LaTeX 2ε for class and package writers
DRAFT

Preliminary draft June 1994

Contents

1 Introduction

This document is an introduction to writing classes and packages for LaTeX, with special attention given to upgrading existing LaTeX 2.09 packages to LaTeX 2_ε. The latter subject will also be covered in an article by Johannes Braams to be published in TUGboat later this year.

1.1 Writing classes and packages for LaTeX 2_ε

LaTeX is a document preparation system that enables the document writer to concentrate on the contents of their text, without bothering too much about the formatting of it. For example, chapters are indicated by `\chapter{`⟨*title*⟩`}` rather than by selecting 18pt bold.

The file that contains the information about how to turn logical structure (like '`\chapter`') into formatting (like '18pt bold ragged right') is a *document class*. In addition, some features (such as colour or included graphics) are independent of the document class, and are contained in *packages*.

One of the largest differences between LaTeX 2.09 and LaTeX 2_ε is in the commands used to write packages and classes. In LaTeX 2.09, there was very little support for writing `.sty` files, and so writers had to resort to using low-level commands.

LaTeX 2_ε provides high-level commands for structuring packages. It is also much easier to build classes and packages on top of each other, for example writing a local technical report class based on `article`.

1.2 Overview

This document contains an overview of how to write classes and packages for LaTeX. It does *not* introduce all of the commands necessary to write packages, which are described in *LaTeX: A Document Preparation System* and *The LaTeX Companion*. But it does describe the new commands for structuring classes and packages.

Section 2 contains some general advice about writing classes and packages. It describes the difference between classes and packages, the command naming conventions, the use of docstrip, how TEX's primitive file and box commands interact with LATEX, and some hits about general LATEX style.

Section 3 describes the structure of classes and packages. This includes building classes and packages on top of other classes and packages, declaring options and declaring commands. It also contains example classes.

Section 4 lists the new class and package commands.

Section 5 gives advice on how to upgrade existing LATEX 2.09 classes and packages to LATEX 2_ε.

1.3 Further information

For a general introduction to LATEX, including the new features of LATEX 2_ε, you should read *LATEX: A Document Preparation System* by Leslie Lamport [3].

A more detailed description of the new features of LATEX, and an overview of over 150 packages, is to be found in *The LATEX Companion* by Michel Goossens, Frank Mittelbach and Alexander Samarin [1].

The LATEX font selection scheme is based on TEX, which is described in *The TEXbook* by Donald E. Knuth [2].

There are a number of documentation files which accompany every copy of LATEX. A copy of *LATEX News* will come out with each six-monthly release of LATEX, and is found in the files `ltnews*.tex`. The author's guide *LATEX 2_ε for Authors* describes the new LATEX document features, and is kept in `usrguide.tex`. *LATEX 2_ε Font Selection* describes the LATEX font selection scheme for class- and package-writers, and is in `fntguide.tex`.

We are gradually turning the source code for LATEX into a LATEX document *LATEX: the program*. This document includes an index of LATEX commands and can be typeset from `source2e.tex`.

For more information about TEX and LATEX, please contact your local TEX Users Group, or the international TEX Users Group, P. O. Box 869, Santa Barbara, CA 93102-0869, USA, Fax: +1 805 963 8358, E-mail: tug@tug.org.

2 Writing classes and packages

This section gives some general points about writing LATEX classes and packages.

2.1 Is it a class or a package?

The first thing to do when you want to put some new LaTeX commands in a file is to decide whether it should be a document class or a package. The rule of thumb is *if the commands could be used with any document class, then make them a package, and if not, make them a class.*

For example, the `proc` document class changes the appearance of the `article` document class. It is of no use with any other document class, so we have `proc.cls` rather than `proc.sty`.

The `graphics` package, however, provides commands for including images into a LaTeX document. Since these commands can be used with any document class, we have `graphics.sty` rather than `graphics.cls`.

A company might have a local `ownlet` class for printing letters with their own headed notepaper. Such a class would build on top of the existing `letter` class, but cannot be used with any other document class, so we have `ownlet.cls` rather than `ownlet.sty`.

2.2 Command names

LaTeX has three types of command.

There are the author commands, such as `\section`, `\emph` and `\times`. Most of these have short names, all in lower case.

There are the class and package writer commands, such as `\InputIfFileExists`, `\RequirePackage` and `\PassOptionsToClass`. Most of these have long mixed-case names.

