A new font selection scheme for TEX macro packages — the basic macros

Frank Mittelbach
Rainer Schöpf
Johannes Gutenberg Universität Mainz

Abstract

We have implemented a new font selection scheme for TEX and its macro packages. This scheme allows font family, series, shape, and size to be specified independently. Additionally, it is not necessary to preload all math fonts.

Contents

1 Introduction

In traditional typesetting one distinguishes four parameters to describe a font: the font *family* (e.g. computer modern), the font *series* (e.g. roman or sansserif), the font *shape* (e.g. normal or bold), and the font *size*. This distinction is not always unique: take for example the slanted typeface LATEX uses. This can be seen as the sloped *shape* of *series* roman or as the normal *shape* of *series* sloped.

Recently several people have asked how to use such a scheme in LATEX. Unfortunately the current implementation of LATEX's font selection scheme does not allow incorporation of this concept.

When typesetting math formulas, one usually needs many more fonts than for ordinary text. In the TEXbook Donald Knuth says:

> All characters that are typeset in math mode belong to one of sixteen *families of fonts*[1], numbered internally from 0 to 15.

The use of the word *family* in this context is unfortunate; it conflicts with the font families we are talking about. To avoid confusion we will always speak of font families from the typesetter's point of view. For math we speak of *math groups* each connected to three fonts called the `\textfont`, the `\scriptfont`, and the `\scriptscriptfont`. From the user's point of view, math formulas consist of characters coming from specific math *alphabets* (e.g. those selected by `\cal`) and of symbols (e.g. `\sum`) selected by a special control sequence and scattered over a number of fonts.

All fonts that can be used together in *one* math formula form a *version*. *Versions* can only be switched outside math formulas. Standard LATEX provides two *versions*: normal and bold.

2 The User Interface

The commands described in the next subsections are primitives used to build up more powerful interfaces. But they are all user accessible. We used these commands to construct two interfaces for LATEX: one is mimicking the old font selection (e.g. `\bf` is used to switch to the font `cmbx..`), the other one implements an orthogonal font selection scheme (here

[1] Emphasis by DEK

`\bf` means: change the current *shape* and select a new font but leave *family*, *series* and *size* untouched.) Details can be found in the article "The new font family selection — User Interface to standard LaTeX".

2.1 Selection of a new font

Selecting a new font is done in two independent steps. First you have to change the values for *family*, *series*, *shape* and/or *size* and then execute a macro which uses the new values to select the desired font. If you don't use this macro the font will not be changed.

The first step is done with the macros `\family`, `\series`, `\shape` and `\size`. For example, if you want to switch to the 'sansserif' *series* you have to say `\series{sansserif}`. Except for `\size`, all those macros have one argument, namely, the desired *family*, *series* or *shape*, respectively. The macro `\size` is somewhat special because we decided that it would be better to force the user to specify a new *size* and a `\baselineskip` together for this *size*, so the macro has two arguments.

All four macros will silently accept their arguments. Warning messages are generated in the second step when the actual font selection is carried out.

To select a (new) font one has to call the `\selectfont` macro. This macro looks up the current *family*, *series*, *shape* and *size*, possibly changed by one of the above mentioned commands, and switches to this font, provided the selected combination of *family*, *series* and *shape* is known to the system. If it is unknown, a warning will be printed, and up to three new trials are made to find a substitute. This is done by changing to `\default@shape`, then to `\default@series` and as a last resort to `\default@family`. At least this combination must have been defined, otherwise we will find ourselves in an endless loop.[2]

It may still be, however, that the *size* requested is not specified in the table. This will lead to an error, and the font given by `\default@errfont` will be selected. All four defaults are given private names (names containing an `@`) to emphasize that their values should be changed only by the "local wizards".

The selection scheme described above may seem unnecessarily complicated. But consider the following example: you are now reading a sentence typeset in the `cmr10` font, that is *family* 'computer modern', *series* 'roman', *shape* 'normal' and *size* '10'. If we want to switch to 'typewriter italic' we say `\series{typewriter} \shape{italic}` and then `\selectfont`. To avoid the call to `\selectfont` we would have to embed it in the definition of `\series`, etc. But this means that either `cmtt10` or `cmit10` would be unnecessarily selected (and probably loaded).

As mentioned before, these commands are primitive; they should be used to define higher level commands for a special application. For example LaTeX's `\bf` command can be defined as

```
\def\bf{\series{roman}%
        \shape{boldext}%
        \selectfont}
```

to work in the same way as before in LaTeX.[3] As an alternative, the definition might be

```
\def\bf{\shape{boldext}\selectfont}
```

which will change to the 'bold extended' *shape* in the current *family*, *series* and *size*.

2.2 Changing the math *version*

`\mathversion` switches to another math *version*, e.g.,

```
\mathversion{bold}
```

will switch to *version* 'bold' provided that it is defined. This command can be used only outside of math formulas. As an example we give the definitions of Standard LaTeX's `\boldmath` and `\unboldmath` macros in terms of `\mathversion`. For this we must assume that two *versions* 'cmnormal' and 'cmbold' are already defined.[4]

```
\def\boldmath{\@nomath\boldmath
              \mathversion{cmbold}}
\def\unboldmath{\@nomath\unboldmath
                \mathversion{cmnormal}}
```

3 Setting up a new format

3.1 Defining a new *family/series/shape* combination

Assume that you want to define the combination *family* 'computer modern', *series* 'concrete', *shape* 'italic'. In the present case we have to write

```
\new@fontshape{cm}{concrete}{italic}{%
    <5>1ccr5%
```

[2] This can be fixed easily but we are not sure if it's worth the effort. The defaults shouldn't be changed by an ordinary user job, and it's not necessary to provide code to check a format file.

