The bits LaTeX package
basic bits (Frankenstein's guts)

Matt Swift <swift@alum.mit.edu>

Version: 1.2 Date: 2001/08/31
Documentation revision: 2001/08/31

Abstract

Bits provides a programmer's interface for a new idea called a *bit*, which is like an environment but has a title, author, and other attributes usually only associated with the `document` environment.

This package is useful but may have problems and is unsupported.

Contents

Part I
Discussion

This documentation is spotty, but should be accurate. Please feel free to write me if you can't figure something out.

1 Basic concepts and terminology

1.1 Users' interface

A *bit* is the smallest part of a document treated as an independent unit. Each *bit* has an explicit type called a *bitstyle*. (Examples: generic, story, poem, recipe, letter.) A bit's bitstyle determines the markup that is legal within the bit. (Additional declarations can be made on a per-bit basis, also; see below.) A bit's formatting is determined by both its bitstyle and the copystyle (see below) of the document. New bitstyles can be added.

A *copystyle* is one of an explicit list of ways to present a collection of bits together in a single document. It might well be called a "documentstyle" and be implemented as a LaTeX documentclass, but if they are kept separate, copystyles and documentclasses can coexist with each other, and this leads to a greater flexibility when dealing with existing documents. The relationship between copystyles and documentclasses is discussed further below. New copystyles can be added.

1.2 Programmers' interface

The formatting of a bit is considered to involve a fixed number of *tasks*. Examples of tasks performed near the beginning of formatting a bit are inserting some space after the preceding bit, and formatting the title of the current bit. Each task must be assigned a *procedure* that accomplishes that task. (Notice that the null procedure might be a suitable procedure for some tasks in some situations.) We call a mapping of some tasks to suitable procedures a *profile*. We call a profile that maps the *complete* set of tasks a *scheme*. You need a scheme to format a bit, and the scheme to use for a given bit is determined by its bitstyle and copystyle.

Now we are in a position to try to visualize the variable space established by the *bits* package. It's not simple, and if you think of a better way, please let me know.

As just explained, to format a bit, you need a scheme, which is a series of procedures suitable for a series of certain tasks. To be concrete, let's assume there are 4 tasks. Identify each task with a general color. In our example, let's use red, blue, green, and yellow. Think of a scheme as a stack of colored blocks, each representing a procedure. The top block, corresponding to the first task, is red, the second block is blue, the third green, and the bottom block, corresponding to the final task, is yellow.

Suppose I create a new scheme, the same as the original except for a slight change in the first task. This would be represented by changing the color of the red, or top, block to a different shade of red. The precise color of a block represents the specific procedure. A general color is associated with a task because it is expected

that no procedure suitable for one task is going to be suitable for another task. The redness of a block represents its suitability for the the first task.

(If we wanted to be exact, we would observe that some procedures, such as the null procedure, might be suitable for more than one task. OK, let such procedures be a shade of grey, with the null task, let us say, being white. Then the colored block at each task's position will have either the associated color of the task or a shade of grey.)

I have drawn in your mind the picture of two stacks of 4 colored blocks, differing only in the shade of the top (red) block. Let's give them the names Jack and Jill. Let Jack and Jill represent two bitstyles. You can imagine as many bitstyles as you like—Jack, Jill, Hank, Wendy, and Beauregard; all are stacks of four colored blocks arranged next to each other in a line, a red block on top, a blue block, a green block, and a yellow block. Some of the stacks might even have identical blocks. This would mean that two bitstyles have the same scheme. You refer to bitstyles by their name, so in the future, you could change one bitstyle and make it different from the other. In fact, when you create a new bitstyle, it is initially simply a copy of an existing bitstyle. Then you can modify the new one if you want to.

bitclass? bittype? bitstyle best describes a scheme. bitscheme?

You've almost got the whole picture now. We need to consider copystyles. Suppose Jack, Jill, Hank, Wendy, and little Beauregard are all siblings. They have the same last name, Grimm. Grimm is the name of a copystyle. Across town, or in a parallel universe, there is another set of siblings Jack, Jill, etc. with a different last name, Beedle. They, too, are represented by a line of stacks of colored blocks. Line up the Beedle siblings behind the Grimm siblings, and you now have a pile of colored blocks 4 high (4 tasks), 5 wide (the 5 bitstyles Jack, Jill, etc.), and 2 deep (the 2 copystyles Grimm and Beedle).

