Document Classes and Packages for LaTeX2ε

Johannes Braams
PTT Research, P.O. box 421, 2260 AK Leidschendam, The Netherlands
J.L.Braams@research.ptt.nl

Abstract

The first section of this article describes what document classes and packages are and how they relate to LaTeX 2.09's style files. Then the process of upgrading existing style files for use with LaTeX2ε is described. Finally there is an overview of standard packages and document classes that come with the LaTeX2ε distribution.

Introduction

This article is written for people who have written document styles for LaTeX 2.09 and want to upgrade them for LaTeX2ε. For a description of the new features of the user level commands, see *LaTeX2ε for authors* (in the file `usrguide.tex` in the LaTeX2ε distribution). The details about the interface for class and package writers can be found in *LaTeX2ε for class and package writers* (in the file `clsguide.tex`). The way LaTeX now deals with fonts is described in *LaTeX2ε font selection* (in the file `fntguide.tex`).

What are document classes and packages?

LaTeX is a document preparation system that enables the document writer to concentrate on the contents of his text, without bothering too much about the formatting of it. For instance, whenever he starts a new chapter the formatting of the chapter is defined outside of his document. The file that contains these formatting rules used to be called a 'document style'. Such a document style can have options to influence its formatting decisions. Some of these options are stored in separate files, 'document style option' files. An example of such option files is `fleqn.sty` which was part of the LaTeX 2.09 distribution. This option changes one aspect of the formatting of a document —it makes displayed equations come out flush left instead of centered.

There are also extensions to LaTeX that implement constructs that are not available in the default system, such as `array.sty`. These extensions are also known as 'document style option' files, although they can often be used with many kinds of documents.

To make a better distinction possible between these two kinds of 'options' new names have been introduced for them. What used to be called a 'document style' is now called a 'document class'[1]. Extensions to the functionality of LaTeX are now called 'packages'.

[1] This also gives a possibility to distinguish between documents written for LaTeX 2.09 and documents written for LaTeX2ε.

Options, options, options... Like the document styles of LaTeX 2.09 document classes can have options that influence their behaviour—to select the type size for instance. But with LaTeX2ε it is now also possible for packages to have options. As a consequence there are now two kinds of options, 'local options'—which are only valid for the package or document class they are specified for—and 'global' options which can influence the behaviour of both the document class and one or more packages. As an example of this let's consider a document written in German. The author chooses to use the `babel` package. He also wants to be able to refer to a figure 'on the following page' so he uses the `varioref` package. The preamble of his document might then look like:

```
\documentclass{article}
\usepackage[german]{babel}
\usepackage[german]{varioref}
...
```

As you see the option 'german' was specified twice. Using a 'global option' this preamble could be changed to read:

```
\documentclass[german]{article}
\usepackage{babel}
\usepackage{varioref}
...
```

This way it is known to the document class as well as *all* packages used in the document that the option 'german' is specified.

Command names. This new version of LaTeX comes with a new set of commands. Those LaTeX users who have written their own extensions to LaTeX in the past know that in version 2.09 basically two types of commands existed, namely "internal" commands—with '@'-signs in their name—and "user level" commands—without '@'-signs in their name.

LaTeX2ε has also commands that have both upper- and lowercase letters in their name. Those commands are part of the interface for package and

`cls`	A file containing a document class
`clo`	A file containing an external option to a document class
`sty`	A file that contains (part of) a package
`cfg`	An optional file that is looked for at run-time and which can contain customization code
`def`	A file containing definitions that will be read in at runtime.
`ltx`	A file used when building the LaTeX2ε format
`dtx`	Documented source code for `.cls`, `.clo`, `.sty`, `.cfg`, `.def`, and `.ltx` files
`fd`	A font definition file
`fdd`	Documented source code for `.fd` files
`ins`	DOCSTRIP instructions to unpack `.dtx` and `.fdd` files

Table 1: Extensions for LaTeX2ε files

class writers. They are not intended for use in documents, but they are meant to provide an 'easy' interface to some of the internals of LaTeX2ε.

Filenames. The new version of LaTeX introduces a number of new file extensions. This makes it easy to distinguish between files that contain a Document Class, files that contain an external option to a Document Class and files that contain Packages. In table 1 you can find an overview of the extensions that have been introduced. I would suggest that you would stick to the same set of extensions when you upgrade your old `.sty` files.