Finally, there are the internal commands used in the LaTeX implementation, such as `\@tempcnta`, `\@ifnextchar` and `\@eha`. Most of these commands contain `@` in their name, which means they cannot be accessed in documents, only in classes and packages.

Unfortunately, for historical reasons, the distinction between these commands is often blurred. For example, `\hbox` is an internal command which should only be used in the LaTeX kernel, whereas `\m@ne` is the constant -1 and could have been `\MinusOne`.

The rule of thumb still applies: if a command has `@` in its name, then it is not part of the supported LaTeX language, and its behaviour may change in future releases. If a command is mixed-case, or is described in *LaTeX: A Document Preparation System*, then you can rely on future releases of LaTeX 2_ε supporting the command.

2.3 Using docstrip

If you are going to write a large class or package for LaTeX, you should consider using the `docstrip` software which comes with LaTeX.

LaTeX classes and packages written using `docstrip` can be processed in two ways: they can be run through LaTeX, to produce documentation, and they can be processed with `docstrip` to produce the `.cls` or `.sty` file.

The `docstrip` software can automatically generate indexes of definitions, indexes of command use, and change log lists. It is very useful for maintaining and documenting large TeX sources.

The LaTeX kernel itself is a `docstrip` document—you can read the source code as one long document by running LaTeX on `source2e.tex`.

For more information on `docstrip`, consult the `docstrip` documentation or *The LaTeX Companion*.

2.4 Loading other files

LaTeX provides the commands `\LoadClass` and `\RequirePackage` for using classes or packages inside other classes or packages. We highly recommend you use them, rather than the primitive `\input` command, for a number of reasons.

Files loaded with `\input` ⟨*filename*⟩ will not be listed in the `\filecontents` list.

If a package is requested more than once with `\RequirePackage` or `\usepackage` it will only be loaded once. If it is loaded with `\input`, then it will be loaded more than once, which may waste time and memory, and produce strange results.

If a package provides option-processing, then this can produces strange results if the package is `\input` rather than loaded with `\usepackage` or `\RequirePackage`.

If the package `foo.sty` loads the package `baz.sty` with `\input baz.sty`, then the user will get a warning:

```
LaTeX Warning: You have requested package ‘foo’,
but the package provides ‘baz’.
```

So using `\input` to load packages is not a good idea.

Unfortunately, if you are upgrading a class file `myclass`, you have to make sure that any old files which contain `\input myclass.sty` still work. This is particularly true of the standard classes `article`, `book` and `report`, since a lot of existing LaTeX 2.09 document styles contain `\input article.sty`. The approach which we took was to provide minimal files `article.sty`, `book.sty` and `report.sty`, which load the appropriate class files. For example, `article.sty` contains:

```
\NeedsTeXFormat{LaTeX2e}
\@obsoletefile{article.sty}{article.cls}
\LoadClass{article}
```

You may wish to do the same, or if you think that it's safe to do so, you may decide to just remove `myclass.sty`.

2.5 Box commands and colour

Even if you do not intend to use colour in your own documents, by taking note of the points in this section you can ensure that your class or package is compatible with the color package. This may benefit other people using your class who may have access to colour printers.

The simplest way to ensure 'colour safety' is to always use LaTeX box commands rather than TeX primitives, that is use `\sbox` rather than `\setbox`, `\mbox` rather than `\hbox` and `\parbox` or `\minipage` rather than `\vbox`. The LaTeX box commands have new options which mean they are now as powerful as the TeX primitives.

As an example of what can go wrong, consider that in a `{\ttfamily` ⟨*text*⟩`}` the font is restored just *before* the `}`, whereas in the similar looking `{\color{green}` ⟨*text*⟩`}` the colour is restored just *after* the final `}`. Normally this distinction does not matter at all, but consider a primitive TeX box assignment such as:

```
\setbox0=\hbox{\color{green} ⟨text⟩}
```

Now the colour-restore occurs after the `}` and so is *not* stored in the box. Exactly what bad effects this can have depends on how colour is implemented, but it can range from getting the wrong colours in the rest of the document, to causing errors in the dvi-driver used to print the document.

Also of interest is the command `\normalcolor`. Again this is normally just `\relax` but you can use it rather like `\normalfont` to set regions of the page such as captions or section headings to the 'main document colour'.

2.6 General style

Apart from the changes you need to make to get your document class or package running again there are also a few changes that we encourage you to make.