[3] Actually this definition behaves differently when used in math mode, because then no **bold** face is selected. We will see a correct definition later.

[4] The `\@nomath` command used here issues a warning if these commands are used in math mode.

```
<6>lccr6%
<7>lccr7%
<8>ccti10 at 8pt%
<9>ccti10 at 9pt%
<10>ccti10%
<11>ccti10 at 10.95pt%
<12>ccti10 at 12pt%
<14>ccti10 at 14.4pt%
<17>ccti10 at 17.28pt%
<20>ccti10 at 20.74pt%
<25>ccti10 at 24.88pt}
```

The general form of the specification in the fourth argument of `\new@fontshape` is

<⟨*size*⟩>⟨*external font name*⟩

You are totally free in what you write between the <> to denote the size.

If you look closely at the example given above you'll notice that the first three lines (for sizes 5, 6, and 7) seem to be wrong: they start with a 1 and the external font names are incorrect. This is a special feature of the font selection code that allows font substitution. The numbers in front of the external font name mean:

0 No effect. Same as no number at all.

1 Issue a warning that the requested *family/series/shape* combination is not available in this size and use the font given instead.

2 Issue a warning that the requested *family/series* does not contain the requested *shape* and use the font given instead.

Additionally, for every *family/series* combination there exists a so-called 'extra' macro that is used to set parameters, etc. common to all *shapes* and *sizes*, e.g. inhibiting hyphenation for typewriter fonts. Its argument is the internal font name.

```
\extra@def{cm}{typewriter}%
        {hyphenchar#1\m@ne}
```

3.2 Preloading fonts

The macro `\preload@sizes` provides an easy way to specify fonts that should be preloaded when dumping a format file. It is used as follows:

`\preload@sizes`{⟨*family*⟩}{⟨*series*⟩}{⟨*shape*⟩}
{⟨*list of sizes*⟩}

where the elements of ⟨*list of sizes*⟩ are delimited by commas. Note that it makes no difference for your documents whether you preload a font or load it on demand. In the latter case, however, processing documents takes more time.

3.3 Defining math *groups*

To specify fonts for math, other primitive commands are provided. They all have @ characters in their names; i.e. they will not normally be accessible to the user, but will be when making format files or style files.

Math fonts can be divided in two classes: fonts that are accessed via `\mathchardef` and those that are selected *only* via a ⟨*math alphabet identifier*⟩.

As we already mentioned, all math fonts come in *groups* of `\textfont`, `\scriptfont`, and `\scriptscriptfont`. A new math *group* is defined by the command

`\new@mathgroup`⟨*math group number*⟩

⟨*math group number*⟩ is a control sequence that is assigned a number that from now on will denote this *group*. It is also possible to use an explicit number, i.e. a sequence of digits, instead of this control sequence to stand for this *group*. However, the first alternative is generally superior since `\new@mathgroup` always assigns a previously unused number to this control sequence. The second alternative is normally used for *groups* 0, 1, 2, and 3 which have a special meaning to TEX.

To specify the fonts of this *group*, the commands `\define@mathgroup` (for the first class) and `\define@mathalphabet` (for the second class) are available.

Take for example one of the `cmsy..` fonts, i.e. the standard math symbol fonts in the computer modern family. (They also contain the calligraphic alphabet.) This font must be loaded prior to its use because of the `\mathchardef` commands in `plain.tex`. To achieve this we write

```
\define@mathgroup {cmnormal}2%
    {cm}{mathsymbol}{normal}
```

This can be read as: define the *group* number 2 in the 'cmnormal' *version* to consist of fonts with *family* 'cm', *series* 'mathsymbol', and *shape* normal. The actual *sizes* for `\textfont`, `\scriptfont`, and `\scriptscriptfont` will be determined when the *group* is selected.

If you want to access such a math *group* also via a ⟨*math alphabet identifier*⟩ you must define this control sequence to switch to the corresponding internal group ⟨*number*⟩, viz.

`\def`⟨*math alphabet identifier*⟩{%
`\group`⟨*number*⟩}

Returning to our example: to define `\cal` to select the calligraphic alphabet ($\mathcal{A}$, $\mathcal{B}$) in a formula one has to add the definition

```
\def\cal{\group2 }
```

If we want to declare a *group* that is accessed always by a ⟨*math alphabet identifier*⟩ the `\define@mathalphabet` macro should be used. Since the corresponding fonts are not accessed

by \mathchardef commands, there is no need to preload them. Loading can be done by the ⟨*math alphabet identifier*⟩.

The macro \define@mathalphabet is similar to \define@mathgroup. If one uses a macro \sfmath to select sansserif letters in a formula one has to make a declaration like

```
\define@mathalphabet{cmnormal}\sfmath
  ⟨math group number⟩{cm}{sansserif}{normal}
```

Here \sfmath is the new ⟨*math alphabet identifier*⟩. The ⟨*math group number*⟩ that must previously be defined using \new@mathgroup.

The text size in math formulas is always determined to be the *size* of the text outside. The sizes for subscripts, etc., i.e. the script and the scriptscript size, must be specified additionally.

