The cjwoutl Package*

Colin J. Wynne†

1996/04/26

Contents

Introduction

This package originated as a simple set of macros for doing outlines under PLAIN TEX. The original macros simply provided *n*`\parindent` as a left indentation for a single paragraph. This was almost three years ago. The macros improved somewhat with my knowledge of (and experience with) TEX. Last summer I finally had the time to add what I had always wanted—automatic level numbering.

This year I decided to find out just what LATEX was all about. After deciding that it wasn't that bad, I rewrote (and improved yet again) the outlining macros. Conveniently enough, a large chunk of the original PLAIN TEX code dedicated to resetting the proper counters disappeared with the change to LATEX.

At a basic level, the outline environment functions like enumeration. However, one needn't nest the calls of the outline environment. Rather, the macros used to identify the entries of the outline itself take care of this.

I think this code is now both fairly generic and rather useful, and I would be more than happy to know that it is useful to others. Feel free to e-mail me with comments, suggestions, etc. The PLAIN TEX version is still available by request for anyone who might want it.

*This file has version 0.6 as of 1996/04/26.

†E-Mail at: `cwynne@brutus.mts.jhu.edu`, `cwynne@jhu.edu` .

1 Making an Outline

`outline` An outlined portion of a document is implemented, naturally enough, inside an environment, the outline environment. There is one optional argument to the `\begin` command. If the argument `new` is given, then the outline counters will be reset. Otherwise, the values of the counters will be exactly as they were at the end of the last outline environment.

`\outl` Inside of an outline environment, there are two main user macros, one for auto-labelled entries and one for user-supplied labels. For labels to be automatically numbered and printed, the user simply enters `\outl`⟨*level*⟩. The ⟨*level*⟩ is simply an integer which corresponds to the depth within the outline. For example, level one of an outline begins at the left margin (more precisely, the *text* begins at the left margin with the label offset by 1 en to the left of that) and in a standard outline, and, indeed, according to this package's defaults, a capitalised roman numeral is used as a label. Therefore

```
  \begin{outline}
\outl{1}This is outline level one.
  \end{outline}
```

will produce

I. This is outline level one.

The package supplies seven defined levels of outline as well as a default 'level'. The default is used for any levels over seven. These levels will still receive `\outlindent` the correct indentation (specifically, level n is prefaced by `\leftskip =` $(n-1)$`\outlindent` worth of space), but will all receive the default label, normally a bullet. Thus, the user is provided with the following:

I. ...

A. ...

1. ...

a. ...

i. ...

(a) ...

(i) ...

- ...
- ...
- ...

Note that levels eight through ten have the default bullet for a label, though the indentation levels are preserved. There are a few special symbols which can be given as the ⟨*level*⟩ argument. If the user enters a `+`, `-` or `=` then the indentation level of that entry will be respectively one greater, one less or the same as that of the previous entry. Thus,

```
  \begin{outline}
\outl{1} Level one,
\outl{+} Level two,
\outl{=} Level two,
\outl{-} Level one.
  \end{outline}
```

gives

I. Level one,

A. Level two,

B. Level two,

II. Level one.

`\ol` The macro `\ol` is almost identical to the `\outl` macro, except that it takes a second argument: `\ol`⟨*level*⟩⟨*label*⟩. The ⟨*level*⟩ argument functions identically, `\npp` and the ⟨*label*⟩ argument is set as entered 1 en to the left of the text. Two macros `\unpp` are derived from `\ol`. Both take a ⟨*level*⟩ argument. The `\npp` macro is essentially `\ol`⟨*level*⟩`{\null}\indent`. The `\unpp` macro does not indent the first line of text. (The macros stand for "new paragraph" and "unindented `\npp`".)

`\outlstyle` There is also a macro to allow for some modifications in how an outline entry `firstpar` is typeset. The default style, `plain`, does nothing. An alternate style, called `\firstparfont` `firstpar`, sets the first paragraph of an outline entry in `\firstparfont`. By default, this is italics, but the font command can, of course, be renewed by the user. To change styles, one simply calls `\outlstyle`⟨*style*⟩; for example,

```
\outlstyle{firstpar}
```

calls the `firstpar` outline style.

N.B.: There is one 'secret' left to tell. Unless one wants to reset the counters by calling `\begin{outline}[new]`, it is really unnecessary to call the environment. Instead, one can simply use the `\ol` or `\outl` commands as desired. The only caveat is that the final outline grouping must be closed. Therefore, the `\endoutline` macro *must* be called. Granted, this doesn't make a whole big difference. I happen to use this 'feature' simply because I have used these macros since long before it was written in a LaTeX-environment format. YMMV.