That's it. Given a bitstyle (e.g., Hank) and a copystyle (e.g., Beedle), you have a particular scheme for formatting a bit.

As you may notice, there is so far no justification for the metaphor of family. Nothing I've said yet about tasks and procedures corresponds to the metaphor's suggestion that Hank Beedle is probably more similar to Wendy Beedle than he is to Hank Grimm; or that when putting together the Beedles in a family photograph, there is a certain pleasing aesthetic unity that would be lacking if any of those odd-looking Grimms were to stray into the picture. In fact, technically, each scheme can be completely different from every other scheme, Hank Beedle being as different from Wendy Beedle as he is from Hank Grimm.

These metaphors are justified when you consider the way in which new copystyles and bistyles are added, and the expected purposes for bitstyles and copystyles.

OR: family of Poems, family of recipes. First name is the copystyle. Then a document would consist of Hank Poem, Hank Recipe, etc. = Standard Poem, Standard Recipe; Fancy Poem, Fancy Recipe.

Or, each family has one Poem, one Recipe, etc. So a document puts together members of a family, Poem, Recipe, etc.

2 Usage

Bits look like LaTeX environments that take two arguments. (Surprise surprise that's exactly what they are.) Here's an example of a bit with hypothetical bitstyle "poem":

```
\begin{poem}{Butterflies}{\subtitle{An Address to My Stomach}}
  O jittery one, quavering tub, \\
  Rumble not your complaints so violently! \\
  Remember your jollitude just last Sunday, \\
  The pound of bovine ambrosia \\
  I filled you to the gills with \\
  To silence you, on this, my wedding day.
\end{poem}
```

3 Extension and customization

3.1 Customizing an instance of a bit

Fat little Hank Beedle might come with a note tied around his neck on a piece of yarn, with special instructions on his care and feeding from his mother. "Peanuts will give him hives," it might say. And "A quart of sour cream is always welcome."

3.2 Adding bitstyles, copystyles, and tasks

When you add a new bitstyle B', you create a new scheme for each of the existing copystyles. You name an existing bitstyle B and zero or more profiles. For each existing copystyle C, the profiles will be composed with the $\frac{B}{C}$ scheme to obtain the new scheme $\frac{B'}{C}$. (Notice that if any of the profiles you name are schemes, the choice of B is irrelevant, and that if you name no profiles, $B' = B$.)

The procedure is identical when you add a new copystyle. Just swap references to copystyles and bibstyles in the last paragraph.

You can add to list of tasks. When you do this, you must assign a default procedure for it (part of the Z scheme, the universal default). All existing bitstyles and copystyles will inherit the same procedure for this task. This makes sense because they've presumably all done the task the same way before, if they've done it at all. You can promulgate changes with `\Promulgate` etc.

4 Notes

FIX; bit type determines its markup; bitstyle is a mapping of tasks to procedures.

Part II
Implementation

5 Version control

\fileinfo
\DoXUsepackagE
\HaveECitationS
\fileversion
\filedate
\docdate
\PPOptArg

These definitions must be the first ones in the file.

```
\def\fileinfo{basic bits (Frankenstein's guts)}
\def\DoXPackageS {}
\def\fileversion{v1.2}
\def\filedate{2001/08/31}
\def\docdate{2001/08/31}
\edef\PPOptArg {%
  \filedate\space \fileversion\space \fileinfo
}
```

If we're loading this file from a `\ProcessDTXFile` command (see the *compsci* package), then `\JusTLoaDInformatioN` will be defined; othewise we assume it is not (that's why the FunkY NamE).