The macro \define@mathsizes is made for this purpose. It takes three arguments: a text size, the corresponding script size and scriptscript size, e.g.

```
\define@mathsizes{10}{7}{5}
```

defines the script and scriptscript sizes for a text set in *size* '10' to be '7' and '5', resp.

When a *size* change occurs, not only the current font must be switched but also all math fonts which can be selected via special symbols. On the other hand, it may be that math fonts are only available or only used in certain *sizes*. For the other *sizes* we do not need to switch the whole set of math fonts.[5] To specify this we provide the command \define@nomathsize that takes only the text size and inhibits math font switching for this *size*.

If there is more than one *version* provided then you better define all *groups* for every *version*. Otherwise switching the *version* (by using \mathversion) will not reset these *groups* properly. E.g., in the LaTeX implementation we therefore have a line

```
\define@mathgroup{cmbold}2%
    {cm}{mathsymbol}{bold}
```

It goes without saying that the *family/series/shape* combination must have been defined previously by a \new@fontshape command.

4 Concept of the implementation

4.1 Handling the font tables

The first problem we had to solve was how to handle such a huge number of fonts. To implement the four dimensional grid of fonts we maintain an association list[6] (i.e. a list of pairs) with elements (size, external font name) for every combination of font *family/series/shape*. We do not redefine the font changing commands: these commands select the correct font by looking into the association list corresponding to the current font *family/series/shape* combination. This association list is hidden in a macro. Its precise form is as follows: For every ⟨*size*⟩ we have a string of the form

<⟨*size*⟩>⟨*external font name*⟩

This strings are simply concatenated to form one long string of characters. In this way all necessary information is available. But this solution would take up far too much of TeX's valuable main memory. Therefore we use a trick, the same trick that is used in plain TeX's \newhelp macro: we enclose the list of characters by \csname...\endcsname making one macro name out of it. This uses up only one token in TeX's main memory (and some string memory but this is comparatively cheap).

As an example take the normal *shape* of *series* roman in the computer modern *family*, i.e. the cmr fonts. We define a macro \cm/roman/normal whose replacement text contains a single token containing all necessary information in its name. This macro itself is undefined.

```
\expandafter\edef
  \csname cm/roman/normal\endcsname{%
    \expandafter\noexpand\csname
     <V>cmr5%
     <VI>cmr6%
     <VII>cmr7%
     <VIII>cmr8%
     <IX>cmr9%
     <X>cmr10%
     <XI>cmr10 at 10.95pt%
     <XII>cmr12%
     <XIV>cmr12 at 14.4pt%
     <XVII>cmr17%
     <XX>cmr17 at 20.736pt%
     <XXV>cmr17 at 24.8832pt%
   \endcsname}
```

The first \expandafter is needed because the macro name consists of / characters and we use \csname...\endcsname to build them into the macro name. We then use \edef so that the second \csname...\endcsname combination is expanded at definition time. Finally we need the \expandafter\noexpand trick to ensure that the resulting (undefined) macro is not expanded.

4.2 Fonts for math

To set up fonts for math one has to set up several assignments of the form

[5] Think of a special *size* used only in titles with no formulas at all.

[6] Lisp hackers note!

⟨*math font assignment*⟩⟨*number*⟩=⟨*font*⟩

where ⟨*math font assignment*⟩ is one of `\textfont`, `\scriptfont`, or `\scriptscriptfont`, ⟨*number*⟩ is a number associated with the particular font *group* (or family in the terminology used by DEK), and ⟨*font*⟩ the internal name of the font to be selected. The assignments have to be changed when the overall size changes or when a new math *version* is requested by the user.

There are several ways to implement this: one can for example build a macro name from the requested *version* and the current size (i.e. `\f@size`). The replacement text would then contain all necessary assignments. In a way the old font selection scheme of LaTeX uses this method by defining macros like `\xpt`, etc.

We decided to use another approach: Since the current size (`\f@size`) is always known we make all necessary assignments by means of one macro to be called for every *version*, viz.

`\getanddefine@fonts`⟨*number*⟩⟨*font shape*⟩,

where ⟨*number*⟩ has the same meaning as before, and ⟨*font shape*⟩ denotes a macro name like `\cm/mathsymbol/normal` which can be used to get the necessary fonts names by appending the desired size. For every text size there exists a script and a scriptscript size. Our method for getting it will be seen in a minute (depending on your speed of reading).

On first sight this seems to be a lot slower than the other method because `\getanddefine@fonts` takes time to put all the information together.[7] But tests have shown that this is not true: we can neglect this extra time.

The above math font assignments must be done for all fonts containing characters accessed via `\mathchardef` because TeX will complain if these fonts are not properly defined when these symbols are used. (The alternative to convert all `\mathchardef`'s into macro calls that test if the font is available seems to be too inefficient but this should be investigated further.)

For math *alphabets* the situation is different. These are selected by means of the corresponding ⟨*math alphabet identifier*⟩ (i.e. `\cal`) so that the fonts can be loaded on demand.[8] Hence in the *version* macros, for these font groups we have lines of the form

```
\def⟨math alphabet identifier⟩{%
   \select@group⟨math alphabet identifier⟩
                ⟨number⟩⟨font shape⟩}%
```

⟨*number*⟩ and ⟨*font shape*⟩ have the same meaning as before; ⟨*math alphabet identifier*⟩ is available to the user to select the math *alphabet*. The actual math font assignments are carried out by the macro `\select@group` which is called only if the user selects the *alphabet* inside a formula. In this way, fonts not used in a certain document are not loaded, thereby saving space and time.