1.1 Outline Labels

`outlN` The counters for the different levels are named `outl`n, where n is a lowercase roman numeral. Therefore, this package defines `outli`, `outlii`,...,`outlvii`. This is, as one easily sees, similar to the numbering scheme for the enumeration environment.

`\theoutlN` The numbering style for level n is given by the macro `\theoutlN`. The actual label `\labeloutlN` which is printed for an entry at level n is given (naturally enough) by `\labeloutlN`. The default label is `\labeloutldef`. The example above showed the label at the sixth level as '(a)'. The relevant definitions are:

```
\renewcommand{\theoutlvi}{\alph{outlvi}}
\renewcommand{\labeloutlvi}{(\theoutlvi)}
```

`\labelfont` The labels themselves are typeset in `\labelfont`, which can also be set by the user. The default is bold-extended.

1.2 Cross References

Cross references function as expected for outlines, using the `\label` and `\ref` commands. The `\outl` function defines the current reference string, so the `\label` should be put after the `\outl` to which it refers. For example, we might have the code

```
  \begin{outline}[new]
\outl{1} Main entry.
\outl{+} Next level.
\outl{+} The price of tea in China.\label{ol:tea}
  \end{outline}
```

producing

I. Main entry.

A. Next level.

1. The price of tea in China.

whereupon the reference `ol:tea` would refer to tea in China I.A.1.

2 The Code

2.1 Allocations

Since the levels of an outline are defined by a `\leftskip` command, it is necessary to determine when a group needs to be closed. Therefore, we define a test to see if we are in a group.

```
\newif\ifoutl@group
```

This essentially keeps the first outline entry from closing a non-existent group. Two lengths, `\aboveoutlskip` and `\belowoutlskip`, initially set to the equivalent values for math displays, are used to separate an outline environment from the surrounding text.

```
\newlength{\aboveoutlskip}
  \setlength{\aboveoutlskip}{\abovedisplayskip}
\newlength{\belowoutlskip}
  \setlength{\belowoutlskip}{\belowdisplayskip}
```

The `\outlindent` macro controls the increment between adjacent outline levels. Its default value is `\parindent`.

```
\newlength{\outlindent}
  \setlength{\outlindent}{\parindent}
```

Now we define the counters to be used. The first two identify the requested indentation level and the previous level.

```
\newcounter {outl@indsize}
\newcounter {outl@lastind}
```

`outlN` Next come the counters for the levels themselves.

```
\newcounter {outli}            % I.
\newcounter {outlii} [outli]   %   A.
\newcounter {outliii}[outlii]  %     1.
\newcounter {outliv} [outliii] %       a.
\newcounter {outlv}  [outliv]  %         i.
\newcounter {outlvi} [outlv]   %           (a)
\newcounter {outlvii}[outlvi]  %             (i)
```

There is also a counter definition for the default label. This will be explained below.

```
\newcounter {outldef}[outlvii]
```

`\theoutlN` Now we can set the proper level numbering.

```
\renewcommand{\theoutli}    {\Roman{outli}}
\renewcommand{\theoutlii}   {\Alph{outlii}}
\renewcommand{\theoutliii}  {\arabic{outliii}}
\renewcommand{\theoutliv}   {\alph{outliv}}
\renewcommand{\theoutlv}    {\roman{outlv}}
\renewcommand{\theoutlvi}   {\alph{outlvi}}
\renewcommand{\theoutlvii}  {\roman{outlvii}}
```

Again we have the default label, which I *promise* will be explained.

```
\renewcommand{\theoutldef}  {}
```

`\labeloutlN` Now the label formats will be defined.

```
\newcommand{\labeloutli}    {\theoutli.}
\newcommand{\labeloutlii}   {\theoutlii.}
\newcommand{\labeloutliii}  {\theoutliii.}
\newcommand{\labeloutliv}   {\theoutliv.}
\newcommand{\labeloutlv}    {\theoutlv.}
\newcommand{\labeloutlvi}   {(\theoutlvi)}
\newcommand{\labeloutlvii}  {(\theoutlvii)}
\newcommand{\labeloutldef}  {\ensuremath{\bullet}}
```

This next group of definitions are taken from the standard classes' definitions for the enumeration environment. I have not plumbed the cross-referencing code deeply enough to see how these macros are applied, but I take it on faith from `classes.dtx` that it works.

```
\renewcommand{\p@outli}     {}
\renewcommand{\p@outlii}    {\theoutli.}
\renewcommand{\p@outliii}   {\p@outlii\theoutlii.}
\renewcommand{\p@outliv}    {\p@outliii\theoutliii.}
\renewcommand{\p@outlv}     {\p@outliv\theoutliv.}
\renewcommand{\p@outlvi}    {\p@outlv\labeloutlv.}
\renewcommand{\p@outlvii}   {\p@outlvi\labeloutlvi.}
```

