# LaTeX 2ε

LaTeX3 Project Team

1994/01/31

**Abstract**

This file contains a short description of most of the new features of LaTeX 2ε—it makes no claims of completeness or of complete accuracy (but please tell us if you find any errors). It assumes a reasonably high level of familiarity with some parts of the LaTeX 2.09 system.

Only the changes to the core system are documented here. Changes to standard packages (such as `ifthen.sty`), and new packages that may be distributed with LaTeX 2ε are *not* described.

Full documentation of LaTeX 2ε can be found in the book "The LaTeX Companion", by Michel Goossens, Frank Mittelbach and Alexander Samarin, Addison-Wesley, Reading, Massachusetts, 1993, ISBN 0-201-54199-8.

## Contents

# 1 LaTeX 2ε—The new LaTeX release

Over the years many extensions have been developed for LaTeX. This is, of course, a sure sign of its continuing popularity but it has had one unfortunate result: incompatible LaTeX formats came into use at different sites. Thus, to process documents from various places, a site maintainer was forced to keep LaTeX (with and without NFSS), SliTeX, AMS-LaTeX, and so on. In addition, when looking at a source file it was not always clear what format the document was written for.

To put an end to this unsatisfactory situation, this new LaTeX release was produced at the end of 1993; it brings all such extensions back under a single format and thus prevents the proliferation of mutually incompatible dialects of LaTeX 2.09. With LaTeX 2$_\varepsilon$ the 'new font selection' will be standard; style files like amstex (formerly AMS-LaTeX format) or slides (formerly SliTeX format) will become extension packages, all working with the same base format. The introduction of a new release also made it possible to add a small number of often-requested features.

The next section of this document describes very briefly the major new features of LaTeX 2$_\varepsilon$ which are for use in documents.

The following sections describe (again, only in summary form) some of the changes and enhancements which are of concern only to writers of classes and packages (these are the files which replace the 'style files').

# 2 User document commands

## 2.1 Initial commands

These are the commands that can appear before the **\documentclass** line.

`\NeedsTeXFormat {`⟨*format-name*⟩`} [`⟨*release-date*⟩`]`

This command is usually used in package and class files, but it could be used as the first line of your document to declare that the document should run with LaTeX 2$_\varepsilon$. This will ensure that someone who tries to use LaTeX 2.09, or plain TeX, would get a reasonably clear error message straight away.

```
\begin{filecontents} {⟨file-name⟩}
⟨file-contents⟩
\end{filecontents}
```

The **filecontents** environment is intended for packaging, within a single document file, the contents of packages, options, or other files. When the document file is run through LaTeX 2$_\varepsilon$ the body of this environment is written verbatim to a file whose name is given as the environment's only argument. However, if that file already exists (maybe only in the current directory if the OS supports a notion of a 'current directory' or 'default directory') then nothing happens (except for an information message) and the body of the environment is by-passed.

The environment is allowed only before **\documentclass**: this helps to ensure that all packages or options necessary for this particular run are present when needed.

## 2.2 Preamble commands

The changes to these commands are intentionally designed to make LaTeX 2ε documents look clearly different from old documents. The commands should be used only before `\begin{document}`.

`\documentclass [`⟨*option-list*⟩`] {`⟨*class-name*⟩`} [`⟨*release-date*⟩`]`

This command (also here called a 'declaration') replaces the LaTeX 2.09 command `\documentstyle`.

There must be exactly one `\documentclass` declaration in a document, and it must come first (except for the 'initial' commands described above).

The ⟨*option-list*⟩ is a list of options, each of which may modify the formatting of elements which are defined in the ⟨*class-name*⟩ file, as well as those in all following `\usepackage` declarations (see below).

The optional argument ⟨*release-date*⟩ can be used to specify the desired release date of the class file; it should contain a date in the format `YYYY/MM/DD`. If a version of the class older than this date is found, a warning is issued.

`\documentstyle [`⟨*option-list*⟩`] {`⟨*class-name*⟩`} [`⟨*release-date*⟩`]`

This command is still supported for compatibility with old files. It is essentially the same as `\documentclass`, but it loads a 'compatibility mode' which redefines certain commands to act as they did in LaTeX 2.09. It also causes any options in the ⟨*option-list*⟩ that are not processed by the class file to be loaded as packages after the class has been loaded. This matches the 2.09 behaviour.

`\usepackage [`⟨*option-list*⟩`] {`⟨*package-name*⟩`} [`⟨*release-date*⟩`]`

Any number of `\usepackage` declarations is allowed. Each package file (as denoted by ⟨*package-name*⟩) defines new elements (or modifies those defined in the class file loaded by the ⟨*class-name*⟩ argument of the `\documentclass` declaration). A package file thus extends the range of documents which can be processed.

The ⟨*option-list*⟩ argument can contain a list of options, each of which can modify the formatting of elements which are defined in this ⟨*package-name*⟩ file.

As above, ⟨*release-date*⟩ can contain the earliest desired release date of the package file in the format `YYYY/MM/DD`; if an older version of the package is found, a warning is issued.

Each package is loaded only once. If the same package is requested more than once, nothing happens in the second or following attempt unless the package

has been requested with options that were not given in the declaration which caused it to be loaded. If such extra options are specified then an error message is produced.

As well as processing the options given in the ⟨*option-list*⟩ of the corresponding **\usepackage** declaration, each package processes the ⟨*option-list*⟩ of the **\documentclass** declaration as well. This means that any option which should be processed by every package (to be precise, by every package that can do so) can be specified just once, in the **\documentclass** declaration, rather than being repeated for each package that needs it.

Further details of this option processing are in Subsection 2.3 below.

Note: class files have the extension **.cls**; packages have the extension **.sty**.

`\listfiles`

If this declaration is placed in the preamble then a list of the files read in (as a result of processing the document) should be displayed on the terminal (and in the log file) at the end of the run. For some of these listed files, extra information about them will also be produced. (We said 'should' above since this will all happen as documented only if class and package writers always read in files using the mechanisms provided by the system, and we cannot guarantee that this will always be done.)

`\setcounter{errorcontextlines} {`⟨*num*⟩`}`

TEX3 introduced a new primitive **\errorcontextlines** which controls the format of error messages. LaTeX $2_\varepsilon$ provides an interface to this through the standard **\setcounter** command. As most LaTeX users do not want to see the internal definitions of LaTeX commands each time they make an error, LaTeX2e sets this to $-1$ by default.

## 2.3 Option processing

The options specified in the **\documentclass** and **\usepackage** declarations are processed as follows.

1. They are first divided into two types, *local* and *global*:
   - for a class, the options from its **\documentclass** command are local and there are no global options;
   - for a package, the options from its **\usepackage** command are local but the options from the **\documentclass** command are global.

2. The local and global options that have been declared within this class or package are processed first.

   These are usually processed in the order of their declarations in the class or package (thus their order in the ⟨*option-list*⟩ is irrelevant).

   However, it is possible for a class or package to process the options in the order that they appear in each ⟨*option-list*⟩: first the global options; then the local ones.

3. Any local options that are not declared by this class or package are then processed. For document classes, this usually means that they are ignored, except for this fact being recorded by adding the option to a list of 'unused options'; they may, of course, be used later since they become global options for every package subsequently loaded. For packages, this usually means that an error message is produced, giving the choice of retyping the option name in case it is incorrect.

Finally, when `\begin{document}` is reached, if there are any global options which have not been used by either the class or any package, the system will produce a warning message and will list them.

## 2.4 Commands for the main document body

The following commands have been added in LaTeX 2ε, or have extended functionality over their LaTeX 2.09 versions. Some of these can also be used in the preamble.

### 2.4.1 Definitions

The means of creating and changing both commands and environments have been enhanced and extended.