For every math *alphabet*, there must obviously exist at least one *version*. But it is perfectly legal that certain *alphabets* are available only in certain *versions*. Therefore we need a way to warn the user if he selects a *version* of an *alphabet* that does not exist.

If a *math alphabet* does not exist in a certain *version* the corresponding part of the *version* macro will look like

```
\def⟨math alphabet identifier⟩{%
   \no@version@warning⟨version⟩
          ⟨math alphabet identifier⟩}%
```

which leads to a warning message if the *alphabet* is selected in this *version*.

A minute ago we promised to tell how we obtain the script and scriptscript sizes for a given text size. For every size and math *group*, you need a `\textfont`, a `\scriptfont`, and a `\scriptscriptfont`. The math fonts have to be switched for every size change. Since the math *group* assignments have to be in effect when the current math formula ends we make them all global. But then the old assignments must be restored at the end of the current group. This is done by inserting a macro call with the `\aftergroup` primitive. The current text size is always available to this macro in the expansion of `\f@size`. The corresponding script size and scriptscriptsize (specified via `\define@mathsizes`), however, must be recorded somewhere. We use the following scheme for this: for every size *s* we define a macro `\S@`*s* (e.g. for size `XX` we define `\S@XX`) that globally defines two macros `\sf@size` and `\ssf@size` to expand to the corresponding script size and scriptscript size. With the help of these macros the right sizes can be extracted easily. Take for example size '10', with script size '7' and scriptscript size '5'. The corresponding macro looks like

```
\expandafter\def\csname S@10\endcsname
   {\gdef\sf@size{7}\gdef\ssf@size{5}}
```

[7] The old song: Time vs. space!

[8] However, the example of `\cal` is a bad one because the fonts containing this *alphabet* must be preloaded anyway. Those fonts also contain the bulk of math symbols accessed via `\mathchardef`.

4.3 Special considerations

There are two special cases we must take care of. Both have to do with size changes within an alignment. Why is this special? The first problem appears when the size change occurs in the last column of an alignment. The token saved by the `\aftergroup` primitive will be inserted just after the `\cr` has been read. More precisely: after the end of the alignment template. But here only `\noalign` or the end of the alignment is allowed, everything else starts a new column. There is a simple fix for this: in the template of the alignment the hash mark (#) denoting the last column must be wrapped in a group. The same problem shows up if a size change occurs inside a `\noalign`.

5 Preliminary macros

As always we begin by identifying the latest version of this file on the VDU and in the log file.

```
\immediate\write\sixt@@n{File: 'fam.tex'
              \fileversion \space <\filedate>  (FMI and RmS)}
\immediate\write\sixt@@n{English Documentation
                                  <\docdate>  (RmS and FMI)}
```

Following are a number of macros that will be used later.

`\@spaces` We define `\@spaces` to be an abbreviation for five space tokens.

```
\def\@spaces{\space\space\space\space\space}
```

This is also defined in `latex.tex`, but this code cautiously does not assume that any macros are defined elsewhere (except those in `plain.tex`).

`\@gobble` The `\@gobble` macro is used to get rid of its argument.

```
\def\@gobble#1{}
```

`\@empty` The `\@empty` macro expands to nothing and is used to test for empty replacement texts.

```
\def\@empty{}
```

`\@height` `\@depth` `\@width` The `\@height`, `\@depth` and `\@width` macros are made to conserve token memory.

```
\def\@height{height}
\def\@depth{depth}
\def\@width{width}
```

`\font@warning` We need a macro that prints a warning message. We write to output stream 16 which means that the message will appear both in the transcript file and on the terminal.

```
\def\font@warning#1{\immediate \write \sixt@@n {Warning: #1.}}
```

`\@nomath` `\@nomath` is used by all macros that should not be used in math mode.

```
\def\@nomath#1{\relax\ifmmode \font@warning{Don't use \string#1 in
                                          math mode}\fi}
```

`\no@version@warning` The macro `\no@version@warning` is called whenever the user requests a math *alphabet* that is not available in the current *version*. The first argument is the name of the *version* (as a sequence of characters), the second is 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.

```
\def\no@version@warning#1#2{\relax \ifmmode
    \font@warning{No '#1' version for math alphabet identifier
                  \string#2}\fi}
```

\new@mathgroup
\group

We have to redefine one plain TEX macro: We must remove \outer from definition of \newfam so that it can be used inside other macros. We also give a new name to \newfam and \fam to avoid verbal confusion (see the introduction).[9]

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

6 Macros for setting up the tables

\new@fontshape

Since this kind of definition is needed several times we provide a macro \new@fontshape that does the work for us.

```
\def\new@fontshape#1#2#3#4{\expandafter
    \edef\csname#1/#2/#3\endcsname{\expandafter\noexpand
                                   \csname #4\endcsname}}
```

\extra@def

The 'extra' macro is defined as follows.

```
\expandafter\def\csname extra//cm/typewriter\endcsname#1{%
  \hyphenchar#1\m@ne}
```

We provide an abbreviation for this:

```
\def\extra@def#1#2#3{%
  \expandafter\def\csname extra//#1/#2\endcsname##1{#3}}
```

so that the above definition looks like

```
\extra@def{cm}{typewriter}{\hyphenchar#1\m@ne}
```