If we're loading from `\ProcessDTXFile`, we want to load the packages listed in `\DoXPackageS` (needed to typeset the documentation for this file) and then bail out. Otherwise, we're using this file in a normal way as a package, so do nothing. `\DoXPackageS`, if there are any, are declared in the `dtx` file, and, if you're reading the typeset documentation of this package, would appear just above. (It's OK to call `\usepackage` with an empty argument or `\relax`, by the way.)

```
\makeatletter% A special comment to help create bst files.  Don't change!
\@ifundefined{JusTLoaDInformatioN} {%
  }{% ELSE (we know the compsci package is already loaded, too)
  \UndefineCS\JusTLoaDInformatioN
  \SaveDoXVarS
  \eExpand\csname DoXPackageS\endcsname\In {%use \csname in case it's undefined
    \usepackage{#1}%
  }%
  \RestoreDoXVarS
  \makeatother
  \endinput
}% A special comment to help create bst files.  Don't change!
```

Now we check for LaTeX2e and declare the LaTeX package.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{bits}[\PPOptArg]
```

6 Preliminaries

6.1 Requirements

```
\RequirePackage{blkcntrl,letterhead,moredefs,relsize}
```

6.2 Option processing

```
\DeclareOption{poetica} {%
  \RequirePackage{poetica}
```

```
  \newcommand\FullTitlePoetica {%
    \begin{Poetica}
      \fontshape{t}\selectfont
      \BitTitle\\%
      \ifx\BitSubtitle\ShortEmpty \else
        \BitSubtitle\\%
      \fi
    \end{Poetica}
  }
  \AtBeginDocument {%
    \let\FullTitle\FullTitlePoetica
  }
}
```

FIX: hmm, I think I always want user options so that I can have two different packages require a package, and they don't have to match up on their options. What about options that can only be used in preamble, that would be the right thing for many cases I think. Aak, `\ends` can't be defined without special handling? Hmm, seems like it can.

```
\DeclareBooleanUserOptions{ends}{noends}
\DeclareBooleanUserOptions{signed}{anonymous}

\DeclareBooleanOptions{titlepage}{notitlepage}

\ExecuteOptions{ends,notitlepage,signed}
\ProcessOptions
```

7 Making new copy and bit styles

7.1 General stuff

\bt@a
\bt@b
\bt@c
\bt@d
\bt@e

```
\ReserveCS\bt@a
\ReserveCS\bt@b
\ReserveCS\bt@c
\ReserveCS\bt@d
\ReserveCS\bt@e
```

\bt@tasks
\bt@bitstyle@list
\bt@copystyle@list

Comma-separated lists of valid bitstyles, copystyles, and tasks. We start them off non-null to bootstrap the system of adding new ones, which adds them preceded by a comma.

```
\newcommand\bt@tasks {%
  environment,end@bit,begin@bit,interbits,firstbit%
  ,settitle,setfirsttitle%
}
\newcommand\bt@bitstyle@list {%
  generic%
}
\newcommand\bt@copystyle@list {%
  standard%
}
```

\if@bt@trymoreschemes@
\@bt@trymoreschemes@true
\@bt@trymoreschemes@false
\bt@list@car
\bt@list@cdr

`\bt@list@car` and `\bt@list@cdr` expand to the car and cdr of a comma-separated list, respectively.

```
\provideboolean{@bt@trymoreschemes@}

\NewName{bt@list@car}{#1,#2\@nil} {%
  #1%
}
\NewName{bt@list@cdr}{#1,#2\@nil} {%
  #2%
}
```

\NewCopystyle
\NewBitstyle
\bt@assign@task

To define a new style, you specify a base style and a list of profiles. Each of these can be empty. If no base style is specified, the root style is used (i.e., standard copystyle or generic bitstyle). See discussion above.