We consider it good practice, when writing packages and classes, to use LaTeX commands as much as possible. So instead of using `\def...` we recommend using one of `\newcommand`, `\renewcommand` or `\providecommand`. Doing that makes it less likely that you inadvertently redefine a command, giving unexpected results.

When you define an environment use `\newenvironment` or `\renewenvironment` instead `\def\foo{...}` and `\def\endfoo{...}`.

If you need to set or change the value of a ⟨*dimen*⟩ or ⟨*skip*⟩ register, use `\setlength`.

To manipulate boxes, use LaTeX commands such as `\sbox`, `\mbox` and `\parbox` rather than `\setbox`, `\hbox` and `\vbox`.

The advantage of this practice is that your code is more readable and, more important, that it is less likely to break when future versions of LaTeX are made available.

Some packages and document styles had to redefine the command `\begin{document}` or `\end{document}` to acheive their goal. This is no longer necessary. You can now use the 'hooks' `\AtBeginDocument` and `\AtEndDocument`. Again, using these hooks makes it less likely that your code breaks when future versions of LaTeX arrive. It makes it also more likely that your package can work together with someone else's.

Use `\PackageError`, `\PackageWarning` or `\PackageInfo` (or the equivalent class commands) rather than `\@latexerr`, `\@warning` or `\wlog`.

It is very useful for exchanging files if your files are as portable as possible. They should only contain visible 7-bit text, and the filenames should be eight letters (plus the three letter extension). It is also useful if local classes or packages have a common prefix, for example the University of Nowhere classes might begin with `unw`. This helps to avoid every University having its own thesis class, all called `thesis.cls`.

It is still possible to declare options by defining `\ds@`⟨*option*⟩ and calling `\@options`, but we recommend using the `\DeclareOptions` and `\ProcessOptions` commands instead. These are more powerful and use less memory. So rather than saying:

```
\def\ds@draft{\overfullrule 5pt}
\@options
```

you should say:

```
\DeclareOption{draft}{\setlength{\overfullrule}{5pt}}
\ProcessOptions
```

If you rely on some features of the LaTeX kernel, or on a package, please specify the release-date you need. For example, the package error commands were introduced in the June 1994 release, so if you use them, you should say:

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

If you are upgrading an existing LaTeX 2.09 style file, we recommend freezing the 2.09 version, and no longer maintaining it. Experience with the various dialects of LaTeX which existed in the early 1990's suggests that maintaining packages for different versions of LaTeX is almost impossible. We recommend maintaining classes and packages only for LaTeX 2$_\varepsilon$, and not for obsolete versions of LaTeX.

3 The structure of a class or package

LaTeX 2$_\varepsilon$ classes and packages have more structure than LaTeX 2.09 style files did. The outline of a class or package is:

identification The file says that it is a LaTeX 2$_\varepsilon$ package or class, and gives a short description of itself.

declarations The file declares some commands, and can also load other files. Usually these commands will be just for defining commands used in the options.

options The file declares and processes its options.

more declarations This is where the file does most of its work, declaring new variables, commands, fonts, and loading other files.

3.1 Identification

The first thing a class or package does is identify itself. Packages do this by saying:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{⟨package⟩}[⟨date⟩ ⟨other information⟩]
```

for example:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{latexsym}[1994/06/01 Standard LaTeX package]
```

Classes do this by saying:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{⟨class-name⟩}[⟨date⟩ ⟨other information⟩]
```

for example:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{article}[1994/06/01 Standard LaTeX class]
```

The ⟨*date*⟩ should be given in the form 'YYYY/MM/DD'. This date is checked whenever a user specifies a date in their `\documentclass` or `\usepackage` command. For example, if a user said:

```
\documentclass{article}[1995/12/23]
```

then they would get a warning that their copy of `article` was out of date.

The description of a class is printed out when the class is used. The description of a package is put into the log file. These descriptions are also produced by `\listfiles`.

3.2 Using classes and packages

The first major difference between LaTeX 2.09 style files and LaTeX 2_ε packages and classes is that LaTeX 2_ε supports *modularity*, that is building files from small building-blocks rather than large single files.

A LaTeX package or class can load a package by saying:

```
\RequirePackage[⟨options⟩]{⟨package⟩}[⟨date⟩]
```

for example:

```
\RequirePackage{ifthen}[1994/06/01]
```

This command has the same syntax as the author command `\usepackage`. It allows packages or classes to use features provided by other packages. For example, by loading the `ifthen` package, a package writer can use the 'if... then... else...' commands provided by that package.

