 \let\math@fonts\@empty
```

`\select@group` `\select@group` has four arguments: the new ⟨*math alphabet identifier*⟩ (a control sequence), the ⟨*math group number*⟩, the extra macro for math mode and the `\curr@fontshape` definition macro name. We first check if we are in math mode.

```
444 %\def\select@group#1#2#3{\relax\ifmmode
```

We do these things locally using `\begingroup` instead of `\bgroup` to avoid the appearance of an empty Ord atom on the math list.

```
445 %  \begingroup
```

We set the math fonts for the *family* in question by calling `\getanddefine@fonts` in the correct environment.

```
446 %     \escapechar\m@ne
447 %    \getanddefine@fonts{\csname c@mv@\math@version\endcsname}#3%
```

We globally select the math fonts...

```
448 %    \globaldefs\@ne  \math@fonts
```

... and close the group to restore `\globaldefs` and `\escapechar`.

```
449 %  \endgroup
```

As long as no *size* or *version* change occurs the ⟨*math alphabet identifier*⟩ should simply switch to the installed *math group* instead of calling `\select@group` unnecessarily. So we globally redefine the first argument (the new ⟨*math alphabet identifier*⟩) to expand into a `\mathgroup` switch and then select this *alphabet*. Note that this redefinition will be overwritten by the next call to a *version* macro. The original code for the end of `\select@group` was

```
\gdef#1{#3\mathgroup #2}#1\fi}
```

i.e. first redefining the ⟨*math alphabet identifier*⟩ and then calling the new definition to switch to the wanted ⟨*math group*⟩. Now we define the ⟨*math alphabet identifier*⟩ as a call to the `\use@mathgroup` command.

```
%  \xdef#1{\noexpand\use@mathgroup\noexpand#2%
%          {\number\csname c@mv@\math@version\endcsname}}%
```

But this is not sufficient, as we learned the hard way. The problem here is that the loading of the fonts that comprise the alphabet identifier `#1`, as well as the necessary math font assignments is deferred until it is used. This is OK so far, but if the fonts are switched within the current formula (which may happen if a sub-formula is a box that contains a math version switch) the font assignments for `#1` are not restored unless `#1` is used again. This is disastrous since TeX sees the wrong fonts at the end of the math formula, when it converts the math list into a horizontal list.

This is taken into account as follows: When a math alphabet identifier is used for the first time in a certain version it modifies the corresponding macro `\mv@`⟨*version*⟩ so that it calls `\getanddefine@fonts` directly in future as well. We use the macro `\extract@alph@from@version` to do this. It takes the math alphabet identifer `#1` and the math version macro as arguments.

```
%     \expandafter\extract@alph@from@version
%        \csname mv@\math@version\expandafter\endcsname
%       \expandafter{\number\csname c@mv@\math@version\endcsname}%
%       #1%
%     \stepcounter{mv@\math@version}%
```

Finally, it is not possible to simply call the new definition since we have an argument (the third argument of `\use@mathgroup` or more exactly the argument od `\math@egroup` if the `margid` option is in force) which would swallow our closing `\fi`. So we use the `\expandafter` technique to remove the `\fi` before the `\use@mathgroup` is expanded.

```
%\expandafter #1\fi}
```

`\extract@alph@from@version` We proceed to the definition of the macro `\extract@alph@from@version`. As stated above, it takes a math alphabet identifier and a math version macro (e.g. `\mv@normal`) as its arguments.

```
\def\extract@alph@from@version#1#2#3{%
```

To extract and replace the definition of math alphabet identifier `#3` in macro `#1` we have to recall how this definition looks like: Somewhere in the replacement text of `#1` there is the sequence

`\install@mathalphabet`⟨*math alphabet identifier*⟩ `#3{%`
⟨*Definitions for* ⟩`#3}`

Hence, the first thing we do is to extract the tokens preceding this definitions, the definition itself, and the tokens following it. To this end we define one auxiliary macro `\reserved@a`.