`\newcommand` {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*definition*⟩}
`\renewcommand` {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*definition*⟩}

The optional argument, ⟨*default*⟩, is new in LaTeX 2ε; it allows you to define a command with one optional argument. If this argument to `\newcommand` is present then:

- the first argument of the command ⟨*cmd*⟩ being defined will be an optional argument of ⟨*cmd*⟩;
- when the command ⟨*cmd*⟩ is used without this first (optional) argument being specified, the value of `#1` (i.e. its first argument) in the ⟨*definition*⟩ will be whatever was specified as [⟨*default*⟩] when ⟨*cmd*⟩ was defined.

For example, after the following:

```
\newcommand{\seq}[2][n]{#2_{0},\ldots\,,#2_{#1}}
```

using `\seq{a}` (within a formula) will produce $a_0, \ldots a_n$, whereas `\seq[k]{x}` will produce $x_0, \ldots x_k$.

`\newenvironment` {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*beg-def*⟩} {⟨*end-def*⟩}
`\renewenvironment` {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*beg-def*⟩} {⟨*end-def*⟩}

As for `\newcommand`, LaTeX 2ε also supports defining an environment with one optional argument.

`\providecommand` {⟨*cmd*⟩} [⟨*num*⟩] [⟨*default*⟩] {⟨*definition*⟩}

This takes the same arguments as `\newcommand`. If ⟨*cmd*⟩ is already defined then the existing definition is kept; but if it is currently undefined then the effect of `\providecommand` is to define ⟨*cmd*⟩ just as if `\newcommand` had been used.

### 2.4.2 Boxes

These next three commands all existed in LaTeX 2.09. They have been enhanced in two ways.

`\makebox` [⟨*width*⟩] [⟨*pos*⟩] {⟨*text*⟩}
`\framebox` [⟨*width*⟩] [⟨*pos*⟩] {⟨*text*⟩}
`\savebox` {⟨*cmd*⟩} [⟨*width*⟩] [⟨*pos*⟩] {⟨*text*⟩}

One small but far-reaching change for LaTeX 2ε is that, within the ⟨*width*⟩ argument only, three special lengths can be used. These are all dimensions of the box which would be produced by using simply `\mbox{`⟨*text*⟩`}`:

`\height` its height above the baseline;

`\depth` its depth below the baseline;

`\totalheight` the sum of `\height` and `\depth`;

`\width` its width.

Thus, to put 'hello' in the centre of a box of twice its natural width, you would use:

```
\makebox[2\width]{hello}
```

The other change is a new possibility for ⟨*pos*⟩. If ⟨*pos*⟩ is 's' then the text is stretched the full length of the box, making use of any 'rubber' lengths (including any inter-word spaces) in the box contents. If no such 'rubber length' is present, and 'underfull box' will be produced.

```
\parbox [⟨pos⟩] [⟨height⟩] [⟨inner-pos⟩] {⟨width⟩} {⟨text⟩}
\begin{minipage} [⟨pos⟩] [⟨height⟩] [⟨inner-pos⟩] {⟨width⟩}
⟨text⟩
\end{minipage}
```

As for the LR-mode boxes above, **\height**, **\width**, etc. may be used in the [⟨*height*⟩] argument to denote the natural dimensions of the box.

The ⟨*inner-pos*⟩ argument is new in LaTeX $2_\varepsilon$. It is the vertical equivalent the ⟨*pos*⟩ argument for **\makebox**, etc, determining the position of ⟨*text*⟩ within the box. The ⟨*inner-pos*⟩ may be any one of **t**, **b**, **c**, or **s**, denoting top, bottom, centred, or 'stretched' alignment respectively. The default value of ⟨*inner-pos*⟩ is to be the same as the actual value of ⟨*pos*⟩.

```
\begin{lrbox} {⟨cmd⟩}
⟨text⟩
\end{lrbox}
```

This is an environment which does not directly print anything. Its effect is to save the typeset ⟨*text*⟩ in the bin ⟨*cmd*⟩. Thus it is like **\sbox** {⟨*cmd*⟩} {⟨*text*⟩}, except that any white space before or after the contents ⟨*text*⟩ is ignored.

This is very useful as it enables both the **\verb** command and the **verbatim** environment to be used within ⟨*text*⟩.

It also makes it possible to define, for example, a 'framed box' environment. This is done by first using this environment to save some text in a bin ⟨*cmd*⟩ and then calling **\fbox{\usebox{**⟨*cmd*⟩**}}**.

The following example defines an environment, called **fmpage**, that is a framed version of **minipage**.

```
\newsavebox{\fmbox}%
\newenvironment{fmpage}[1]%
  {\begin{lrbox}{\fmbox}\begin{minipage}{#1}}%
  {\end{minipage}\end{lrbox}\fbox{\usebox{\fmbox}}}%
```

### 2.4.3 Measuring things

The first of these next commands was in LaTeX 2.09. The two new commands are the obvious analogues.

`\settowidth {⟨length-cmd⟩} {⟨lr text⟩}`
`\settoheight {⟨length-cmd⟩} {⟨lr text⟩}`
`\settodepth {⟨length-cmd⟩} {⟨lr text⟩}`

### 2.4.4 Controlling page breaks

Sometimes it is necessary, for a final version of a document, to 'help' LaTeX break the pages in the best way. LaTeX 2.09 had a variety of commands for this situation: **\clearpage**, **\samepage** etc. LaTeX 2ε provides, in addition, commands which can produce longer pages as well as shorter ones.

`\enlargethispage {⟨size⟩}`
`\enlargethispage* {⟨size⟩}`

These commands increase the height of a page (from its normal value of **\textheight**) by the specified amount ⟨*size*⟩, a rigid length. This change affects *only* the current page.

This can be used, for example, to allow an extra line to be fitted onto the page or, with a negative length, to produce a page one line shorter than normal.

The star form also shrinks any vertical white space on the page as much as possible, so as to fit the maximum amount of text on to the page.

### 2.4.5 Floats

There is a new command, **\suppressfloats**, and a new 'float specifier'. We hope that these will enable people to gain better control of LaTeX's float placement algorithm.

`\suppressfloats [⟨placement⟩]`

This command stops any further floating environments from being placed on the current page. With an optional argument, which should be either **t** or **b** (not both), this restriction applies only to putting further floats at the top or at the bottom.

The extra float location specifier: !

This can be used, along with at least one of **h**, **t**, **b** and **p**, in the location optional argument of a float.

If a **!** is present then, just for this particular float, whenever it is processed by the float mechanism the following are ignored:

- all restrictions on the number of floats which can appear;
- all explicit restrictions on the amount of space on a text page which may be occupied by floats or must be occupied by text.

The mechanism will, however, still attempt to ensure that pages are not overfull and that floats of the same type are printed in the correct order.

Note that its presence has no effect on the production of float pages.

A ! specifier overrides the effect of any **\suppressfloats** command for this particular float.

### 2.4.6 Font changing: text

The font change commands are described in detail in Section 11.

Here is a brief description of the two major new classes of commands which affect the font to be used to typeset text.

```
\rmfamily
\bfseries
⋮
```

These are font declarations whose use is the same as the declarations **\rm**, **\bf**, etc. The difference is that each command changes just one attribute of the font (the attribute changed is part of the name). One result of this is that, for example, **\bfseries\itshape** produces both a change of series and a change of shape, to give a bold italic font.

```
\textrm {⟨text⟩}
\textbf {⟨text⟩}
\emph {⟨text⟩}
⋮
```

These are command forms; this means that they take as an argument the text which is to be typeset in the particular font. They also automatically insert italic corrections where required; if you do not like the result, this can be over-ridden by using \/ or **\nocorr**.

### 2.4.7 Font changing: maths

Most of the fonts used within math mode do not need to be explicitly invoked; but to use letters from a range of fonts, the following class of commands is provided.

```
\mathrm {⟨letters⟩}
\mathnormal {⟨letters⟩}
\mathcal {⟨letters⟩}
.
.
.
```

These are also command forms so they take as an argument the letters which are to be typeset in the particular font. The argument is processed in math mode so spaces within it will be ignored. Also, in general, the treatment of punctuation, digits, etc is not affected.