```
\newcommand\NewCopystyle [2][Z] {% args: [profile-list] new-copystyle
  \addto@macro\bt@copystyle@list{,#2}%
  \@for\bt@a:=\bt@bitstyle@list \do {%
    \@for\bt@b:=\bt@tasks \do {%
      \eExpand\bt@b\In {%
          \expandafter\bt@assign@task
            \expandafter{\bt@a}{#2}{##1}{#1}%
      }%
    }%
  }%
}
\newcommand\NewBitstyle [2][Z] {% args: [scheme-list] new-bitstyle
  \addto@macro\bt@bitstyle@list{,#2}%
  \@for\bt@a:=\bt@copystyle@list \do {%
    \@for\bt@b:=\bt@tasks \do {%
      \eExpand\bt@a\In {%
        \eExpand\bt@b\In {%
%        \eExpandNest\bt@b\In {%  FIX?
          \bt@assign@task{#2}{##1}{####1}{#1}%
        }%
      }%
    }%
  }%
}
\newcommand\bt@assign@task [4] {% args: bitstyle copystyle
                                 %       task scheme-list
  \@bt@trymoreschemes@true
  \edef\bt@d{#4}% scheme-list that diminishes; edef not necessary
  \@whilesw\if@bt@trymoreschemes@\fi {%
```

There's at least one element to start; and the last scheme (Z) is always defined.

```
    \edef\bt@c{%
      \bt@list@car\bt@d,\@nil
    }
    \@ifundefined{bt@make@#3@\bt@c} {%
        \edef\bt@d{\bt@list@cdr\bt@d,\@nil}%
        }{% ELSE
      \@nameuse{bt@make@#3@\bt@c}{#1}{#2}%
      \@bt@trymoreschemes@false
      }%
  }%
}
```

romulgateTaskAcrossCopystyles
PromulgateTaskAcrossBitstyles
```
\newcommand\PromulgateTaskAcrossCopystyles [3] {% args: task scheme bitstyle
  \@for\bt@a:=\bt@copystyle@list \do {%
    \@nameuse{bt@make@#1@#2}{#3}{\bt@a}%
  }%
}
\newcommand\PromulgateTaskAcrossBitstyles [3] {% args: task scheme copystyle
  \@for\bt@a:=\bt@bitstyle@list \do {%
    \@nameuse{bt@make@#1@#2}{\bt@a}{#3}%
  }%
}
```

\bt@begin@bit@common

\bt@begin@bit@common should be called early in the sequence of beginning a bit.

\BitStyle
\PreBitAll

```
\providesavebox\sc@box@a
\newcommand\bt@begin@bit@common [2] {%
  \title{#1}%
```

We want to clear all optional attributes here, such as subtitle. FIX: should be a list of attributes and perhaps even abstract clearing functions on them, etc.

```
  \subtitle{}%
```

Execute stuff: FIXME: nice hooks can be set here to execute things later We introduce the

so that extra spaces in the argument are not going to mean anything. It would be complicated but possible by means perhaps of \nullfont to avoid starting a paragraph here. Arg, the kernel is so opaque on how it handles ignoring spaces and what it's doing with everypar.

```
  \par #2%
  \refstepcounter{bit}%
  \ifnum\value{bit} = \@ne
    \@nameuse{bt@firstbit@\CopyStyle}%
  \else
    \@nameuse{bt@interbits@\CopyStyle}%
  \fi
}
\ReserveCS\BitStyle
\ReserveCS\PreBitAll
```

7.2 Schemes

7.2.1 The Z scheme

\FullTitle

The Z scheme is always the last resort deafult, and it corresponds to the standard copystyle and the generic bitstyle.

FIXME: whenever we have a dummy we have to use DefName or else we error FIXME: Aak, when Promulgating we need defnames for everyting!

```
\newcommand\bt@make@environment@Z [2] {% args: bitstyle dummy
  \DefName{end#1}{}{}%
  \DefName{#1}{##1##2} {%
    \DefName{end#1}{} {%
      \@nameuse{bt@end@bit@#1@\CopyStyle}{##1}%
    }%
    \@nameuse{bt@begin@bit@#1@\CopyStyle}{##1}{##2}%
  }%
```