However, this is inefficient if there is nothing to do (i.e. if the third argument is empty), so we provide a special test for this case. Here is the actual definition:

```
\def\extra@def#1#2#3{%
```

We store the argument #3 in a temporary macro \@tempa. This must have one parameter since #1 is allowed in the third argument of \extra@def (otherwise TEX will not accept the definition).

```
  \def\@tempa##1{#3}%
```

We compare \@tempa with a macro with one argument and empty replacement text, i.e. with \@gobble. If these two are the same, we \let the 'extra' macro equal \@gobble.

```
  \ifx \@tempa\@gobble
    \expandafter\let\csname extra//#1/#2\endcsname\@gobble
```

Otherwise, we build a definition.

```
  \else \expandafter\def\csname extra//#1/#2\endcsname##1{#3}\fi}
```

\preload@sizes

As we already explained, the macro \preload@sizes provides a convenient way to specify fonts to be preloaded. It takes four arguments and its definition is as follows:

```
\def\preload@sizes#1#2#3#4{%
```

We define a macro \next[10] that grabs the next *size* and loads the corresponding font. This is done by delimiting \next's only argument by the token , (comma).

```
  \def\next##1,{%
```

[9] 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.

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

The end of the list will be detected when there are no more elements, i.e. when \next'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 \next equal to \relax. (We use the > character here since it cannot appear in font file names.)

```
\if>##1>
  \let\next\relax
\else
```

Otherwise, we define \font@name appropriately and call \pickup@font to do the work. Note that the requested *family/series/shape* combination must have been defined, or you will get an error.

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

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

```
\next}%
```

We finish by reinserting the list of sizes after the \next macro and appending an empty element so that the end of the list is recognized properly.

```
\next#4,,}
```

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

```
\newif\ifdefine@mathfonts \define@mathfontstrue
```

\define@mathsizes \define@mathsizes takes the text size, script size, and scriptscript size as arguments and defines the right \S@... macro. (\define@mathfontstrue might be omitted if math fonts are to be defined for every size.)

```
\def\define@mathsizes#1#2#3{\expandafter \def
    \csname S@#1\endcsname{\gdef\sf@size{#2}\gdef\ssf@size{#3}%
                           \define@mathfontstrue}}
```

\define@nomathsize \define@nomathsize takes only the text size as argument and defines \S@... to not change math fonts.

```
\def\define@nomathsize#1{\expandafter \let
    \csname S@#1\endcsname \define@mathfontsfalse}
```

7 Selecting a new font

7.1 Macros for the user

\family \series \shape \f@family \f@series \f@shape As we said in the introduction, a font is described by four parameters. We first define macros to specify the desired *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.

```
\def\family#1{\edef\f@family{#1}}
\def\series#1{\edef\f@series{#1}}
\def\shape#1{\edef\f@shape{#1}}
```

\size \f@size \setnew@baselineskip We also define a macro that allows specification of a size. In this case, however, we also need the value of \baselineskip. We cannot set \baselineskip immediately, so it is recorded in the macro \setnew@baselineskip. We use \edef here because the second argument (#2) might be a macro.

```
\def\size#1#2{%
    \edef\f@size{#1}%
    \edef\setnew@baselineskip{\baselineskip #2\relax}}
```

\selectfont
\glb@currsize

The macro \selectfont is called whenever a font change must take place.

```
\def\selectfont{%
```

Its first action is to determine if the new font has the same size as the previous one. Here the macro \glb@currsize holds the current font size. Its expansion text may also be empty which means that we do not know what the current size is. As its name indicates, it is always set globally.

```
  \ifx \glb@currsize \f@size
```

If the *size* is to be changed we must also change \baselineskip and a number of other parameters. This is done by the macro \glb@settings.

```
  \else \glb@settings
```

Since these changes are done globally, we must ensure that the old values are restored at the end of the current group. We use TEX's \aftergroup primitive to call \glb@settings again just after the current group ends. And that's all of special code for a size change.

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

We now generate the internal name of the font by concatenating *family*, *series*, *shape*, and current *size*, with slashes as delimiters between them. This is much more readable than standard LATEX's \twfbf, etc.

```
  \edef\font@name{%
    \csname\f@family/\f@series/\f@shape/\f@size\endcsname}%
```

We call the macro \pickup@font which will load the font if necessary.

```
  \pickup@font
```

Finally, we select the font. This finishes the macro \selectfont.

```
  \font@name}
```

\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.

```
\def\mathversion#1{\expandafter\ifx\csname #1\endscname\relax
        \font@warning{The requested version `#1' is unknown}\else
        \def\math@version{#1}\glb@settings\aftergroup\glb@settings\fi}
```

7.2 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@family, \f@series, \f@shape, and \f@size macros.

```
\gdef\split@name#1/#2/#3/#4\@nil{\def\f@family{#1}%
```

```
\def\f@series{#2}%
\def\f@shape{#3}%
\def\f@size{#4}}
```

`\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. But then we must save `\escapechar`'s current value. We use count register `\count@` for this purpose.

```
\count@\escapechar
\escapechar\m@ne
```

Then we extract *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 *family/series/shape* 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\f@family/\f@series/\f@shape\endcsname \relax
  \wrong@fontshape\else \extract@font\fi
```

We are nearly finished and must only restore the `\escapechar`.

```
\escapechar\count@}
```

`\wrong@fontshape` Before we come to the macro `\extract@font`, we have to take care of unknown *family/series/shape* 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 incorrectly set, one deserves nothing else!