Upgrading existing 'styles' — general remarks

Is it a class or a package? The first thing to do when you upgrade an existing style file for LaTeX2ε, is to decide whether it should become a document class or a package. Here are a few points which might help you to decide what to do with your `.sty` file.

- Was the original `.sty` file a documentstyle? Then turn it into a document class.
- Was the original `.sty` file meant to be used for a certain type of document? In that case you should consider turning it into a document class, possibly by building on top of an existing class. An example of this is `proc.sty` which is now `proc.cls`.
- Was it just changing some aspects of the way LaTeX does things? In that case you would probably want to turn your `.sty` file into a package.
- Was it adding completely new functionality to LaTeX? Examples of this kind of `.sty` file are files such as `fancyheadings.sty` and `XYpic.sty`. This you most certainly will want to turn into a package for LaTeX2ε.

Style options → packages

Trying it out unchanged. After you've decided to produce a package file, you should first try to run a document that uses your `.sty` file through LaTeX2ε unmodified. This assumes that you have a suitable test set that tests all functionality provided by the `.sty` file. (If you haven't, now is the time to make one!) The experience of the last months has shown that most of the available `.sty` files will run with LaTeX2ε without any modification. Yet if it does run, please enter a note into the file that you have checked that it runs and resubmit it to the archives if it was a distributed file.

Bits that might have failed. Some `.sty` files will need modification before they can be used successfully with LaTeX2ε. Such a modification is needed for instance when you used an internal macro from the old font selection scheme. An example is `\fivrm` which is used by some packages to get a small dot for plotting. The obvious solution for this seems be to include a definition such as:

```
\newcommand{\fivrm}
      {\normalfont
       \fontsize{5}{6.5pt}\selectfont}
```

But that involves a lot of internal processing and may result in long processing times for your documents that use this. For this purpose the command `\DeclareFixedFont` is available. It bypasses a lot of the overhead of the font selection scheme. Using this command the solution becomes:

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

This tells LaTeX that the command `\fivrm` should select a font with OT1 encoding, `cmr` family, medium weight, normal shape and size 5 point.

Pieces of code that might need checking. If your `.sty` file uses commands that used to be part of the way LaTeX used to deal with fonts than your file will almost certainly *not* work. You will have to look in *LaTeX2ε font selection* or *The LaTeX Companion* (Goossens et al. 1994) to find out the details about what needs to be done.

Commands such as `\tenrm` or `\twlsf` have to be replaced:

```
\tenrm  →  \fontsize{10}{12pt}\rmfamily
\twlsf  →  \fontsize{12}{14.5pt}\sffamily
```

Another possibility is to use the `rawfonts` package, described in *LaTeX2ε for Authors*.

Also commands such as `\xipt` do not exist any longer. They also have to be replaced:

```
\vpt   →  \fontsize{5}{6.5pt}\selectfont
\xipt  →  \fontsize{11}{13.6pt}\selectfont
```

LaTeX 2.09 used commands with names 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 *LaTeX2ε for class and package writers*). If your package redefined one of the `\p`-commands, you should replace the redefinition by one using `\DeclareRobustCommand`.

When you use internal commands from NFSS version 1 you will have to be very careful to check if everything still works as it was once intended.

Note that macros such as `\rm` are now defined in class files, so their behaviour may differ for each class. Instead you should use the lower level commands such as `\rmfamily` in packages. When you want to make sure that you get a certain font, independent of the environment in which your macro is activated, you can first call `\normalfont` and then switch the various parameters of the font selection scheme as necessary.

In some cases you may need to use the user level commands such as `\textrm`. This is necessary for instance when you define a command that may also be used in mathmode.

Document styles → Classes

Minimal updates are necessary. When you are upgrading a document style to a document class there are a few things that you really *have* to change, or your class will not work.

One of the things that must be done, is making sure that your class doesn't define `\@normalsize` but `\normalsize`. Make sure that `\renewcommand` is used to redefine `\normalsize` as it is already defined in the kernel of LaTeX, but to produce a warning that it needs to be given a real definition.

Another aspect that needs to be dealt with, is that the parameters `\@maxsep`, `\@dblmaxsep` and `\footheight` no longer exist. The first two were part of the float placement algorithm, but a change in that algorithm made them superfluous. The parameter `\footheight` was reserved in LaTeX 2.09, but it was never used.