### 2.4.8 Ensuring math mode

`\ensuremath {⟨math commands⟩}`

In LaTeX 2.09, if you wanted a command to work both in math mode and in text mode, the suggested method was to define something like:

```
\newcommand{\Gp}{\mbox{$G_p$}}
```

Unfortunately, the `\mbox` stops `\Gp` changing size correctly in (for instance) subscripts or a fraction.

In LaTeX $2_\varepsilon$ you can define it thus:

```
\newcommand{\Gp}{\ensuremath{G_p}}
```

Now `\Gp` will work correctly in all contexts.

This is because the `\ensuremath` does nothing when `\Gp` is used within math mode, but it ensures that math mode is entered (and exited) as required when `\Gp` is used in text mode.

### 2.4.9 Hiding special tokens.

`\literal {⟨text⟩}`

`\literal` is used to 'hide' characters.

This is necessary in certain contexts: for instance, to use a `]` inside an optional argument you need to type: `\item[\literal{]}]`. It can also be used to 'break a ligature': compare the detailed appearance of 'shelfful' and 'shelfful', produced by '`shelfful`' and '`shel\literal{f}ful`'. (The TeXBook suggests `shelf{}ful` for the latter effect but unfortunately that can fail as TeX will 're-constitute the ligature' ff if it tries to hyphenate `shelf{}ful`.)

### 2.4.10 Logos

```
\LaTeX
\LaTeXe
```

`\LaTeX` (producing 'LaTeX') is still the 'main' logo command, but if you need to refer to the new features, you can say `\LaTeXe` (producing 'LaTeX 2ε').

# 3 Warning: stop here?

The rest of this document is aimed exclusively at those who need a quick reference guide to the inner workings of LaTeX 2ε. If this is not for you, then stop here!

# 4 Command naming conventions in LaTeX 2ε

LaTeX 2ε attempts to classify commands as follows, and to link this classification to their appearance (syntactic form).

**Document commands** Commands that occur in the main document file. These should have lower case names. (For example, `\alpha`, `\mbox`.)

**Class & Package Interface commands** The high level interfaces for packages and their options. These commands have mixed case names. (For example `\ProvidesClass`.)

**LaTeX programmer commands** These commands help package writers to build up the structures they require. They are lower case, usually with a single `@` as the first character. (For example `\@ifundefined`, `\@tfor`.)

**Saved definitions** These commands save original definitions. They start with `@@`. (For example `\@@end`.) LaTeX programmers are unlikely to need to use these commands, but they should avoid naming their own commands with names starting `\@@`...

**Internal commands for a particular package** These commands also have names containing `@` but they often have more than one, or it is not the first letter in the command name. When writing a package, it is good practice to give all the internal commands a consistent naming scheme as this reduces the chances of a clash with other packages. (For example, longtable internal commands include `\LT@output`, `\LT@start`.)

Note that the above conventions can be broken (**\Box** is a user level command that is mixed case, **\outer** has a user-level lowercase name but should never be used at all!). Also, it is not always clear to which category a command belongs; nevertheless, it is helpful to bear these conventions in mind when writing new packages.

The system itself does not try to enforce these conventions except that commands named with an **@** may not usually be used in the main document.

# 5 System dependent commands

When the LaTeX $2_\varepsilon$ format is made, certain tests are run to try to determine the behaviour of the TeX implementation in use and the syntax of external file names. These tests may be overruled by the site maintainer but in any case the following commands should be set up to match the local system.

The file **dircheck.dtx** contains a more detailed explanation of these requirements, including examples. This documentation may be typeset by running LaTeX $2_\varepsilon$ on the file **dircheck.drv**.

`\@currdir⟨filename⟩⟨space⟩`

**\@currdir**⟨*filename*⟩⟨*space*⟩ should expand to the syntax on the local operating system for the file ⟨*filename*⟩ in the current directory. The expansion should end with a ⟨*space*⟩.

`\input@path`

On most TeX systems **\input@path** should be undefined. It is needed on systems where the TeX program does not look in the same place for files when using both **\input** and **\openin**. On these systems, LaTeX needs **\input@path** to be defined to be a list of directories (with the same syntax as **\@currdir**). Each directory should be in a **{}** group.

Examples:

```
{ {/usr/lib/tex/inputs/} {/usr/local/lib/tex/inputs/} }
{ {c:/tex/inputs/} }
{ {tex$inputs:} }
```

`\@filename@parse {⟨filename⟩}`

**\filename@parse{**⟨*filename*⟩**}** defines the following three macros:
**\filename@area** the 'area' or 'directory' part of ⟨*filename*⟩
**\filename@base** the 'base' part of ⟨*filename*⟩

`\filename@ext` the 'extension' part of ⟨*filename*⟩
If ⟨*filename*⟩ did not have an extension, `\filename@ext` will be `\let` to `\relax`.

# 6 High level class & package interface

The commands in this section allow class and package writers to clearly identify their files and to define the options that they support. In LaTeX 2.09, a main style could only define options using 'low level' definitions. Packages (which were implemented as options to the main style) could not have options in previous releases.

## 6.1 Identification

`\NeedsTeXFormat {`⟨*format-name*⟩`} [`⟨*release-date*⟩`]`

Ensures that the current file is processed under a TeX format with name ⟨*format-name*⟩. With ⟨*release-date*⟩ one can specify the earliest release date of the format that should still work. Any release older than this date will be flagged with a warning.

The standard ⟨*format-name*⟩ is **LaTeX2e**. The date, if present, must be in the form YYYY/MM/DD.

Example:

```
\NeedsTeXFormat{LaTeX2e}[1994/01/01]
```

`\ProvidesClass {`⟨*class-name*⟩`} [`⟨*release-info*⟩`]`
`\ProvidesPackage {`⟨*package-name*⟩`} [`⟨*release-info*⟩`]`

This declares that the current file contains the definitions for the document class or package ⟨*class-name*⟩. The optional ⟨*release-info*⟩ contains the release date in the form YYYY/MM/DD, optionally followed by a short description (any sequence of non-special characters). The date information can be used by `\LoadClass` or `\documentclass` (for classes) or `\RequirePackage` or `\usepackage` (for packages) to test if the release is not obsolete. The full information is displayed by `\listfiles` and should therefore not be too long.

Example:

```
\ProvidesClass{article}[1994/01/01 Standard LaTeX2e class]
\ProvidesPackage{ifthen}[1994/01/01 Standard LaTeX2e package]
```

`\ProvidesFile {`⟨*file-name*⟩`} [`⟨*release-info*⟩`]`

As for the two previous commands, but here the full filename, including extension must be given. Used for declaring any files other than main class and package files.

`\RequirePackage [`⟨*options-list*⟩`] {`⟨*package-name*⟩`} [`⟨*release-info*⟩`]`

With this command, packages can load other packages. The behaviour is similar to that of **\usepackage**: if the package ⟨*package-name*⟩ has already been loaded, then nothing happens unless the requested options are not a subset of the options with which it was loaded; if they are not then an error is signalled.

Example:

```
\RequirePackage{ifthen}[1994/01/01 Standard LaTeX2e package]
```

`\LoadClass [`⟨*options-list*⟩`] {`⟨*package-name*⟩`} [`⟨*release-info*⟩`]`

This command is similar to **\RequirePackage**, but it is for use by classes only, i.e. it must not be used in packages files.

`\PassOptionsToPackage {`⟨*options-list*⟩`} {`⟨*package-name*⟩`}`
`\PassOptionsToClass {`⟨*options-list*⟩`} {`⟨*package-name*⟩`}`

With this command, packages can pass options to another package. This adds the ⟨*option-list*⟩ to the list of options of any future **\RequirePackage** or **\usepackage** command for package ⟨*package-name*⟩.

For example,

```
\PassOptionsToPackage{foo,bar}{fred}
\RequirePackage[baz]{fred}
```

is the same as:

```
\RequirePackage[foo,bar,baz]{fred}
```

Similarly, **\PassOptionsToClass** may be used to pass options to a class.