```
\def\wrong@fontshape{%
```

We remember the desired *family/series/shape* combination which we will need in a moment.

```
\edef\@tempa{\csname\f@family/\f@series/\f@shape\endcsname}%
```

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

```
\font@warning{Font/shape `\@tempa' unknown}%
\shape\default@shape
```

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

```
\expandafter\ifx\csname\f@family/\f@series/\f@shape\endcsname\relax
   \series\default@series
```

If this is still undefined, try the default *family*. Otherwise give up.

```
\expandafter\ifx\csname\f@family/\f@series/\f@shape\endcsname\relax
   \family\default@family
\fi \fi
```

At this point a valid *family/series/shape* combination must have been found. We inform the user about this fact.

```
\font@warning{Using `\f@family/\f@series/\f@shape' instead}%
```

If we substitute a *family/series/shape* combination by the default, 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 desired combination equal to its substitution. This requires the use of four \expandafter's since \csname...\endcsname has to be expanded before \@tempa (i.e. the requested combination), and this must happen before the \let is executed.

```
\global\expandafter\expandafter\expandafter\let\expandafter\@tempa
      \csname\f@family/\f@series/\f@shape\endcsname
```

Now we can redefine \font@name accordingly.

```
\edef\font@name{\csname\f@family/\f@series/\f@shape/\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}
```

\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).

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

\extract@font Here it comes: the macro solving all our problems (well, nearly all). What must this macro do? This is explained best with an example. Assume that *family* is 'cm', *series* is 'sansserif', *shape* 'normal', and *size* '12'. Assume further that this combination is defined, i.e. there exists the macro \cm/sansserif/normal. (Otherwise \extract@font doesn't get called.) Its replacement text consists of one (undefined) control sequence looking like

```
\<10>cmss10<12>cmss12<17>cmss17
```

For reasonable styles one usually needs more sizes but this is sufficient to get the flavour. We will define a macro \extract@fontinfo to find the external font name ('cmss12') for us:

```
\def\extract@fontinfo#1<12>#2<#3\@nil{%
    \global\font\cm/sansserif/normal/12#2}
```

so that when it gets called via

```
\expandafter\extract@fontinfo
   \string\<10>cmss10<12>cmss12<17>cmss17\@nil
```

#1 will contain all characters before <12>, #2 will be exactly cmss12, and #3 will be 17>cmss17. The expansion is therefore

```
\global\font\cm/sansserif/normal/12 cmss12
```

which is exactly what we want.

But this is only part of the whole story. It may be that the size requested does not occur in the \cm/sansserif/normal macro. And the simple definition of \extract@fontinfo we gave above does not allow us to specify the font substitution that we explained in 3.1.

Both problems are solved with the same trick: We define \extract@fontinfo as follows:

```
\def\extract@fontinfo#1<12>#2#3<#4\@nil{%
   \global\font\cm/sansserif/normal/12
      \ifcase 0#2#3\relax\or
              #3 \font@warning{Size 12 not available
```

```
                - using '#3' instead}\or
        #3 \font@warning{Family/series/shape not available
                - using '#3' instead}\else
        \default@errfont \errmessage{Font not found}\fi}
```

How does this work? The first difference from the previous definition is that the characters of the external font name are split between parameters #2 and #3, #2 receiving only the first character. If this first character is not a digit, the `\ifcase` will get the 0 and select the first alternative. #2 and #3 are combined again and used as a file name. If #2 is a digit then the expansion of `\ifcase` will combine the 0 and #2 to a number.[11] Cases 1 and 2 select the second and third alternatives that use #3 as the substitution font.

The default case is reserved for a size that cannot be found in the tables. We achieve this by calling `\extract@fontinfo` via

```
\expandafter \extract@fontinfo
   \string\<10>cmss10<12>cmss12<17>cmss17
   <12>3\@nil
```

If the size ('12' in this case) appears in the `\<10>...` macro everything works as explained above, the only difference being that argument #4 of `\extract@fontinfo` additionally gets the `<12>3` tokens. However, if the size is not found, everything up to the final `<12>` is in argument #1, #2 gets 3, and #3 and #4 are empty. Therefore the `\ifcase` will select the default alternative and write an error message.

We have cheated a bit, of course. Normally digits and characters like `/<>` are not allowed as part of control sequences. Additionally the macros are hidden inside other control sequences so that we have to build `\extract@fontinfo` in several steps. Putting everything together we define `\extract@font` as follows.

```
\def\extract@font{%
```

`\@tempa` is made an abbreviation for the head of the definition of `\extract@fontinfo`.

```
   \def\@tempa{\def\extract@fontinfo####1}%
```

Then we define `\@tempb` so that it expands to `<`⟨*size*⟩`>`. We use this slightly complicated construction to ensure that all characters have `\catcode` 12. This is needed for the delimiter matching in macro expansion.

```
   \edef\@tempb{<\expandafter\strip@prefix\meaning\f@size>}%
```

Now we can define `\extract@fontinfo`.

```
   \expandafter\@tempa\@tempb##2##3<##4\@nil{%
```

Remember that `\font@name` expands to the internal font name.

```
      \global\expandafter\font \font@name
```

Here comes the `\ifcase`. For the benefit of the user, the warning messages are a bit more eloquent.