A LaTeX class can load another class by saying:

```
\LoadClass[⟨options⟩]{⟨class-name⟩}{⟨date⟩}
```

for example:

```
\LoadClass[twocolumn]{article}
```

This command has the same syntax as the author command `\documentclass`. It allows classes to be based on the syntax and appearance of another class. For example, by loading the `article` class, a class writer only has to change the bits of `article` they don't like, rather than writing a new class from scratch.

3.3 Declaring options

The other major difference between LaTeX 2.09 styles and LaTeX 2_ε packages and classes is in option handling. Packages and classes can now declare options, which can be used by authors, for example the `twocolumn` option to the `article` class.

An option is declared by saying:

```
\DeclareOption{⟨option⟩}{⟨code⟩}
```

for example the `dvips` option to the `graphics` package is implemented as:

```
\DeclareOption{dvips}{\input{dvips.def}}
```

This means that when an author says `\usepackage[dvips]{graphics}`, the file `dvips.def` is loaded. As another example, the `a4paper` option is declared in the `article` class to set the `\paperheight` and `\paperwidth` lengths:

```
\DeclareOption{a4paper}{%
   \setlength{\paperheight}{297mm}%
   \setlength{\paperwidth}{210mm}%
}
```

Sometimes a user will request an option which the class or package has not explicitly declared. By default this will produce a warning (for classes) or error (for packages), but this behaviour can be altered, by saying:

```
\DeclareOption*{⟨code⟩}
```

for example to make the package `fred` produce a warning rather than an error, you could say:

```
\DeclareOption*{%
   \PackageWarning{fred}{Unknown option `\CurrentOption'}%
}
```

then if an author says `\usepackage[foo]{fred}` they will get a warning `Package fred Warning: Unknown option 'foo'.` As another example, the `fontenc` package troes to load a file ⟨*encoding*⟩`enc.def` whenever the ⟨*encoding*⟩ option is used. This can be done by saying:

```
\DeclareOption*{%
   \input{\CurrentOption enc.def}%
}
```

It is possible to pass options on to another package or class, using the command `\PassOptionsToPackage` or `\PassOptionsToClass`. For example, to pass every unknown option on to the `article` class, you can say:

```
\DeclareOption*{%
   \PassOptionsToClass{\CurrentOption}{article}%
}
```

If you do this, you should make sure you load the class, otherwise the options will never be processed!

So far, we have only seen how to declare options, and not how to execute them. To process the options which the file was called with, you say:

```
\ProcessOptions
```

This executes the ⟨*code*⟩ for each option that was declared (see Section 4.3 for details of how this is done).

For example, if the `jane` package says:

```
\DeclareOption{foo}{\typeout{Saw foo.}}
\DeclareOption{baz}{\typeout{Saw baz.}}
\DeclareOption*{\typeout{What's \CurrentOption?}}
\ProcessOptions
```

and an author says `\usepackage[foo,bar]{jane}`, then they will see the messages `Saw foo` and `What's bar?`

3.4 Declarations

Most of the work of a class or package is in defining new commands, or changing the appearance of documents. This is done in the body of the package, using commands such as `\newcommand`, `\setlength` and `\setbox`.

However, there are some new commands for helping class and package writers. These are described in detail in Section 4.

There are three definitions that every class *must* provide. These are `\normalsize`, `\textwidth` and `\textheight`. So a minimal document class file is:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{minimal}[1994/04/01 Minimal class]
\renewcommand{\normalsize}{\fontsize{10}{12}\selectfont}
\setlength{\textwidth}{6.5in}
\setlength{\textheight}{8in}
```

However, most classes will provide more than this!

3.5 Example: a local letter class

A company may have its own letter class, for setting letters in the company style. This section shows a simple implementation of such a class, although a real class would need more structure.

The class begins by announcing itself as `neplet.cls`.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{neplet}[1995/04/01 NonExistent Press letter class]
```

Then it passes any options on to the `letter` class, which is loaded with the `a4paper` option.

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{letter}}
\ProcessOptions
\LoadClass[a4paper]{letter}
```

Then it uses the comany letter head. This is done by redefining the `firstpage` page style, since this is the page style that is used on the first page of letters.