And now we have the promised explanation of having a counter for the default labels. Simply put, the `\p@...` commands are defined *via* `\setcounter`. To be able to supply cross-references for outline entries above the seven numbered levels it seems necessary to have the relevant counter definition already made. I could be wrong on this, and maybe I could have defined the cross-reference label directly. If so, let me know. Anyway, that was the explanation—probably not worth the wait, was it?

```
\renewcommand{\p@outldef}   {\p@outlvii\ldots}
```

2.2 The Environment

outline

The outline environment itself has a very simple definition. It has a single optional argument. The default value of the argument is fairly arbitrary, so long as it isn't `new`.

```
\newenvironment{outline}[1][keep]%
```

To start the environment, we must simply test to see if the counters should be reset or not.

```
 {\def\@tempa{#1} \def\@tempb{new}
  \ifx\@tempa\@tempb
    \outl@reset
  \fi
  \penalty\predisplaypenalty\vspace{\aboveoutlskip}}%
```

To finish an outlined section of a document, we include a `\par` (to make sure the `\leftskip` functions correctly), close the last outline group and let it be known that we are no longer in an outline.

```
 {\par\endgroup\global\outl@groupfalse%
  \penalty\postdisplaypenalty\vspace{\belowoutlskip}}
```

The macro to reset the outline counters is about as straightforward as possible.

```
\newcommand{\outl@reset}{%
  \setcounter{outli}  {0}
  \setcounter{outlii} {0}
  \setcounter{outliii}{0}
  \setcounter{outliv} {0}
  \setcounter{outlv}  {0}
  \setcounter{outlvi} {0}
  \setcounter{outlvii}{0}}
```

2.3 Outline Styles

\outlstyle

Outline styles are implemented as a macro call to the desired style just before the text of the outline entry is typeset. *i.e.*, in abstract terms, the `\ol` or `\outl` macros will have ⟨*label*⟩⟨*style-macro*⟩*Text text text...* The styles are macros named `\ols@`⟨*style*⟩. Thus, the `\outlstyle` macro shecks to see if the style is defined. If not, it emits a warning and selects the plain style.

```
\newcommand{\outlstyle}[1]{%
  \@ifundefined{ols@#1}
    {\PackageWarning{cjwoutl}{Outline style `#1' is undefined.  Using
                              style `plain' instead}%
     \def\outl@style{\ols@plain}}
```

If the requested style exists, we point the macro `\outl@style` towards it. (Note: if anyone can think of a better way to write this, let me know. I wanted to do it as a `\let`, but couldn't figure out the correct sequence of `\expandafter`s to get it to evaluate correctly.)

```
    {\def\outl@style{\csname ols@#1\endcsname}}}
```

Now we define the two basic styles, `plain` and `firstpar`.

```
\def\ols@plain{\@empty}
```

The `firstpar` style simply takes the first paragraph of the entry as an argument and sets it in `\firstparfont`.

```
\def\ols@firstpar#1\par{%
  \bgroup\firstparfont #1\par\egroup}

\DeclareOldFontCommand{\firstparfont}{\normalfont\itshape}{\relax}
```

Lastly, we select `plain` as the default style.

```
\outlstyle{plain}
```

2.4 The Entries

`\ol` The `\ol` macro first takes care of the outline groupings and then processes the ⟨*level*⟩ argument given by the user.

```
\newcommand{\ol}[2]{%
  \outl@checkgroups \outl@processlvl{#1}%
```

We do some minor error handling—like disallowing negative numbers. If a numerical argument is given, the `\outl@processlvl` macro decrements it by one—thus, the actual `outl@indsize` is not the level number that the user enters, but rather the number of `\outlindent`s for the requested level, and the first entry occurs at zero indents, *i.e.*, on the left margin. Thus zero, and not one, is the minimum value which can be assigned to `\outl@indsize`.

```
  \ifnum \c@outl@indsize < 1\relax%
    \setcounter{outl@indsize}{0}%
    \leftskip = 0pt%
  \else%
    \leftskip = \c@outl@indsize\outlindent%
  \fi%
```

We specify that the first paragraph of an entry should not be indented and we make the label.

```
  \noindent\theoutlabel{#2}%
```

The `\ignorespaces` macro allows the user to enter `\outl{1} Text` instead of the slightly less readable `\outl{1}Text` in the source file without the extra spaces showing up in the document. Anyway, then we call `\outl@style` which has been defined to be the selected style macro.

```
  \ignorespaces\outl@style}
```