The declarative font changing commands (`\rm`, `\sf` etc.) are no longer defined by default. Their definitions have been moved to the class files. Make sure that you define them or that they are not used by the users of your class. The standard document classes all contain definitions such as the following:

```
\DeclareOldFontCommand{\rm}
          {\normalfont\rmfamily}{\mathrm}
```

This tells LaTeX that when `\rm` is used in the text it should switch to `\normalfont` and then select the roman family. When `\rm` is used in mathmode LaTeX will select the font that would be selected by `\mathrm`[2].

[2] See *LaTeX2ε font selection* for more details.

Build on standard classes. When upgrading your own document style you should consider to reimplement it by building on an existing Document Class. With the new features of LaTeX2ε this has become very easy. The advantage of this approach is that you don't have to maintain a whole lot of code that is probably basically a copy of the code in one of the standard document classes. (See below for a few examples of how to build your own document class on an existing class.) Some documentstyles written for LaTeX 2.09, such as `ltugboat`, contain a command such as `\input{article.sty}`. This was the only solution in LaTeX 2.09—to build a new documentstyle upon an existing style. But, there was no way of ensuring that the file `article.sty` which was found by LaTeX wasn't out of date. As you see in the examples below, it is now possible to ensure that you use a version of `article.cls` that was released after a certain date.

Suggested updates. Apart from the essential changes to your document class, there are also a few changes that you are encouraged to make. Most of these changes have to do with the new possibilities the package and class writers interface gives you.

In a LaTeX 2.09 document style an option was declared by defining a command that starts with `\ds@` followed by the name of the option. Later on in the documentstyle the command `\@options` was called to execute the code for the options that were supplied by the user. For example, the document style `article` contained the following lines of code:

```
...
\def\ds@twoside{\@twosidetrue
           \@mparswitchtrue}
\def\ds@draft{\overfullrule 5\p@}
...
\@options
...
```

This code fragment defined two options, twoside and draft.

The same effect can be achieved by using LaTeX2ε syntax, as is shown by the following code fragment from the document class `article`:

```
...
\DeclareOption{oneside}
      {\@twosidefalse \@mparswitchfalse}
\DeclareOption{twoside}
       {\@twosidetrue  \@mparswitchtrue}
\DeclareOption{draft}
          {\setlength\overfullrule{5pt}}
\DeclareOption{final}
          {\setlength\overfullrule{0pt}}
...
\ProcessOptions
```

As you can see, the intention of this code is easier to understand.

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

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

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

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

Some packages and document styles had to redefine the \begin{document} or \end{document} commands to achieve their goal. This is no longer necessary. The "hooks" \AtBeginDocument and \AtEndDocument are now available. They make it more likely that your package will work together with someone else's.

When a document class needs to pass information to the user, you can use one of the commands \ClassInfo, \ClassWarning, \ClassWarningNoLine or \ClassError. A similar set of commands exists for packages.

Be colour safe. One of the new features of LaTeX2ε is the support for coloured documents. To create a document that contains colour you need:

- the color package, which is part of the LaTeX2ε distribution;
- a driver which supports colour—dvips by Tomas Rokicki is an example of such a driver;
- colour safe macros.

The first two points are probably obvious, the third point needs some explanation. TeX has no knowledge of colour, therefore the macros need to keep track of the colour. To achieve that, various changes have been made to the kernel of LaTeX. This has been done in such a way that the changes are 'dormant' when the color package isn't used. As an example, here is the current definition[3] of the LaTeX command \sbox:

```
\def\sbox#1#2{\setbox#1\hbox{%
  \color@@setgroup#2\color@@endgroup}}
```

The extra level of grouping is activated by the color package and is needed to keep colour changes local. For more information about being 'color safe' you should read the documentation that comes with the color package.

If you use the LaTeX commands for boxing sunch as \mbox, \sbox, \fbox, etc. instead of the low level commands \hbox, \vbox and \setbox, your code will be automatically 'colour safe'.

[3] Shown here only as an illustration; the actual implementation may change.

Upgrading existing 'styles'—an example tour

A minimal class. 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 class or package, using commands such as \newcommand, \setlength and \sbox (or \savebox).