```
         \ifcase0##2##3\relax\or
                ##3
                 \font@warning{Font/shape '\f@family/\f@series/\f@shape'
                          in size \@tempb\space  not available}%
                 \font@warning{Using '##3' instead}\or
                ##3
                 \font@warning{Font/shape '\f@family/\f@series/\f@shape'
                          not available}%
                \font@warning{Using '##3' instead}\else
```

[11] Recall that 01 is a valid ⟨*number*⟩ for TEX.

There are two points to be explained here: `\default@errhelp` is the font to be selected if the requested size is not found in the tables. `\nofont@help` denotes a token register that contains a help message for the user. Its definition is given below.

```
\default@errfont \errhelp\nofont@help
      \errmessage{Font \expandafter
                \string\font@name\space
                not found}%
\fi}%
```

Now we must extract the font information from the *family/series/shape* macro. This is done in two steps: first generate the macro name by `\csname`...`\endcsname` and expand it to get its replacement text. Then use `\string` to convert this text into a sequence of character tokens with `\catcode` 12. We define `\font@info` to contain this sequence followed by <⟨*size*⟩> (which is stored in `\@tempb`).

```
\edef\font@info{\expandafter\expandafter\expandafter\string
   \csname \f@family/\f@series/\f@shape \endcsname\@tempb}%
```

Now we call `\extract@fontinfo`. Note the 3<`\@nil` tokens at the end.

```
\expandafter\extract@fontinfo\font@info 3<\@nil
```

Finally we call the corresponding "extra" macro to finish things.

```
\csname extra//\f@family/\f@series \expandafter
  \endcsname \font@name \relax}
```

The `\relax` at the end needs to be explained. This is inserted to prevent TEX from scanning too far when it is executing the replacement text of the "extra" macro.

`\nofont@help` `\nofont@help` is a token register containing a help message. It is defined using plain TEX's `\newhelp` macro.

```
\newhelp\nofont@help
       {You requested a font/series/shape/size combination that is
        totally^^Junknown. \space I have inserted a special font name
        that will produce^^Jinteresting effects in your output. \space
        There are two cases in which^^Jthis error can occur:^^J\space
        \space 1) You used the \string\size\space macro to select
                  a size that is not available.^^J\space
        \space 2) If you did not do that, go to your local 'wizard'
                  and^^J \@spaces complain fiercely that the font
                  selection tables are corrupted!^^J%
        (And do not worry about the missing escape characters in the
         error^^Jtraceback above!)^^J}
```

8 Assigning math fonts to *versions*

`\define@mathalphabet` We begin with the definition of the macro `\define@mathalphabet` which is built to append definitions specific to a new math *alphabet* to the replacement text of a *version* macro. It takes six arguments: the math *version* name (as a string of characters), a control sequence identifying the new math *alphabet*, the number of the new math *group* (normally a control sequence defined via `\countdef`), and finally three strings of characters denoting font *family*, *series*, and *shape*. If the shape parameter (#6) is empty then the *alphabet* #2 is not available in *version* #1.

```
\def\define@mathalphabet#1#2#3#4#5#6{%
```

The first thing it does is to check if the name of the math *version* is already defined. This is the case if there already exist other math *alphabets* in this *version*. We must of course remember these definitions. To do so we save the contents of the macro in the token register `\toks@`.

```
\expandafter\ifx\csname #1\endcsname\relax
```

If there is no other math *alphabet* in this *version*, we simply store an empty token list in this register.

```
\toks@{}%
```

Otherwise, we generate the control sequence denoting the macro using `\csname`... `\endcsname` and store its replacement text in `\toks@`. Note the three `\expandafter` primitives to achieve this.

```
\else
  \toks@\expandafter\expandafter\expandafter
      {\csname #1\endcsname}%
\fi
```

Depending on the shape parameter (`#6`) we have different things to do. We save the sequence of character tokens in a temporary control sequence.

```
\def\@tempa{#6}%
```

Now we globally redefine the *version*. Since the name of the *version* is given as a sequence of characters we must again build a macro name out of it. We use an `\xdef` so that the definition is expanded first.

```
\expandafter\xdef\csname#1\endcsname
```

This is necessary since we want to insert the contents of token register `\toks@`.

```
{\the\toks@
```

Then we append the new definitions for the *alphabet* `#1`. The `\noexpand` is necessary to insert the ⟨*math alphabet identifier*⟩ without expanding it.

```
\gdef\noexpand#2%
```

We must now catch the case that the shape parameter `#6` saved in `\@tempa` is empty, i.e. that the *alphabet* is not available in this *version*. We simply include a call to the `\no@version@warning` defined earlier.

```
\ifx\@tempa\@empty
   {\noexpand\no@version@warning
    \noexpand\math@version
    \noexpand#2}%
```

Otherwise, we include a call to `\select@group` (see below) with the three arguments ⟨*math alphabet identifier*⟩, ⟨*math group number*⟩, and font *family/series/shape* definition macro.

```
\else
   {\noexpand\select@group
    \noexpand#2#3%
    \expandafter\noexpand\csname #4/#5/#6\endcsname}%
\fi}%
```

Now the macro switching to the *version* `#1` contains a definition for *alphabet* `#2`. Finally we force a call to `\glb@settings` at the next time the fonts change by globally redefining `\glb@currsize`.

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

`\define@mathgroup` `\define@mathgroup` is similar to `\define@mathalphabet`. This macro is never called when processing a document, only during the font definition phase (e.g. by a style file in LaTeX or when dumping a format file). It is used for those math *groups* that are used via `\mathchardef` primitives. Since we don't need a ⟨*math alphabet identifier*⟩ to select those symbols, the macro takes only five arguments: the math *version* name as a sequence of character tokens, the ⟨*math group number*⟩ as a control sequence (it must already be allocated using `\new@mathgroup`) or as a digit (for *groups* 0 to 3, and font *family*, *series*, and *shape* name (as a sequence of character tokens). The first part is therefore completely analogous to the definition of `\define@mathalphabet`.