`\@ifclassloaded {`⟨*class-name*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`
`\@ifpackageloaded {`⟨*package-name*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`

These two commands can be used to execute different pieces of code, depending on whether a class or package has already been loaded.

`\@ifclasslater {`⟨*class-name*⟩`} {`⟨*release-date*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`
`\@ifpackagelater {`⟨*package-name*⟩`} {`⟨*release-date*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`

With these two commands, different pieces of code can be executed, depending on whether a class or package has already been loaded that is version more recent than ⟨*release-date*⟩.

`\@ifclasswith {`⟨*class-name*⟩`} {`⟨*option-list*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`
`\@ifpackagewith {`⟨*package-name*⟩`} {`⟨*option-list*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`

With these two commands, different pieces of code can be executed, depending on whether a class or package has already been loaded at least the options given in ⟨*option-list*⟩.

## 6.2 Option handling

`\DeclareOption {`⟨*option-name*⟩`} {`⟨*code*⟩`}`

Declares ⟨*option-name*⟩ to be an option for the current class or package and ⟨*code*⟩ the code to be executed if that option is specified.

The ⟨*code*⟩ can contain any valid LaTeX 2ε construct, plus some special commands for use within this argument which are described below.

Example:

```
\DeclareOption{twoside}{\@twosidetrue}
```

`\DeclareOption* {`⟨*code*⟩`}`
`\OptionNotUsed`

Declares ⟨*code*⟩ to be executed for every option which is otherwise not explicitly declared. By default, undeclared options to a class will be silently passed to all packages (just like the declared options for the class); undeclared options to a package will produce an error.

The ⟨*code*⟩ can contain any valid LaTeX 2ε construct, plus some special commands for use within this argument which are described below.

The example below handles an undeclared option by loading a `.clo` file if it exists. Otherwise it declares the option as unused.

```
\DeclareOption*{%
  \InputIfFileExists{\CurrentOption.clo}{}{\OptionNotUsed}}
```

`\CurrentOption`

Refers to the name of the current option within the ⟨*code*⟩ of `\DeclareOption` or `\DeclareOption*`.

`\ProcessOptions`

Executes the code for all options specified in the order they are defined in the file. Afterwards reclaims the memory used by ⟨*code*⟩ (as given in a `\DeclareOptions` declaration) for *all* options defined in the file.

`\ProcessOptions*`
`\@options`

Like `\ProcessOptions` but executes options in the order specified in the calling command rather than in the order specified in the class or package. The `\@options` command from LaTeX 2.09 has been made equivalent to this in order to ease the task of updating old main document styles to LaTeX $2_\varepsilon$ class files.

`\AtEndOfClass {`⟨*code*⟩`}`
`\AtEndOfPackage {`⟨*code*⟩`}`

These commands cause ⟨*code*⟩ to be saved away in an internal hook, and then executed at the end of the current class (package). Repeated use of the commands work, and the arguments are executed in the order they are declared.

`\AtBeginDocument {`⟨*code*⟩`}`
`\AtEndDocument {`⟨*code*⟩`}`

These commands cause ⟨*code*⟩ to be saved internally and executed while LaTeX is executing `\begin{document}` (`\end{document}`).

## 6.3 Safe Input Macros

`\InputIfFileExists {`⟨*file-name*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`

Inputs ⟨*file-name*⟩ if it exists. Immediately before the input, ⟨*true*⟩ is executed. Otherwise ⟨*false*⟩ is executed.

`\IfFileExists {`⟨*file-name*⟩`} {`⟨*true*⟩`} {`⟨*false*⟩`}`

As above, but does not input the file. One thing that you might like to put in the ⟨*false*⟩ clause is:

`\@missingfileerror {⟨file-basename⟩} {⟨extension⟩}`

This starts an interactive request for a filename, supplying default extensions. Just hitting the Return key causes the whole input to be skipped.

`\input {⟨file-name⟩}`

This command has been redefined from the LaTeX2.09 definition, in terms of `\InputIfFileExists` and `\@missingfileerror`.

# 7 LaTeX programmer commands

There have been many internal changes made while preparing LaTeX 2ε. Only those changes that directly affect package writers will be listed here.

This section is not yet complete.

## 7.1 Obsolete commands

These commands, etc. are no longer available.

`\footheight`

This parameter was declared in LaTeX2.09, and some styles set it to a value, but the value was never used. It has been removed.

`\@maxsep`
`\@dblmaxsep`

These two parameters are no longer used so they have been removed.

## 7.2 New commands

`\@checkcommand {⟨cmd⟩} [⟨num⟩] [⟨default⟩] {⟨definition⟩}`

This takes the same arguments as **\newcommand** but, rather than define ⟨*cmd*⟩, it checks that the current definition of ⟨*cmd*⟩ is ⟨*definition*⟩. An error is raised if the definition is different.

This command may be useful for checking the state of the system before your package starts altering command definitions. It allows you to check that no other package has redefined the same command.

`\two@digits {⟨num⟩}`

This is like the primitive command **\number** except that numbers less than 10 (warning: this includes all negative numbers!) get a leading '0'. This is useful for producing dates of the form 1994/01/01.

`\@percentchar`

This produces a non-special % token. It may be used in **\typeout** and similar commands to produce a percent-sign.

`\if@compatibility`

This flag is true if the document is being run in LaTeX 2.09 compatibility mode.

`\@firstofone {⟨arg⟩}`
`\@firstoftwo {⟨arg1⟩} {⟨arg2⟩}`
`\@secondoftwo {⟨arg1⟩} {⟨arg2⟩}`

**\@firstofone** just returns its argument (it has its uses...). **\@firstoftwo** returns its first argument, discarding the second. **\@secondoftwo** returns its second argument, discarding the first.

`\paperheight`
`\paperwidth`

These two parameters are usually set by the class to be the size of the paper being used. This should be actual paper size, unlike **\textwidth** and **\textheight** which are the size of the main text body within the margins.

# 8 Incompatibilities and changes

Section 2.2 describes the more obvious changes which affect the preamble of the document.

This section describes the major differences which will be found when processing a document. It compares LaTeX $2_\varepsilon$ with various old versions of LaTeX.

## 8.1 Compared to the original LaTeX 2.09

### 8.1.1 Font changes

In LaTeX 2.09 the only font changing commands (apart from the size commands) were the 'two-letter' forms `\rm`, `\bf` etc. These each referred to a *fixed* font (at the given size) so, for instance, the standard commands could not select a bold italic font. The combination `\bf\it` selected a medium weight italic (the `\it` completely replacing the `\bf`) and similarly `\it\bf` produced an upright bold font.

The size-changing commands such as `\large` always switched to a roman font. LaTeX $2_\varepsilon$ introduces the more flexible font changing declarations, such as `\rmfamily`, and commands, such as `\textrm`.

The old 'two-letter' font commands *are not defined*! by the core LaTeX $2_\varepsilon$ system.

The standard classes define `\rm`, `\bf` etc., to select a fixed text font. The size commands, however, just change the size; they do not select a roman font. Furthermore, the commands `\rm`, `\bf`, `\it` are allowed in math mode (unlike the new NFSS text font commands and declarations.)

### 8.1.2 Compatibility mode

If the document begins with `\documentstyle` rather than `\documentclass` then a more complete emulation of LaTeX 2.09 is used.

- A `.sty` file will be loaded if a `.cls` file cannot be found.
- Size changing commands switch to a roman font.
- All the standard two-letter font commands are allowed in math.
- The declaration `\sloppy`, which normally has an improved definition, reverts to its old definition; this helps maintain the line-breaking on existing documents.

## 8.2 Compared to 2.09 with NFSS release 1

Release 1 of the NFSS introduced the concept of orthogonal font switching but not the new declaration names such as `\rmfamily`. With the `newlfont` option it redefined the existing two-letter forms and their use was not allowed in math mode. LaTeX2e is distributed with a package `newlfont` that defines `\rm`, `\bf` etc in a way similar to NFSS1.