```
\def\reserved@a##1\install@mathalphabet#3##2##3\@nil{%
```

When `\reserved@a` is expanded, it will have the tokens preceding the definition in question in its first argument (`##1`), the following tokens in its third argument (`##3`), and the replacement text for the math alphabet identifier `#3` in its second argument. (`##2`). This is then recorded for later use in a temporary macro `\reserved@b`.

```
    \def\reserved@b{##2}%
```

Additionally, we define a macro `\reserved@c` to reconstruct the definitions for the math version in question from the tokens that will remain unchanged (`##1` and `##3`) and the yet to build new definitions for the math alphabet identifier `#3`.

```
    \def\reserved@c####1{\gdef#1{##1####1##3}}%
```

Then we execute our auxiliary macro.

```
  \expandafter\reserved@a#1\@nil
```

OK, so now we have to build the new definition for `#3`. To do so, we first extract the interesting parts out of the old one. The old definition looks like:

`\select@group`⟨*math alphabet identifier*⟩
⟨*math group number*⟩⟨*math extra part*⟩
⟨`curr@fontshape` *definition*⟩

So we define a new temporary macro `\reserved@a` that extracts these parts.

```
  \def\reserved@a\select@group#3##1##2\@nil{%
```

This macro can now directly rebuild the math version definition by calling `\reserved@c`:

```
    \reserved@c{%
      \getanddefine@fonts{#2}##2%
      \install@mathalphabet#3{%
        \relax\ifmmode \else \non@alpherr#3\fi
        \use@mathgroup##1{#2}}}%
```

In addtion it defines the alphabet the way it should be used from now on.

```
   \gdef#3{\relax\ifmmode \else \non@alpherr#3\fi
           \use@mathgroup##1{#2}}}%
```

Finally, we only have to call this macro `\reserved@a` on the old definitions recorded in `\reserved@b`:

```
  \expandafter\reserved@a\reserved@b\@nil
  }
```

`\math@bgroup` `\math@egroup` Here are the default definitions for `\math@bgroup` and `\math@egroup`. We use `\bgroup` instead of `\begingroup` to avoid 'leaking out' of style changes. This has the side effect of always producing mathord atoms.

```
\let\math@bgroup\bgroup
\def\math@egroup#1{#1\egroup}
```

`\calculate@math@sizes` Here is the default definition for `\calculate@math@sizes` a more elaborate interface is under testing in mthscale.sty.

```
\gdef\calculate@math@sizes{%
  \@font@info{Calculating\space math\space sizes\space for\space
              size\space <\f@size>}%
  \dimen@\f@size \p@
  \@tempdimb \defaultscriptratio \dimen@
  \dimen@ \defaultscriptscriptratio \dimen@
  \expandafter\xdef\csname S@\f@size\endcsname{%
    \gdef\noexpand\tf@size{\f@size}%
    \gdef\noexpand\sf@size{\strip@pt\@tempdimb}%
    \gdef\noexpand\ssf@size{\strip@pt\dimen@}%
    \noexpand\math@fontstrue}}
```

`\defaultscriptratio` `\defaultscriptscriptratio` The default ratio for math sizes is: 1 to `\defaultscriptratio` to `\defaultscriptscriptratio`. By default this is 1 to .7 to .5.

```
\def\defaultscriptratio{.7}
\def\defaultscriptscriptratio{.5}
```

`\noaccents@` If we don't have a definition for `\noaccents@` we provide a dummy.

```
\ifx\noaccents@\@undefined
  \let\noaccents@\@empty
\fi
```

`\showhyphens` The `\showhyphens` command must be redefined since the version in `plain.tex` uses `\tenrm`. We have also made some further adjustments for its use in LaTeX.

```
⟨/2ekernel⟩
⟨latexrelease⟩\IncludeInRelease{2017/01/01}{\showhyphens}%
⟨latexrelease⟩                    {XeTeX support for \showhyphens}%
⟨*2ekernel | latexrelease⟩
\ifx\XeTeXcharclass\@undefined
```

Version for engines other than XeTeX.