However, there are some new commands for helping class and package writers. These are described in detail in *LaTeX2ε for class and package writers.*

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}
              [1994/06/01 Minimal class]
\renewcommand{\normalsize}{%
  \fontsize{10}{12}\selectfont}
\setlength{\textwidth}{6.5in}
\setlength{\textheight}{8in}
```

However, most classes will provide more than this!

Extending a class with new commands. The first example shows how you can extend an existing class with a few extra commands. Suppose you call your new class extart. It could start off with the following code:

```
%----------- Identification -----------%
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\ProvidesClass{extart}
    [1994/08/01 v2.0j
     Article like class with new commands]
```

This first line tells LaTeX that your code was written for LaTeX2ε, released after june first, 1994. The second line informs LaTeX that this file provides the document class extart, dated august 1, 1994, and with version 2.0j.

```
%----------- Option handling ----------%
\DeclareOption*{%
  \PassOptionsToClass{\CurrentOption}
                     {article}}
```

The code above instructs LaTeX to pass on every option the user asked for to the document class article.

```
\ProcessOptions
%----------- Load other class ----------%
\LoadClass[a4paper]{article}[1994/06/01]
```

The command \ProcessOptions executes the code associated with each option the user specified. The \LoadClass command subsequently loads the class file. The first optional argument to \LoadClass passes the option a4paper to the class; the second optional argument to \LoadClass asks for article.cls dated june first, 1994, or later.

Note that if you change your mind and load report instead you also have to change the second argument of \PassOptionsToClass.

```
%------------ Extra command -------------%
\newcommand\foo{\typeout{Hello world!}}
...
```

The rest of the file contains the extra code you need such as the definition of the command \foo.

Changing the layout produced by another class. The first few lines of a class that modifies the layout of an existing class would look much the same as in the example above.

```
%------------ Identification ------------%
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\ProvidesClass{review}
  [1994/08/01 v1.0
   Article like class with changed layout]
%------------ Option handling -----------%
\DeclareOption*{%
  \PassOptionsToClass{\CurrentOption}
                     {article}}
\ProcessOptions
%------------ Load other class ----------%
\LoadClass{article}[1994/06/01]
```

Suppose we have to print on paper 7 inch wide and 9.875 inch tall. The text should measure 5.5 inch by 8.25 inch

```
%------------- Layout of text -----------%
\setlength{\paperwidth}{7in}
\setlength{\paperheight}{9.875in}
\setlength{\textwidth}{5.5in}
\setlength{\textheight}{8.25in}
```

What we have to do now is position the body of the text in a proper place on the paper.

```
\setlength{\topmargin}{-.5625in}
\setlength{\oddsidemargin}{-.25in}
\setlength{\evensidemargin}{-.25in}
\setlength{\marginparwidth}{.25in}
\setlength{\headsep}{.1875in}
```

We could go on and modify other aspects of the design of the text, but that is beyond the scope of this article.

Extending a class with new options. As before, we start the document class with some identification.

```
%------------ Identification ------------%
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\ProvidesClass{optart}
  [1994/08/01 v1.0
   Article like class with extra options]
```

Suppose you want to be able to print a document in 9pt type, or when you want to be loud, print it in 14pt type. You know that the standard LaTeX classes contain the command
\input{size1\@ptsize.clo}
just after the execution of \ProcessOptions. Supposing you don't want to print an article in 19pt type, you can use the file name size9.clo to implement your design for a layout that assumes the type size is 9pt. To implement a design for 14pt type you create the file size14.clo.

Adding the options to your extended document class is done by the following two lines of code:

```
\DeclareOption{9pt}%
              {\renewcommand\@ptsize{9}}
\DeclareOption{14pt}%
              {\renewcommand\@ptsize{4}}
```

All other options have to be passed on to the article class.

```
%------------ Option handling -----------%
\DeclareOption*{%
  \PassOptionsToClass{\CurrentOption}
                     {article}}
\ProcessOptions
%------------ Load other class ----------%
\LoadClass{article}[1994/06/01]
```

A real life example. Apart from adding options to an existing document class it is also possible to *disable* options that are allowed by the document class you are building upon. An example of this is the document class ltxdoc, used by the LaTeX3 project team for the documented source code of LaTeX. It contains the following lines of code:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{ltxdoc}
     [1994/05/27 v2.0n
      Standard LaTeX documentation class]
\DeclareOption{a5paper}%
    {\@latexerr{Option not supported}%
    {}}
\DeclareOption*{%
    \PassOptionsToClass  {\CurrentOption}%
                         {article}}
...
```

The interesting bit is the line that associates the option a5paper with an error message. When someone specifies the a5paper option to the class ltxdoc he will be warned that this document class does not support printing on A5 paper.