The following list details some more differences between NFSS1 and LaTeX $2_\varepsilon$.

- These obsolete commands from NFSS1 have been removed: `\family`, `\series`, `\shape`, and `\size`. Instead use `\fontfamily`, `\fontseries`, `\fontshape`, and `\fontsize`.
- The math alphabets `\mit` and `\cal` have been renamed to `\mathnormal` and `\mathcal`.
- The commands `\normalshape` and `\mediumseries` have been renamed to `\upshape` and `\mdseries`.
- The following internal commands are also now obsolete: `\extra@defs`, `\new@fontshape`, `\subst@fontshape`, `\newmathalphabet`, `\addtoversion`.
- In NFSS1, an explicit request for `\fontsize{14}{16pt}` was actually trying to load a font at 14.4pt; in other words '14' was used as a label and not as a real size specification. In LaTeX $2_\varepsilon$ this request is now interpreted as asking for a font at '14pt' sharp. Since in the standard FontShape declarations such a font is not available, LaTeX $2_\varepsilon$ will attempt to use a nearby size and therefore will (after a warning) also use the '14.4pt' size, so documents will come out unchanged.

  Also, the new version of `\fontsize` allows other units to be used in its arguments (defaulting to `pt`) so `\fontsize{12dd}{15dd}` is an acceptable declaration (assuming you have fonts for that size).
- The internal switch `\ifdefine@mathfonts`, used to suppress math font changes, was removed. Use `\ifmath@fonts` instead but do not rely on it for future interfaces either!

## 8.3 Compared to 2.09 with NFSS2

While LaTeX $2_\varepsilon$ was being developed, a prototype NFSS2 release was made available for use with LaTeX 2.09. The internal structure of LaTeX $2_\varepsilon$ has not significantly altered from this but the prototype, like NFSS1, redefined the two-letter font changing commands rather than introducing the new `\rmfamily` forms. Use the package newlfont if you would prefer this behaviour in LaTeX $2_\varepsilon$.

The other major difference is that all the styles (packages) distributed with the prototype had a consistent naming scheme nf.... For example nftimes.sty, nfoldlfnt.sty. After some discussion, it was decided to drop this idea and revert to the old names times.sty, oldlfont.sty.

# 9 Option naming conventions

No option names are built into the LaTeX core system, but we hope that package and class writers will follow these conventions for names of options that have general applicability. Of course, packages may also have options with names particular to the package.

**language options** If your package produces 'fixed strings' that produce text in the output then please consider supporting language options as defined by the babel package.

**paper size options** The standard classes support the options `letterpaper`, `legalpaper`, `executivepaper`, `a4paper`, `a5paper`, `b5paper`; any paper size can be modified by the option `landscape`. These set the `\paperheight` and `\paperwidth` parameters. When writing a class that supports different paper sizes you should make similar options, with names ending in `paper`, for those sizes that you support.

**driver options** Many graphics related packages need to know what dvi-driver is being used (so that the correct syntax may be used in the `\special` command). It is hoped that conventional option-names will be agreed denoting the major drivers. Currently the suggested list includes: `emtex`, `dvips`, `oztex` etc.

**debugging options** errorshow warningshow infoshow debugshow pausing

# 10 File extension conventions

The LaTeX $2_\varepsilon$ system uses many different types of files. The main file extensions and their meanings are listed here.

`l2e` Plain text files containing basic documentation, etc.

`ins` Docstrip batch files: run iniTeX on these to unpack files.
`ltx` Internal LaTeX files, used only during the installation.
`cfg` Configuration files: these are the only files that you are allowed to edit to customise LaTeX for your site.

`fdd` Documented sources for font definition files.
`dtx` Documented sources for other files.
`drv` Driver files: run LaTeX on these to produce documentation.

`cls` Class files.
`clo` Files implementing class options.
`sty` Package files (and LaTeX 2.09 style files).
`fd` Font definition files.
`def` Other LaTeX definition files.

`tex` LaTeX document.
`aux` LaTeX document information.
`toc` LaTeX document table-of-contents information.
`ist` Makeindex style files.
`idx` Raw index files.
`ind` Sorted index files.
`gls` Raw glossary files.
`glo` Sorted Glossary files.
`log` TeX log file (some systems).
`lis` TeX log file (some systems).
`ilg` Makeindex log file.
`fmt` TeX format file (binary).
`dvi` TeX output file (binary).

# 11 Font interface commands

## 11.1 Using text fonts

There are two ways of changing the text font: commands, which take the affected text as an argument; declarations, which change the text from that point onward.

Although the command form is the best one for use in documents, it does more work so it is not as efficient as the declaration form. Therefore, when writing package files it is better to use a declaration unless the automatic insertion of the italic correction (see below) is needed.

### 11.1.1 Font changing commands

Font changes use the following commands.
THe font family is changed with:

```
\textrm{...} \textsf{...} \texttt{...}
```

The font series (weight/width) is changed with:

```
\textbf{...} \textmd{...}
```

The font shape is changed with:

```
\textit{...} \textsl{...} \textsc{...} \textup{...}
```

Finally, emphasis can be expressed with

```
\emph{...}
```

All such commands take care of any necessary italic correction (\/). In case you want to suppress this extra space you can use **\nocorr** at an appropriate place, depending on whether you wish to suppress the correction before or after the argument text.

The automatic insertion of the italic correction is regulated by the contents of **\nocorrlist**. This is a list of characters; immediately in front of these, no italic correction is inserted. Its value could, for example, be set as follows:

```
\def\nocorrlist{.,;:}
```

### 11.1.2 Font changing declarations

Each of the above commands has a corresponding declaration.

Font family:

```
{\rmfamily ...} {\sffamily ...} {\ttfamily ...}
```

Font series (weight/width):

```
{\bfseries ...} {\mdseries ...}
```

Font shape:

```
{\itshape ...} {\slshape ...} {\scshape ...} {\upshape ...}
```

Emphasis:

```
{\em ...}
```

### 11.1.3 Font hooks

NFSS provides a set of built-in hooks that modify the behaviour of the high-level font changing commands. These hooks are shown in table 1.

The initial setting of **\familydefault** means that changing only **\rmdefault** will also change **\familydefault** to its value. However, if **\familydefault** is changed **\rmdefault** is not affected.

### 11.1.4 Accessing symbols in text

Use **\symbol{**⟨*number*⟩**}** to select the symbol in position ⟨*number*⟩ of the current font. The characters ' or " at the beginning of ⟨*number*⟩ denote octal or hexadecimal notation, respectively.

| *Hook* | *Default value* | *Description* |
|---|---|---|
| \encodingdefault | OT1 | Encoding scheme for 'main font' |
| \familydefault | \rmdefault | Font family selected for 'main font' |
| \seriesdefault | m | Series selected for 'main font' |
| \shapedefault | n | Shape selected for 'main font' |
| \rmdefault | cmr | Font family selected by **\rmfamily** and **\textrm** |
| \sfdefault | cmss | Font family selected by **\sffamily** and **\textsf** |
| \ttdefault | cmtt | Font family selected by **\ttfamily** and **\texttt** |
| \bfdefault | bx | Series selected by **\bfseries** and **\textbf** |
| \mddefault | m | Series selected by **\mdseries** and **\textmd** |
| \itdefault | it | Shape selected by **\itshape** and **\textit** |
| \sldefault | sl | Shape selected by **\slshape** and **\textsl** |
| \scdefault | sc | Shape selected by **\scshape** and **\textsc** |
| \updefault | n | Shape selected by **\upshape** and **\textup** |

Table 1: Font attribute hooks

Use **\oldstylenums{**⟨*number*⟩**}** to produce non-aligned digits like 1982: this works in text and math and honours spaces in the argument if used in text. Do not try to put other characters in the argument, the result would be unpredictable and may change in later releases.

## 11.2 Math fonts

### 11.2.1 Predefined math versions

The standard format declares two math versions: 'normal' and 'bold'.