The `\npp` and `\unpp` macros are straightforward derivatives of the `\ol` macro.

```
\newcommand{\npp}[1]{\ol{#1}{\null}\null\indent}
\newcommand{\unpp}[1]{\ol{#1}{\null}}
```

`\outl` The `\outl` macro is more or less identical to `\ol` above.

```
\newcommand{\outl}[1]{%
  \outl@checkgroups \outl@processlvl{#1}%
  \ifnum \c@outl@indsize < 1\relax%
    \setcounter{outl@indsize}{0}%
    \leftskip = 0pt%
  \else%
    \leftskip = \c@outl@indsize\outlindent%
  \fi%
```

The difference is in calling the `\outl@label` macro which takes care of the automatic labels.

```
\noindent\outl@label{\c@outl@indsize}%
\ignorespaces\outl@style}
```

`\outl@checkgroups` Now we define the macro that balances the outline groupings. If we are already in a group, provide a `\par` to get the `\leftskip` right, then close the group and change the group flag.

```
\newcommand{\outl@checkgroups}{%
  \ifoutl@group
    \par\endgroup\global\outl@groupfalse
    \smallbreak
  \fi
```

Since we have just finished an outline group, we can set `outl@lastind`, then provide appropriate space (encouraging breaks between enries) and start a new group.

```
  \setcounter{outl@lastind}{\c@outl@indsize}
  \smallbreak
  \begingroup
    \global\outl@grouptrue}
```

`\outl@processlvl` The ⟨*level*⟩ argument can be a number or one of `+`, `-` and `=`. The check is implemented as a set of nested `\if` statements.

```
\newcommand{\outl@processlvl}[1]{%
  \def\@tempa{+}\def\@tempb{-}\def\@tempc{=}
  \def\@tempd{#1}
  \ifx\@tempa\@tempd
    \addtocounter{outl@indsize}{1}
  \else\ifx\@tempb\@tempd
         \addtocounter{outl@indsize}{-1}
       \else\ifx\@tempc\@tempd
              % Nothing---stay at same level.
            \else
              \setcounter{outl@indsize}{\@tempd}
```

If we are setting to a new level number, we decrement by one, as mentioned in the definition of `\ol`, in order to convert the level number to an indentation size.

```
              \addtocounter{outl@indsize}{-1}
            \fi
       \fi
  \fi}
```

2.5 Labels

`\outl@label` The `\outl@label` macro, called by `\outl` with the `outl@indsize` as an argument, simply decides which level to make by way of an `\ifcase` statement.

```
\newcommand{\outl@label}[1]{%
  \ifcase #1
    \@dolabeli\or\@dolabelii\or\@dolabeliii%
    \or\@dolabeliv\or\@dolabelv\or\@dolabelvi%
    \or\@dolabelvii\else\@dolabeldef%
  \fi}
```

The labels are set in `\labelfont`. I have a set of personal macros that define a `\labelfont` for other uses, so the declaration is surrounded by a test.

```
\@ifundefined{labelfont}
  {\DeclareOldFontCommand{\labelfont}%
    {\normalfont\bfseries\mathversion{bold}}{\mathbf}}
  {} % Do nothing---|\labelfont| is already defined
```

The generic form of the outline labels is given by `\theoutlabel`.

```
\newcommand{\theoutlabel}[1]{%
  \llap{\hbox{\labelfont#1\enskip}}}
```

Finally we define the macros which are called for the auto-numbering and label creation for the `\outl` macro.

```
\newcommand{\@dolabeli}{%
  \refstepcounter{outli}\theoutlabel{\labeloutli}}
\newcommand{\@dolabelii}{%
  \refstepcounter{outlii}\theoutlabel{\labeloutlii}}
\newcommand{\@dolabeliii}{%
  \refstepcounter{outliii}\theoutlabel{\labeloutliii}}
\newcommand{\@dolabeliv}{%
  \refstepcounter{outliv}\theoutlabel{\labeloutliv}}
\newcommand{\@dolabelv}{%
  \refstepcounter{outlv}\theoutlabel{\labeloutlv}}
\newcommand{\@dolabelvi}{%
  \refstepcounter{outlvi}\theoutlabel{\labeloutlvi}}
\newcommand{\@dolabelvii}{%
  \refstepcounter{outlvii}\theoutlabel{\labeloutlvii}}
\newcommand{\@dolabeldef}{%
  \refstepcounter{outldef}\theoutlabel{\labeloutldef}}
```

And with that, we are finished.