This document class allows customization by checking if a file ltxdoc.cfg exists. If a file with that name is found the user is told that the file is read in.

```
\InputIfFileExists{ltxdoc.cfg}
{\typeout{%
 *************************************^^J%
 * Local config file ltxdoc.cfg used^^J%
 *************************************}}
{}
```

Such a configuration file might contain the instruction to use A4 paper for printing:
\PassOptionsToClass{a4paper}{article}

When the configuration file is read, the options are processed and the `article` class is loaded.

```
\ProcessOptions
\LoadClass{article}
```

Then the package doc is required. This package is needed to print documented TeX source code, which the document class `ltxdoc` is made for.

```
\RequirePackage{doc}
```

The last line from this document class that is interesting is the following:

```
\AtBeginDocument{\MakeShortVerb{\|}}
```

This instructs LaTeX to store the command `\MakeShortVerb` together with its argument (`\|`) to be executed when `\begin{document}` is encountered.

Informing the user

Error handling. LaTeX2ε contains a set of commands that provide an interface for error handling. There are commands to signal an error (and prompt for corrective user input); commands to issue a warning about something and commands to just provide some information. In figure 1 you can see an example of the use of the command `\PackageWarningNoLine`. The result of executing the command is also shown.

Compatibility with LaTeX 2.09. Upwards compatibility is provided by the compatibility mode of LaTeX2ε. This mode was introduced to be able to run old LaTeX 2.09 documents through LaTeX2ε, yielding (almost) the same result. If this is what you need to achieve, than you may be pleased to know that the `\if@compatibility` switch can be used to test for compatibility mode. Using this switch, you can develop a full blown LaTeX2ε Package or Document Class out of a LaTeX 2.09 style file and yet still be able to print your old documents without changing them.

Possible Pitfalls while upgrading. Some mistakes that might be easily made and that can lead to unexpected results:

- You declare options in your package using `\DeclareOption` but forget to call `\ProcessOptions`. LaTeX will give an error, 'unprocessed options' unless sometimes other errors in the class file intervened and prevent the system detecting this mistake.
- The usage of either `\footheight`, `\@maxsep` or `\@dblmaxsep` outside of compatibility mode will lead to a complaint from TeX about an unknown command sequence.
- With LaTeX 2.09 the order in which options to a documentstyle were specified was *very* significant. A document would fail if the options were given in the wrong order. By default LaTeX2ε does *not* process the options in the order that they were specified in the document. It rather processes them in the order that they are declared in the class or package file. When the order of processing the options is relevant to your code you can use the command `\ProcessOptions*`. This will make LaTeX2ε evaluate the options in the order that they were specified in by the user.

  For the babel package for instance, the order of processing the options is significant. The last language specified in the option list will be the one the document starts off with.

`article`	successor of the `article` document style
`report`	successor of the `report` document style
`book`	successor of the book document style
`letter`	successor of the `letter` document style
`slides`	successor of the `slides` document style *and* SLiTeX
`proc`	Successor of the proc style option
`ltxdoc`	to typeset the documented sources of LaTeX2ε
`ltxguide`	to typeset the LaTeX2ε guides
`ltnews`	to typeset the news letter that comes with each release of LaTeX

Table 2: Document classes that are part of LaTeX2ε

Document Classes and Packages in the LaTeX2ε distribution

Standard Document Classes. In table 2 an overview is given of the document classes that are available when you get the standard distribution of LaTeX2ε.

Most of these will be familiar to you, they are the successors of their LaTeX 2.09 counterparts. Basically these document classes behave like the old document styles. But there are a few changes:

- The options openbib and twocolumn are now internal options, the files `openbib.sty` and `twocolumn.sty` do not exist any more.
- A number of new options are implemented; supporting a range of paper sizes. Currently implemented are a4paper, a5paper, b5paper, letterpaper, legalpaper and executivepaper. These options are mutually exclusive.

  Another new option is the landscape option. It switches the dimensions set by one of the ..paper options. Note that this does not necessarily mean that when you combine a4paper and landscape the whole width of the paper will be used for the text. The algorithm which computes the

```
\PackageWarningNoLine{babel}
  {The language 'Dutch' doesn't have hyphenation patterns\MessageBreak
   I will use the patterns loaded for \string\language=0 instead.}
```

produces:

```
Package babel Warning: The language 'Dutch' doesn't have hyphenation patterns
(babel)                I will use the patterns loaded for \language=0 instead.
```

Figure 1: An example of the use of the command `\PackageWarning`

`\textwidth` from the `\paperwidth` has an upper bound in order to make lines of text not too long.