(You can switch between them using **\boldmath** and **\unboldmath** or directly using **\mathversion{...}**.)

### 11.2.2 Predefined math alphabets

The system provides the following.

```
\mathnormal \mathcal \mathrm \mathbf \mathsf \mathit \mathtt
```

If you use the above commands often, consider defining an abbreviation in the preamble of your document, e.g.

```
\newcommand{\mrm}{\mathrm}
```

### 11.2.3 Declaring math sizes

`\DeclareMathSizes` ⟨*t-size*⟩ ⟨*mt-size*⟩ ⟨*s-size*⟩ ⟨*ss-size*⟩

Declares that ⟨*mt-size*⟩ is the math text size, ⟨*s-size*⟩ is the script size and ⟨*ss-size*⟩ the scriptscript size to be used in math, when ⟨*t-size*⟩ is the current text size. For text sizes for which no such declaration is given the script and scriptscript size will be calculated and then fonts are loaded for the calculated sizes or the best approximation (this may result in some warning message by NFSS).

Normally ⟨*t-size*⟩ and ⟨*mt-size*⟩ will be identical, however, if, for example, PostScript text fonts are mixed with bit-map math fonts you may not have for every ⟨*t-size*⟩ a corresponding set of math sizes.

Example:

```
\DeclareMathSizes{13.82}{12.4}{10}{7}
```

### 11.2.4 Declaring math versions

`\DeclareMathVersion` ⟨*version-name*⟩

Defines ⟨*version-name*⟩ to be a math version. Initializes this version with the default for all symbol fonts declared so far (see **\DeclareSymbolFont**). Internally stores the stuff in **\mv@**⟨*version-name*⟩.

If used on an already existing version, a warning is issued and all previous **\SetSymbolFont** declarations for this version are overwritten by the symbol font defaults, i.e. one ends up with a virgin math version.

Example:

```
\DeclareMathVersion{normal}
```

### 11.2.5 Declaring math alphabets

`\DeclareMathAlphabet` ⟨*id*⟩ ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩

Defines ⟨*id*⟩ to be a new math alphabet with the default ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩ in all versions (the old **\newmathalphabet*** plus encoding scheme). If ⟨*shape*⟩ is empty then the ⟨*id*⟩ is declared to be invalid in all versions unless overwritten later by a **\SetMathAlphabet** command.

Checks that ⟨*id*⟩ can be used and that ⟨*cdp*⟩ is a valid encoding scheme.

In these examples, **\foo** is defined everywhere but **\baz**, by default, is defined nowhere.

```
\DeclareMathAlphabet{\foo}{OT1}{cmtt}{m}{n}
\DeclareMathAlphabet{\baz}{OT1}{}{}{}
```

`\SetMathAlphabet` ⟨*id*⟩ ⟨*version-name*⟩ ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩

Changes the setting of the math alphabet ⟨*id*⟩ in math version ⟨*version-name*⟩ to ⟨*cdp*⟩⟨*family*⟩⟨*series*⟩⟨*shape*⟩.

Checks that ⟨*id*⟩ is a math alphabet, ⟨*version-name*⟩ is a math version and ⟨*cdp*⟩ is a known encoding scheme.

This example defines `\baz` for the 'normal' math version only):

```
\SetMathAlphabet\baz{normal}{OT1}{cmss}{m}{n}
```

Note that this declaration is not used for all math alpahabets: section 11.2.6 describes `\DeclareSymbolFontAlphabet`, which is used to set up math alphabets contained in fonts which have are declared as symbol fonts.

### 11.2.6 Declaring symbol fonts

`\DeclareSymbolFont` ⟨*sym-font-name*⟩ ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩

Defines ⟨*sym-font-name*⟩ to be a symbolic name for a new symbol font.

The arguments ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩ are the default values for this symbol font in all math versions if not redefined later with a `\SetSymbolFont` command.

Checks if ⟨*cdp*⟩ is a declared encoding scheme.

Internally it allocates a new math group with name `\sym`⟨*sym-font-name*⟩.

For example, the following sets up the first four standard math fonts:

```
\DeclareSymbolFont{operators}{OT1}{cmr}{m}{n}
\DeclareSymbolFont{letters}{OML}{cmm}{m}{it}
\DeclareSymbolFont{symbols}{OMS}{cmsy}{m}{n}
\DeclareSymbolFont{largesymbols}{OMX}{cmex}{m}{n}
```

`\SetSymbolFont` ⟨*sym-font-name*⟩ ⟨*version name*⟩ ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩

Changes the setting for the symbol font ⟨*sym-font-name*⟩ in math version ⟨*version name*⟩ to ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩.

Checks that ⟨*sym-font-name*⟩ is a symbol font, ⟨*version name*⟩ is a known math version and ⟨*cdp*⟩ is a declared encoding scheme.

For example, the following come from the set up of the 'bold' math version:

```
\SetSymbolFont{operators}{bold}{OT1}{cmr}{bx}{n}
\SetSymbolFont{letters}{bold}{OML}{cmm}{b}{it}
```

`\DeclareSymbolFontAlphabet` ⟨*id*⟩ ⟨*sym-font-name*⟩

Allows use of the math alphabet contained in the symbol font that was declared earlier as ⟨*sym-font-name*⟩.

This declaration should be used in preference to **\DeclareMathAlpahbet** and **\SetMathAlphabet** when the math alphabet is part of a symbol font; this is because it makes better use of the limited number (only 16) of TEX's math groups.

Checks that ⟨*id*⟩ can be defined and that ⟨*sym-font-name*⟩ is a symbol font.

Example:

```
\DeclareSymbolFontAlphabet{\mathrm}{operators}
\DeclareSymbolFontAlphabet{\mathcal}{symbols}
```

## 11.3 Math symbols

`\DeclareMathSymbol` ⟨*symbol*⟩ ⟨*type*⟩ ⟨*sym-font-name*⟩ ⟨*position*⟩

The ⟨*symbol*⟩ can be either a single character, eg '>', or a macro name, eg **\sum**.

Defines this macro or character ⟨*symbol*⟩ to be a math symbol of type ⟨*type*⟩ (values 0-7 or symbolic see below) in font position ⟨*position*⟩ of symbol font ⟨*sym-font-name*⟩ which must have been declared previously. The ⟨*position*⟩ can be decimal, octal, or hexadecimal with the usual notation ' or " for octal and hex. Symbolic values for ⟨*type*⟩ are the corresponding **\mathord**, **\mathbin**,... commands.

Allows a macro ⟨*symbol*⟩ to be redefined only if it was previously defined to be a math symbol.

Checks that ⟨*symbol*⟩ can be used and that ⟨*sym-font-name*⟩ is a declared symbol font.

Example:

```
\DeclareMathSymbol{\alpha}{0}{letters}{"0B}
\DeclareMathSymbol{\lessdot}{\mathbin}{AMSb}{"0c}
\DeclareMathSymbol{\foo}{\mathalpha}{AMSb}{"0c}
```

`\DeclareMathDelimiter` ⟨*cmd*⟩ ⟨*type*⟩ ⟨*sym-font-name-1*⟩ ⟨*position-1*⟩ ⟨*sym-font-name-2*⟩ ⟨*position-2*⟩

or

`\DeclareMathDelimiter` ⟨*char*⟩ ⟨*sym-font-name-1*⟩ ⟨*position-1*⟩ ⟨*sym-font-name-2*⟩ ⟨*position-2*⟩

Defines ⟨*cmd*⟩ or ⟨*char*⟩ to be a math delimiter where the small variant is in font position ⟨*position-1*⟩ of symbol font ⟨*sym-font-name-1*⟩ and the large variant in font position ⟨*position-2*⟩ of symbol font ⟨*sym-font-name-2*⟩. Both symbol fonts must have been declared previously. The ⟨*position-i*⟩ can be decimal, octal, or hexadecimal with the usual notation ' or " for octal and hex.

Checks that ⟨*cmd*⟩ can be used and that ⟨*sym-font-name-i*⟩ are both declared symbol fonts.