```
\def\define@mathgroup#1#2#3#4#5{%
```

```
\expandafter\ifx\csname#1\endcsname\relax
          \toks@{}%
    \else
      \toks@\expandafter\expandafter\expandafter
          {\csname#1\endcsname}%
    \fi
```

Since this code is never called by the user there is no need to issue a warning when the ⟨*math group number*⟩ isn't allocated. However, the font tables must be defined consistently![12] Instead of `\select@group` it uses `\getanddefine@fonts` which has only two arguments: ⟨*math group number*⟩ and the font *family/series/shape* combination.

```
\expandafter\xdef\csname#1\endcsname
   {\the\toks@
    \noexpand\getanddefine@fonts#2%
    \expandafter\noexpand\csname #3/#4/#5\endcsname}%
```

The tail is literally the same as in `define@mathalphabet`.

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

`\getanddefine@fonts` `\getanddefine@fonts` has two arguments: the ⟨*math number number*⟩ and the *family/series/shape* name as a control sequence.

```
\def\getanddefine@fonts#1#2{%
```

We append the current `\f@size` to #2 to obtain the font name.[13]

```
\edef\font@name{\csname \string#2/\f@size\endcsname}%
```

Then we call `\pickup@font` to load it if necessary. We remember the internal name as `\textfont@name`.

```
\pickup@font \let\textfont@name\font@name
```

Same game for `\scriptfont` and `\scriptscriptfont`:

```
\edef\font@name{\csname \string#2/\sf@size\endcsname}%
\pickup@font \let\scriptfont@name\font@name
\edef\font@name{\csname \string#2/\ssf@size\endcsname}%
\pickup@font
```

Then we append the new `\textfont...` assignments to the `\math@fonts`.

```
\edef\math@fonts{\math@fonts
            \textfont#1\textfont@name
            \scriptfont#1\scriptfont@name
            \scriptscriptfont#1\font@name}}
```

`\select@group` `\select@group` has three arguments: the new ⟨*math alphabet identifier*⟩ (a control sequence), the ⟨*math group number*⟩, and the *family/series/shape* definition macro name. We first check if we are in math mode.

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

We do these things locally:

```
\bgroup
```

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

```
\let\math@fonts\@empty  \escapechar\m@ne
\getanddefine@fonts#2#3%
```

[12] Terrible harm will come to you if you don't do it right! A crowd of angry users might come to stone you!

[13] One might ask why this expansion does not generate a macro name that starts with an additional \ character. The solution is that `\escapechar` is set to −1 before `\getanddefine@fonts` is called.

We globally select the math fonts...

```
\globaldefs\@ne  \math@fonts
```

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

```
\egroup
```

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

```
\gdef#1{\group #2}#1%
```

If we are not in math mode nothing needs to be done.

```
\fi}
```

\glb@settings The macro \glb@settings globally selects all math fonts for the current size. The first thing it does is to open up a group.

```
\def\glb@settings{\begingroup
```

This is done to keep the following changes local: set the \escapechar to −1 and make \math@fonts to expand to nothing.

```
\escapechar \m@ne
\let\math@fonts\@empty
```

Why do we \let \math@fonts equal to \@empty at this point? When \glb@settings gains control, a size change was requested and all previous font assignments need to be replaced. Therefore the old values of the fonts are no longer needed. For every *group* the new assignments are appended to \math@fonts. Now we set the script size and scriptscript size.

```
\csname S@\f@size\endcsname
```

This also sets the define@mathfonts switch. If it is true, we must switch the math fonts. We execute the macro for the current math *version*. This sets \math@fonts to a list of \textfont... assignments.

```
\ifdefine@mathfonts \csname \math@version \endcsname \fi
```

Then we set \globaldefs to 1 so that all following changes are done globally.

```
\globaldefs\@ne
```

The math font assignments recorded in \math@fonts are executed, \glb@currsize is set to the wanted \f@size, and the \baselineskip parameter is set accordingly by the macro \setnew@baselineskip and then multiplied by \baselinestretch.

```
\math@fonts
\let \glb@currsize \f@size
\setnew@baselineskip
\baselineskip\baselinestretch\baselineskip
```

Then we set the \strutbox and \normalbaselineskip.

```
\setbox\strutbox\hbox{\vrule\@height.7\baselineskip
    \@depth.3\baselineskip \@width\z@}%
\normalbaselineskip\baselineskip
```

The macro ends by closing the group. This restores all parameters changed locally (including \globaldefs!) to their previous values.

```
\endgroup}
```

\baselinestretch In \glb@settings we used \baselinestretch as a factor when assigning a value to \baselineskip. We use 1 as a default (i.e. no stretch).

```
\def\baselinestretch{1}
```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

◇ Frank Mittelbach
Rainer Schöpf
Fachbereich Mathematik
Universität Mainz
Staudinger Weg 9
D-6500 Mainz
Federal Republic of Germany
Bitnet: schoepf@dmznat51