- The document class `letter` now also supports the option twoside. It does not support the option landscape.
- The document class `slide` can now be used with LaTeX, SLiTeX does not exist as a separate format any longer.

  Two column (using the option twocolumn) slides are not supported.

  While processing the document class `slides` LaTeX tries to load the optional file `sfonts.cfg`. This file can be used to customize the fonts used for making slides.
- The former *option* `proc.sty` has now been turned into a separate document class, which is implemented by building on `article` using the `\LoadClass` command. This class does not allow the options a5paper, b5paper and onecolumn.

A few new document classes have been added to the distribution of LaTeX. These are mainly meant to be used for documents produced by the LaTeX3 project team, but they can be used as an example of how to build a new class on top of an existing class. These classes are not yet finished and will probably change in the future.

- The document class `ltxdoc` is used in the documentation of all the LaTeX2ε source code. The document class is built upon the `article` class and also loads the doc package.

  It defines the command `\DocInclude` which works like the `\include` command from LaTeX, but sets things up for formatting documented source code.

  The formatting of the source code can be customized by creating the file `ltxdoc.cfg`. Such a file could for instance select your favorite paper size. This can be done by entering the following command in `ltxdoc.cfg`:

  ```
  \PassOptionsToClass{a4paper}{article}
  ```

  Selecting a5paper is not allowed; the source listings wouldn't fit.

`ifthen`	successor of the ifthen option
`makeidx`	successor of the makeidx option
`showidx`	successor of the showidx option
`doc`	successor of the doc option
`shortvrb`	implements `\MakeShortVerb` and `\DeleteShortVerb`
`newlfont`	successor of the newlfont option
`oldlfont`	successor of the oldlfont option
`latexsym`	makes the LaTeX symbol fonts available
`exscale`	implements scaling of the math extension font 'cmex'
`fontenc`	supports switching of *output* encoding
`syntonly`	successor of the syntonly option
`tracefnt`	successor of the tracefnt option

Table 3: Packages that are part of LaTeX2ε

- The document class `ltxguide` is used for the user guides that are included in the distribution.
- The document class `ltnews` is used for the short newsletter that accompanies the LaTeX distribution.

Packages. The packages that are contained in the LaTeX2ε distribution are listed in table 3. Most of these packages are described in *The LaTeX Companion*.

The package `ifthen` (which used to be the option ifthen) has been enhanced and now also defines `\newboolean`, `\setboolean` and `\boolean{...}` to provide a LaTeX interface to TeX's switches. Other new commands are `\lengthtest` and `\ifodd`.

The package `shortvrb` has only recently been introduced. It contains the definitions of the commands `\MakeShortVerb` and `\DeleteShortVerb` from the doc package. By providing this package those commands can also be used in other documents besides LaTeX source code documentation.

Related software bundles. Table 4 lists some related software bundles that are distributed separately.

The packages in these bundles come with documentation and each of them is also described in

`amslatex`	Advanced mathematical typesetting from the American Mathematical Society
`babel`	Supports typesetting in over twenty different languages
`color`	Provides support for colour
`graphics`	Inclusion of graphics files
`mfnfss`	Typesetting with bit-map (Metafont) fonts
`psnfss`	Typesetting with Type 1 (PostScript) fonts
`tools`	Miscellaneous packages written by the LaTeX3 project team

Table 4: Software bundles *not* part of LaTeX2$_\varepsilon$

at least one of the books *The LaTeX Companion* (Goossens et al. 1994) and *LaTeX: A document preparation system* (Lamport 1994).

References

Goossens, Michel, Frank Mittelbach and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley Publishing Company, 1994.

Lamport, Leslie. *LaTeX: A Document Preparation System.* Addison-Wesley Publishing Company, second edition, 1994.

The LaTeX3 Project team. *LaTeX2$_\varepsilon$ for Authors.* A document provided in the LaTeX2$_\varepsilon$ distribution in file `usrguide.tex`

The LaTeX3 Project team. *LaTeX2$_\varepsilon$ for class and package writers.* A document provided in the LaTeX2$_\varepsilon$ distribution in file `clsguide.tex`

The LaTeX3 Project team. *LaTeX2$_\varepsilon$ font selection.* A document provided in the LaTeX2$_\varepsilon$ distribution in file `fntguide.tex`