If TEX is not looking for a delimiter, ⟨*cmd*⟩ is treated just as if it had been defined with **\DeclareMathSymbol** using ⟨*sym-font-name-1*⟩ ⟨*position-1*⟩. In other words, if a command is defined as a delimiter then this automatically defines it as a math symbol.

However, if this command is used to define a single character as a delimiter this has no effect on its normal use. It can be defined, via **\DeclareMathSymbol**, to have completely different behaviour when it is used as a math symbol.

Example:

```
\DeclareMathDelimiter{\langle}{\mathopen}{symbols}{"68}
                                          {largesymbols}{"0A}
```

`\DeclareMathAccent` ⟨*cmd*⟩ ⟨*type*⟩ ⟨*sym-font-name*⟩ ⟨*position*⟩

Defines ⟨*cmd*⟩ to act as a math accent. The accent character comes from ⟨*sym-font-name*⟩ in position ⟨*position*⟩. The ⟨*type*⟩ can be either **\mathord** or **\mathalpha**; in the latter case the accent character changes font when used in a math alphabet.

Example:

```
\DeclareMathAccent{\acute}{\mathalpha}{operators}{"13}
\DeclareMathAccent{\vec}{\mathord}{letters}{"7E}
```

`\DeclareMathRadical` ⟨*cmd*⟩ ⟨*sym-font-name-1*⟩ ⟨*position-1*⟩ ⟨*sym-font-name-2*⟩ ⟨*position-2*⟩

Defines ⟨*cmd*⟩ to be a radical where the small variant is in font position ⟨*position-1*⟩ of symbol font ⟨*sym-font-name-1*⟩ and the large variant in font position ⟨*position-2*⟩ of symbol font ⟨*sym-font-name-2*⟩. Both symbol fonts must have been declared previously. ⟨*position-i*⟩ can be decimal, octal, or hexadecimal with the usual notation ' or " for octal and hex.

Example (probably the only use for it!):

```
\DeclareMathRadical\sqrt{symbols}{"70}{largesymbols}{"70}
```

## 11.4 Text font declarations

### 11.4.1 Encoding

`\DeclareFontEncoding` ⟨*cdp*⟩ ⟨*text-settings*⟩ ⟨*math-settings*⟩

Declares a new encoding scheme ⟨*cdp*⟩. The ⟨*text-settings*⟩ are declarations which are executed every time **\selectfont** switches to encoding scheme ⟨*cdp*⟩. (More exactly, they are already executed when **\fontencoding** is encountered and the new encoding differs from the previous encoding.) This can be used for changing definitions of accents or other beasts that depend on font positions.

The ⟨*math-settings*⟩ are similar but for math alphabets (only!). They will be executed whenever a math alphabet with this encoding is called. Spaces within the arguments are ignored to avoid surplus spaces in the document in funny places. If a real space is necessary use **\space**.

Example:

```
\DeclareFontEncoding{OT2}{}{\noaccents@}
```

`\DeclareFontEncodingDefaults` ⟨*text-settings*⟩ ⟨*math-settings*⟩

Declares ⟨*text-settings*⟩ and ⟨*math-settings*⟩ for all encoding schemes. These are executed before the encoding scheme dependent ones are executed so that one can use the defaults for the major cases and overwrite them if necessary using **\DeclareFontEncoding**.

If **\relax** is used as an argument, the current setting of this default is left unchanged.

This example is used by amsfonts.sty for accent positioning; it changes only the math settings:

```
\DeclareFontEncodingDefaults{\relax}{\def\accentclass@{7}}
```

### 11.4.2 Family, shape and size

`\DeclareFontFamily` ⟨*cdp*⟩ ⟨*family*⟩ ⟨*loading-options*⟩

Declares a font family ⟨*family*⟩ to be available in an encoding scheme ⟨*cdp*⟩. The ⟨*loading-options*⟩ are executed for every font shape of that combination once at the time of loading.

Checks that ⟨*cdp*⟩ was previously declared.

This example refers to the Computer Modern Typewriter in Cork encoding:

```
\DeclareFontFamily{T1}{cmtt}{\hyphenchar\font=-1}
```

`\DeclareFontShape` ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩ ⟨*loading-info*⟩ ⟨*loading-option*⟩

Declares a font shape combination; here ⟨*loading-info*⟩ contains the information that combines sizes with external fonts. The syntax is complex and is described in Section 12 below.

The ⟨*loading-options*⟩ are used as above; they allow overwriting, for this particular shape, of the general loading options made at the family level.

Checks that ⟨*cdp*⟩+⟨*family*⟩ was previously declared via `\DeclareFontFamily`.

Example:

```
\DeclareFontShape{OT1}{cmr}{m}{sl}{%
          <5-8> sub * cmr/m/n
          <8> cmsl8
          <9> cmsl9
          <10> <10.95> cmsl10
          <12> <14.4> <17.28> <20.74> <24.88> cmsl12
          }{}
```

`\DeclareFixedFont` ⟨*cmd*⟩ ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩ ⟨*size*⟩

Declares command ⟨*cmd*⟩ to be a font switch which selects the font that is specified by the attributes ⟨*cdp*⟩, ⟨*family*⟩, ⟨*series*⟩, ⟨*shape*⟩, and ⟨*size*⟩ without any overhead by NFSS table processing.

The NFSS tables are evaluated only when this declaration is made. The font is selected without any adjustments to baselineskip and other surrounding conditions.

This example makes `\picturechar` . select a small dot very quickly:

```
\DeclareFixedFont{\picturechar}{OT1}{cmr}{m}{n}{5}
```

### 11.4.3 Font substitution

`\DeclareErrorFont` ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩ ⟨*size*⟩

Declares ⟨*cdp*⟩⟨*family*⟩⟨*series*⟩⟨*shape*⟩ to be the font shape used in cases where the standard substitution mechanism fails (i.e. would loop).

Also initializes `\f@encoding` ... `\f@baselineskip` with the corresponding values in case a `\selectfont` command is used during the early part of the style file.

Enforces, at `\begin{document}`, that ⟨*cdp*⟩⟨*family*⟩⟨*series*⟩⟨*shape*⟩ is defined via `\DeclareFontShape`.

Example:

```
\DeclareErrorFont{OT1}{cmr}{m}{n}{10}
```

`\DeclareFontSubstitution` ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩

Declares the default values for font substitution which will be used when a font has to be loaded but cannot be found with the current attributes. These are local to the encoding scheme because the encoding scheme is never substituted! They are tried in the order ⟨*shape*⟩ then ⟨*series*⟩ and finally ⟨*family*⟩.

If no defaults are set up for an encoding, the values given by **\DeclareErrorFont** are used.

Enforces, at `\begin{document}`, that ⟨*cdp*⟩⟨*family*⟩⟨*series*⟩⟨*shape*⟩ is defined via **\DeclareFontShape**.

Example:

```
\DeclareFontSubstitution{T1}{cmr}{m}{n}
```

`\fontsubfuzz` = ⟨*dimen*⟩

Parameter that is used to decide whether or not to produce a terminal warning if a font size substitution takes place. If the difference between the requested and the chosen size is less than **\fontsubfuzz** the warning is only written to the transcript file. The default value is **0.4pt**.

### 11.4.4 Size functions

`\DeclareSizeFunction` ⟨*name*⟩ ⟨*code*⟩

Declares a size-function ⟨*name*⟩ for use in **\DeclareFontShape** commands. The interface is still lousy but then there should be no real need to a define new size functions.

The ⟨*code*⟩ is executed when the size or size-range in **\DeclareFontShape** matches the requested user size.

The arguments of the size-function are automatically parsed and placed into **\mandatory@arg** and **\optional@arg** for inspection. Also available, of course, is **\f@size**, which is the user requested size.

To signal success ⟨*code*⟩ has to define the command **\external@font** to contain the external name and any scaling options of the font to be loaded.