```
\renewcommand{\ps@firstpage}{%
   \renewcommand{\@oddhead}{⟨letterhead goes here⟩}%
   \renewcommand{\@oddfoot}{⟨letterfoot goes here⟩}%
}
```

And that's it!

3.6 Example: a newsletter class

A simple newsletter can be set in LaTeX, using a variant of the `article` class. The class begins by announcing itself as `smplnews.cls`.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{smplnews}[1995/04/01 The Simple News newsletter class]
```

Then it passes any options on to the `article` class, apart from the `onecolumn` option, which is switched off, and the `green` option, which sets the headline in green.

```
\newcommand{\headlinecolor}{\normalcolor}
\DeclareOption{onecolumn}{\OptionNotUsed}
\DeclareOption{green}{\renewcommand{\headlinecolor}{\color{green}}}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\ProcessOptions
\LoadClass[twocolumn]{article}
```

Since we're using colour, we load the `color` package. We don't specify a device driver, which should be specified by the user of the `smplnews` class.

```
\RequirePackage{color}
```

The class then redefines `\maketitle` to produce the title in 72pt Helvetica bold oblique, in the appropriate colour.

```
\renewcommand{\maketitle}{%
   \twocolumn[%
      \fontsize{72}{80}\fontfamily{phv}\fontseries{b}%
      \fontshape{sl}\selectfont\headlinecolor
      \@title
   ]%
}
```

It redefines `\section` and switches off section numbering.

```
\renewcommand{\section}{%
   \@startsection
      {section}{1}{0pt}{-1.5ex plus -1ex minus -.2ex}%
      {1ex plus .2ex}{\large\sffamily\slshape\headlinecolor}%
}
\setcounter{secnumdepth}{0}
```

In practice, a class would need more than this: it ought to set the page sizes, provide commands for issue numbers, authors of articles, page styles and so on, but this skeleton gives a start. The `ltnews` class is not much more complex than this one.

4 Commands for class and package writers

This section describes each of the new commands for class and package writers. You should also read the commands described in *LaTeX: A Document Preparation System*, *The LaTeX Companion* and *LaTeX 2ε for Authors*.

4.1 Identification

The first group of commands to be discussed are the ones that are used in identifying your class or package file.

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

This command tells TeX that it has to be processed using a format with name ⟨*format-name*⟩. With ⟨*release-date*⟩ one can specify the earliest release date of

the format that should still work. When the release date of the format is older than the one specified a warning will be generated. The standard ⟨*format-name*⟩ is `LaTeX2e`. The date, if present, must be in the form YYYY/MM/DD.

Example:

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

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

This declares that the current file contains the definitions for the document class ⟨*class-name*⟩ or package ⟨*package-name*⟩. The optional ⟨*release-info*⟩ contains the release date in the form YYYY/MM/DD, followed by the version of the file, optionally followed by a short description. 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}[1994/06/01 v1.0 Standard LaTeX class]
\ProvidesPackage{ifthen}[1994/06/01 v1.0 Standard LaTeX package]
```

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

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

Example:

```
\ProvidesFile{T1enc.def}[1994/06/01 Standard LaTeX file]
```

4.2 Loading files

This group of commands can be used to build your own document class or package upon existing classes or packages.

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

With this command, packages and classes can load other packages and classes. Its use is the same as the author command `\usepackage`.

Example:

```
\RequirePackage{ifthen}[1994/06/01]
```

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

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

Example:

```
\LoadClass{article}[1994/06/01]
```

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

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.

4.3 Option handling

The following commands deal with the declaration and handling of options to document classes and packages.

`\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*⟩`}`

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.

`\CurrentOption`

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

`\ProcessOptions`

These commands execute the ⟨*code*⟩ for each selected option.

We shall first describe how `\ProcessOptions` works in packages, and then describe classes.

To understand in detail what `\ProcessOptions` does, you have to know the difference between *local* and *global* options. Local options are those which have been explicitly passed to the package with `\PassOptionsToPackage`, `\usepackage[`⟨*options*⟩`]` or `\RequirePackage[`⟨*options*⟩`]`. For example if the `fred` package is called with:

```
\documentclass[german,twocolumn]{article}
\PassOptionsToPackage{dvips}{fred}
\RequirePackage[errorshow]{fred}
```

then `fred`'s local options are `german`, `errorshow` and `dvips`, and the only global option is `twocolumn`. Then when `\ProcessOptions` is called, the following happen:

- For each option declared by `\DeclareOption`, it looks to see if that option has been requested. If it has, the corresponding code is executed.
- *Then* for each remaining option in the *local* option list, `\ds@`⟨*option*⟩ is executed if it exits, otherwise the default option is executed. If no default option has been declared, then an error is raised.

For example, if `fred.sty` contains:

```
\DeclareOption{dvips}{\typeout{DVIPS}}
\DeclareOption{german}{\typeout{GERMAN}}
\DeclareOption{french}{\typeout{FRENCH}}
\DeclareOption*{\PackageWarning{fred}{Unknown `\CurrentOption'}}
\ProcessOptions
```

then the result will be:

```
DVIPS
GERMAN
Package fred Warning: Unknown ‘errorshow’.
```

Note that the `dvips` option is executed before the `german` option, because that is the order they are declared in `fred.sty`. Also note that the `errorshow` option produces a warning, but the `twocolumn` option does not, because `twocolumn` is a global option.

In classes, `\ProcessOptions` is the same, except that *all* options are local, and that the default value for `\DeclareOption*` is `\OptionNotUsed` rather than an error.

`\ProcessOptions*`

Like `\ProcessOptions` but executes the 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 document styles to LaTeX 2_ε class files.

4.4 Delaying code

`\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 or 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}` or `\end{document}`.

Note that the ⟨*code*⟩ specified in the argumnent to `\AtEndDocument` is executed *before* any leftover floating environments are processed. If you need your code to be executed after that you may want to include a `\clearpage` in ⟨*code*⟩.

4.5 Safe Input Commands

These commands deal with reading a file; they have been implemented in such a way that the case that the file doesn't existed can be handled in a user-friendly way.

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

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

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

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

4.6 Generating errors

These commands are used by third party classes and packages to report errors, or to provide information to authors.

`\ClassError {`⟨*class-name*⟩`} {`⟨*error*⟩`} {`⟨*help*⟩`}`
`\PackageError {`⟨*package-name*⟩`} {`⟨*error*⟩`} {`⟨*help*⟩`}`

These produce an error message. The ⟨*error*⟩ is displayed, and the error prompt is shown. If the user types `h`, they will be shown the ⟨*help*⟩ information.

Within the ⟨*error*⟩ or ⟨*class-name*⟩, `\protect` can be used to stop a command from expanding, `\MessageBreak` causes a line-break, and `\space` prints a space. For example:

```
\newcommand{\foo}{FOO}
\PackageError{ethel}{%
   Your hovercraft is full of eels,\MessageBreak
   and \protect\foo\space is \foo
}{%
   Oh dear.\MessageBreak Something's gone wrong.
}
```

produces:

```
! Package ethel Error: Your hovercraft is full of eels,
(ethel)                and \foo is FOO.
See the ethel package documentation for explanation.
```

If the user types `h`, they will be shown:

```
Oh dear.
Something's gone wrong.
```

`\ClassWarning {`⟨*class-name*⟩`} {`⟨*warning*⟩`}`
`\PackageWarning {`⟨*package-name*⟩`} {`⟨*warning*⟩`}`
`\ClassWarningNoLine {`⟨*class-name*⟩`} {`⟨*warning*⟩`}`
`\PackageWarningNoLine {`⟨*package-name*⟩`} {`⟨*warning*⟩`}`

These commands are similar, but produce warnings on the screen. The `Warning` versions show the line number where the warning ocurred, and the `NoLine` versions do not.

`\ClassInfo {`⟨*class-name*⟩`} {`⟨*info*⟩`}`
`\PackageInfo {`⟨*package-name*⟩`} {`⟨*info*⟩`}`

These commands are similar, but produce information in the log file.

`\MessageBreak`

Produces a line-break in an error, warning or info message.

4.7 Defining commands

LaTeX 2$_\varepsilon$ contains a number of new commands that are meant to be used in class and package files.

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

This command takes the same arguments as `\newcommand` but it declares a robust command, even if the ⟨*definition*⟩ is fragile. You can use this command to define new commands, or redefine existing commands.

Example:

```
\DeclareRobustCommand{\seq}[2][n]{%
   \ifmmode
      #1_{1}\ldots#1_{#2}%
   \else
      \PackageWarning{fred}{You can't use \protect\seq\space in text}%
   \fi
}
```

Now `\seq` can be used in moving arguments, even though `\ifmmode` cannot, for example:

```
\section{Stuff about sequences $\seq{x}$}
```

`\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.

4.8 Layout parameters

`\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.

5 Upgrading LaTeX 2.09 classes and packages

This section describes how to upgrade any existing LaTeX styles to packages or classes.

5.1 Try it first!

The first thing you should do with an old style is try to run a LaTeX 2$_\varepsilon$ document that uses it unmodified. This assumes that you have a suitable test set that tests all functionality provided by the style file. (If you haven't, now is the time to make one!) Please run the test document in both LaTeX 2$_\varepsilon$ native mode, and LaTeX 2.09 compatibility mode, since some old styles will only work in compatibility mode.

Many existing style files will run with LaTeX 2$_\varepsilon$ without any modification. If it does run, please enter a note into the file that you have checked that it runs, and distribute it to your users. You might like to take the opportunity to make use of the new document structuring commands.

If your style file does not work with LaTeX 2$_\varepsilon$, there are two likely reasons. LaTeX now has a robust, well-defined designer's interface for selecting fonts, which is very different from the LaTeX 2.09 internals. And your style file may have used some LaTeX 2.09 internal commands which have changed, or which have been removed.

5.2 Font commands

Some commands are now defined by the document class rather than by the LaTeX kernel. If you are upgrading a LaTeX 2.09 document style, you should add definitions for these commands.

`\rm \sf \tt \bf \it \sl \sc`

The LaTeX 2.09 font selection commands are now defined in the document class. They are defined in the kernel to produce an error message.

`\normalsize`
`\@normalsize`

The command `\@normalsize` is retained for compatibility with LaTeX 2.09 packages which may have used it, but it is now defined in the kernel by:

```
\newcommand{\@normalsize}{\normalsize}
```

This means that classes should now define `\normalsize` rather than `\@normalsize`, for example:

```
\renewcommand{\normalsize}{\fontsize{10}{12}\selectfont}
```

Note that `\normalsize` is defined by the LaTeX kernel to be an error message, whereas the other size-changing commands `\tiny`, `\footnotesize`, `\small`, `\large`, `\Large`, `\LARGE`, `\huge` and `\Huge` are not defined at all. This means you should use `\renewcommand` for `\normalsize` and `\newcommand` for the other commands.

5.3 Obsolete commands

Some packages will not work with LaTeX 2_ε, normally because they relied on an internal LaTeX command which was never supported, and has now changed, or been removed.

In many cases there will now be a robust, high-level means of achieving what previously required low-level commands. Please consult Section 4 to see if you can use the LaTeX 2_ε class and package writers commands.

Too many of the internal commands of LaTeX 2.09 have been re-implemented to list here. We will list some of the more important commands which are no longer supported.

```
\tenrm \elvrm \twlrm ...
\tenbf \elvbf \twlbf ...
\tensf \elvsf \twlsf ...
    ⋮
```

The seventy pre-loaded LaTeX 2.09 fonts are now no longer pre-loaded. If your package uses them, then *please* replace them with new font commands described in *LaTeX 2ε Font Selection*. For example the command `\twlsf` could be replaced by:

```
\fontsize{12}{14}\sffamily
```

Another possibility is to use the `rawfonts` package, described in *LaTeX 2ε for Authors*.

`\prm`, `\pbf`, `\ppounds`, `\pLaTeX` ...

LaTeX 2.09 used commands beginning with `\p` for 'protected' commands. For example, `\LaTeX` was defined to be `\protect\pLaTeX`, and `\pLaTeX` produced the LaTeX logo. This made `\LaTeX` robust, even though `\pLaTeX` was not. These commands have now been reimplemented using `\DeclareRobustCommand` (described in Section 4.7). If your package redefined one of the `\p`-commands, you should replace the redefinition by one using `\DeclareProtectedCommand`.

`\vpt` `\vipt` `\viipt` ...

These commands used to be the internal size-selecting commands in LaTeX 2.09. They are still supported in LaTeX 2.09 compatibility mode, but not in native mode. Please use the command `\fontsize` instead (see *LaTeX 2ε Font Selection* for details) for example replace `\vpt` with `\fontsize{5}{6}\selectfont`.

`\footheight`
`\@maxsep`
`\@dblmaxsep`

These parameters are not used by LaTeX 2ε, and so have been removed, except for in LaTeX 2.09 compatibility mode. Classes should no longer set them.

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[2] Donald E. Knuth. *The TeXbook*. Addison-Wesley, Reading, Massachusetts, 1986. Revised to cover TeX3, 1991.

[3] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, second edition, 1994.