```
\gdef\showhyphens#1{%
  \setbox0\vbox{%
    \color@begingroup
    \everypar{}%
    \parfillskip\z@skip\hsize\maxdimen
    \normalfont
    \pretolerance\m@ne\tolerance\m@ne\hbadness\z@\showboxdepth\z@\ #1%
    \color@endgroup}}
```

```
\else
```

XeTeX version. When using system fonts XeTeX reports consecutive runs of characters as a single item in box logging, which means the standard `\showhyphens`

does not work. This version typesets the text into a narrow box to force hyphenation and then reconstructs a horizontal list with explicit hyphens to generate the display. Note that the `lmr` OpenType font is forced, this works even if the characters are not in the font as hyphenation is attempted due to the width of the space and hyphen character. It may generate spurious Missing Character warnings in the log, these are however suppressed from the terminal output by ensuring that `\tracingonline` is locally zero.

```
\long\def\showhyphens#1{%
  \setbox0\vbox{%
    \usefont{TU}{lmr}{m}{n}%
    \hsize 1sp %
    \hbadness\@M
    \hfuzz\maxdimen
    \tracingonline\z@
    \everypar={}%
    \leftskip\z@skip
    \rightskip\z@skip
    \parfillskip\z@skip
    \hyphenpenalty=-\@M
    \pretolerance\m@ne
    \interlinepenalty\z@
    \clubpenalty\z@
    \widowpenalty\z@
    \brokenpenalty1127 %
    \setbox\z@\hbox{}%
    \noindent
    \hskip\z@skip
    #1%
    \par
```

Note here we stop the loop if made no progress, non-removable items may mean that we can not process the whole list (which would be testable as `\lastnodetype=-1`).

```
     \loop
     \@tempswafalse
     \ifnum\lastnodetype=11\unskip\@tempswatrue\fi
     \ifnum\lastnodetype=12\unkern\@tempswatrue\fi
     \ifnum\lastnodetype=13 %
      \count@\lastpenalty
      \unpenalty\@tempswatrue
    \fi
    \ifnum\lastnodetype=\@ne
     \setbox\tw@\lastbox\@tempswatrue
     \setbox0\hbox{\unhbox\tw@\unskip\unskip\unpenalty
                   \ifnum\count@=1127 \else\ \fi
                   \unhbox0}%
     \count@\z@
    \fi
    \if@tempswa
```

```
    \repeat
  \hbadness\z@
  \hsize\maxdimen
  \showboxdepth\z@
  \tolerance\m@ne
  \hyphenpenalty\z@
  \noindent\unhbox\z@
}}
\fi
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}{\showhyphens}%
⟨latexrelease⟩                  {XeTeX support for \showhyphens}%
⟨latexrelease⟩\gdef\showhyphens#1{%
⟨latexrelease⟩  \setbox0\vbox{%
⟨latexrelease⟩    \color@begingroup
⟨latexrelease⟩    \everypar{}%
⟨latexrelease⟩    \parfillskip\z@skip\hsize\maxdimen
⟨latexrelease⟩    \normalfont
⟨latexrelease⟩    \pretolerance\m@ne\tolerance\m@ne
⟨latexrelease⟩    \hbadness\z@\showboxdepth\z@\ #1%
⟨latexrelease⟩    \color@endgroup}}
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

`\addto@hook` We need a macro to add tokens to a hook.

```
\long\def\addto@hook#1#2{#1\expandafter{\the#1#2}}
```

`\@vpt`

```
\def\@vpt{5}
```

`\@vipt`

```
\def\@vipt{6}
```

`\@viipt`

```
\def\@viipt{7}
```

`\@viiipt`

```
\def\@viiipt{8}
```

`\@ixpt`

```
\def\@ixpt{9}
```

`\@xpt`

```
\def\@xpt{10}
```

`\@xipt`

```
\def\@xipt{10.95}
```

```
\@xiipt
575 \def\@xiipt{12}
\@xivpt
576 \def\@xivpt{14.4}
\@xviipt
577 \def\@xviipt{17.28}
\@xxpt
578 \def\@xxpt{20.74}
\@xxvpt
579 \def\@xxvpt{24.88}
580 ⟨/2ekernel⟩
```