This example sets up the 'empty' size function (simplified):

```
\DeclareSizeFunction{}
        {\edef\external@font{\mandatory@arg\space at\f@size}
```

The system provides the following functions.

'' **(empty)** Load the external font (in ⟨*fontarg*⟩) at user requested size. If ⟨*optarg*⟩ present, used as scale-factor.

**s** Like the empty function but without terminal warnings, only loggings.

**gen** Generates the external font from the mandatory arg followed by the user requested size, e.g. `<8> <9> <10> gen * cmtt`

**sgen** Like the 'gen' function but without terminal warnings, only loggings.

**sub** Tries to load a font from a different fontshape declaration given by ⟨*fontarg*⟩ in the form ⟨*family*⟩/⟨*series*⟩/⟨*shape*⟩.

**ssub** Silent variant of 'sub', only loggings.

**subf** Like the empty function but issues a warning that it has to substitute the external font ⟨*fontarg*⟩ because the desired font shape was not available in the requested size.

**ssubf** Silent variant of 'subf', only loggings.

**fixed** Load font as is (disregarding user size) from ⟨*fontarg*⟩. If present ⟨*optarg*⟩ denotes the "at ... pt" size to be used.

**sfixed** Silent variant of 'fixed', only loggings.

### 11.4.5 Preloading

`\DeclarePreloadSizes` ⟨*cdp*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩ ⟨*size-list*⟩

Specifies the font (.tfm) files that should be loaded by the format.
Example:

```
\DeclarePreloadSizes{OT1}{cmr}{m}{sl}{10,10.95,12}
```

## 11.5 Naming conventions

### 11.5.1 Math alphabets

Math alphabet commands all start with `\math...`; e.g. `\mathbf`, `\mathcal`, etc. Define your private shorthands in the preamble if desired.

### 11.5.2 Text font changes

The text font changing commands with arguments all start with `\text...`; e.g. `\textbf`, `\textrm`, `\emph`, etc.

### 11.5.3 Encoding schemes

Names for encoding schemes are strings of up to three letters all upper case. Currently the names officially supported by the NFSS2 release are:

| *Encoding* | *Description* | *declared by* |
|---|---|---|
| T1 | TEX text Cork encoding | NFSS |
| OT1 | TEX text as defined by Don (more or less) | NFSS |
| OT2 | TEX text cyrillic UW fonts | — |
| OT3 | IPA University of Washington (AMS encoding) | — |
| OML | TEX math letters (italic) as defined by Don | NFSS |
| OMS | TEX math symbol as defined by Don | NFSS |
| OMX | TEX math extended symbol as defined by Don | NFSS |
| U | Unknown encoding | NFSS |

This table will be extended in the future if the TEX community, e.g. the national user groups, agree on additional encodings.

The following starting letters are reserved for future use by the NFSS maintainers and should not used to declare new encodings: `T` (standard 256-long text encodings), `M` (standard 256-long math encodings), `S` (standard 256-long symbol encodings), `OT` (standard 128-long text encodings), `OM` (standard 128-long math encodings).

Encoding schemes which are local to a site should start with `L`. Please do not use other starting letters for non-portable encodings. Instead, please inform us if a new standard encoding should be added to a later release.

### 11.5.4 Font families

Font family names should contain up to five lower case letters.

### 11.5.5 Font series

Font series names should contain up to four lower case letters.

### 11.5.6 Font shapes

Font shapes should contain up to two letters lower case.

### 11.5.7 Symbol fonts

Names for symbol fonts are built from lower and upper case letters with no restriction.

NFSS defines the following names for symbol fonts:

| | | |
|---|---|---|
| `operators` | fam 0 | font for operator names (sin, lim, etc.) |
| `letters` | fam 1 | where the normal letters come from |
| `symbols` | fam 2 | normal size symbols |
| `largesymbols` | fam 3 | large size symbols |

# 12 Font file loading information

The information which tells LaTeX exactly which font (`.tfm`) files to load is contained in the ⟨*loading-info*⟩ part of a `\DeclareFontShape` declaration. This part consists of one or more ⟨*fontshape-decl*⟩s, each of which has the following form:

⟨*fontshape-decl*⟩ : : = ⟨*size-info*⟩ ⟨*font-info*⟩
⟨*size-info*⟩ : : = "<" ⟨*number-or-range*⟩ ">"
⟨*font-info*⟩ : : = [ ⟨*size-function*⟩ "*" ] "[" ⟨*optarg*⟩ "]" ⟨*fontarg*⟩
| ⟨*fontarg*⟩

The ⟨*number-or-range*⟩ denotes the size or size-range for which this entry applies. If it contains a hyphen char it is a range: lower bound on the left (if missing, zero implied), upper bound to the right of hyphen (if missing, `\infty` implied). For ranges, the upper bound is *not* included in the range and the lower bound is.

Examples:

| | | |
|---|---|---|
| `<10>` | simple size | 10pt only |
| `<-8>` | range | all sizes less than 8pt |
| `<8-14.4>` | range | all sizes greater than or equal to 8pt but less than 14.4pt |
| `<14.4->` | range | all sizes greater than or equal 14.4pt |

If more than one ⟨*number-or-range*⟩ entry follows without any intervening ⟨*font-info*⟩, they all share the next ⟨*font-info*⟩.

The ⟨*size-function*⟩, if present, handles the use of ⟨*font-info*⟩. If not present, the 'empty' ⟨*size-function*⟩ is assumed.

All the ⟨*size-info*⟩s are inspected in the order in which they appear in the font shape declaration. If a ⟨*size-info*⟩ matches the requested size, its ⟨*size-function*⟩ is executed. If `\external@font` is non-empty afterwards this process stops, otherwise the next ⟨*size-info*⟩ is inspected. (See also `\DeclareSizeFunction`.)

If this process does not lead to a non-empty **\external@font**, NFSS tries the nearest simple size.

## 12.1 Utilities

`\hexnumber@` ⟨*num*⟩

For ⟨*num*⟩ in the range $0 \leq \langle num \rangle < 16$, converts ⟨*num*⟩ into a single hex digit. It is completely expandable and thus can be used to construct longer hex numbers in an emergency.

Example:

```
\xdef\yen{\noexpand\mathhexbox\hexnumber@\symAMSa 55 }
```

`\f@warn@break`

This command is used within warning messages generated by NFSS and is defined to produce a new line beginning with the string "**Font**" (plus a suitable number of spaces). This means that it is fairly easy to extract all NFSS related messages from the transcript file. If this is not desired, redefine it to be less conspicuous; for example:

```
\def\f@warn@break{^^J\@spaces\@spaces\@spaces}
```

## 12.2 Low-level interfaces

The low-level commands used to select a text font are as follows.

`\fontencoding` ⟨*encoding*⟩
`\fontfamily` ⟨*family*⟩
`\fontseries` ⟨*series*⟩
`\fontshape` ⟨*shape*⟩
`\fontsize` ⟨*size*⟩ ⟨*baselineskip*⟩

Each of these commands sets the value of just one of the font attributes; **\fontsize** also sets the value of **\baselineskip**. The actual font in use is not altered by these commands, but the current attributes are used to determine which font to use after the the next **\selectfont** command.

`\selectfont`

Selects a text font, based on the current values of the font attributes.

`\usefont` ⟨*encoding*⟩ ⟨*family*⟩ ⟨*series*⟩ ⟨*shape*⟩

A short hand for the equivalent `\font`... commands followed by a call to `\selectfont`.

The current values of the font attributes are held internally in these locally set macros:

| | |
|---|---|
| `\f@encoding` | current encoding value |
| `\f@family` | current family value |
| `\f@series` | current series value |
| `\f@size` | current size value (in pt, without an explicit dimension) |
| `\f@baselineskip` | current baselineskip value |
| `\tf@size` | current text size for math (normally the same as `\f@size`) |
| `\sf@size` | current script size for math |
| `\ssf@size` | current scriptscript size for math |

As an example of their use, this will set the size to 12 without changing the baselineskip:

```
\fontsize{12}{\f@baselineskip}
```