```
}
\newcommand\bt@make@interbits@Z [2] {% args: dummy copystyle
  \DefName{bt@interbits@#2}{} {%
    \relax
    \par\pagebreak[2]\bigskip\bigskip
    \@nameuse{bt@settitle@\BitStyle @#2}%
  }%
}
\newcommand\bt@make@firstbit@Z [2] {% args: dummy copystyle
  \DefName{bt@firstbit@#2}{} {%
    \relax
    \par\pagebreak[2]%
    \thispagestyle{empty}%
    \@nameuse{bt@setfirsttitle@\BitStyle @#2}%
  }%
}
\newcommand\bt@make@begin@bit@Z [2] {% args: bitstyle copystyle
  \DefName{PreBit#1}{}{}%
  \DefName{bt@begin@bit@#1@#2}{##1##2} {%
    \def\BitStyle{#1}%
    \bt@begin@bit@common{##1}{##2}%
    \PreBitAll
    \@nameuse{PreBit#1}%
  }%
}
\newcommand\bt@make@end@bit@Z [2] {% args: bitstyle copystyle
  \DefName{bt@end@bit@#1@#2}{##1} {%
    \TheEnd
  }%
}
\newcommand\bt@make@settitle@Z [2] {% args: bitstyle copystyle
  \DefName{bt@settitle@#1@#2}{} {%
    \relax
    \begin{center}\FullTitle\end{center}
  }%
}
\newcommand\bt@make@setfirsttitle@Z [2] {% args: bitstyle copystyle
  \DefName{bt@setfirsttitle@#1@#2}{} {%
    \relax
    \begin{center}\FullTitle\end{center}
    \DTypeout{FIXME setfirsttitle in Z scheme}%
  }%
}
```

When there is no subtitle, it sometimes seems better for the title to be set one size larger than the copy; but when you have a subtitle, you need the title two sizes larger so that the subtitle can be an intermediate size. With CM fonts, because their boldface is extended, a bold subtitle at the same size as the copy looks good, and larger is too larger. So this is a tricky issue. A basic kludge that would work for me, given the font families I have an use, would be to check for CM fonts here, or on the other hand specifically for an extended boldface, or for Dante/Bulmer, or set a flag in my dante.sty or bulmer.sty.

```
\newcommand\FullTitle {%
  \relsize{2}\bfseries
```

```
%  \relsize{1}\bfseries
  \BitTitle\\%
  \ifx\BitSubtitle\ShortEmpty
  \else
    \csname sbseries\endcsname  % this way it's harmless if undefined
    \relsize{-1}\BitSubtitle\\%
  \fi
}
```

7.2.2 The P scheme

\bt@make@setfirsttitle@P
\bt@make@settitle@P
\bt@make@end@bit@P
\bt@make@begin@bit@P

The P scheme was created for the poem bitstyle.

```
\newcommand\bt@make@begin@bit@P [2] {% args: bitstyle copystyle
  \DefName{bt@begin@bit@#1@#2}{##1##2} {%
    \def\BitStyle{#1}%
    \bt@begin@bit@common{##1}{##2}%
    \PreBitAll
    \verse        % FIX: why does the PreBitAll hook come before \verse and the
  % BitHook come after?  Why doesn't the PreBitAll come in
  % @common@ -- well, so that a bit can put stuff before the hooks.
    \@nameuse{PreBit#1}%
  }%
}
\newcommand\bt@make@end@bit@P [2]  {% args: bitstyle copystyle
  \DefName{bt@end@bit@#1@#2}{##1} {%
    \relax
    \endverse
    \TheEnd
  }%
}
\newcommand\bt@make@settitle@P [2]  {% args: bitstyle copystyle
  \DefName{bt@settitle@#1@#2}{} {%
    \relax
    \begin{verse}\FullTitle\end{verse}
  }%
}
\newcommand\bt@make@setfirsttitle@P [2]  {% args: bitstyle copystyle
  \DefName{bt@setfirsttitle@#1@#2}{} {%
    \relax
    \begin{verse}\FullTitle\end{verse}
    \DTypeout{this is setfirsttitle in P scheme}%
  }%
}
```

7.2.3 The R scheme

For the prose bitstyle.

```
\newcommand\bt@make@settitle@R [2] {%
  \DefName{bt@settitle@#1@#2}{} {%
    \relax
    \ifx\BitTitle\ShortEmpty\else
      \noindent
      \begingroup
```

```
        \relsize{1}%
        \bfseries
        \BitTitle
      \endgroup
      \par\smallskip
    \fi
  }%
}
```

7.2.4 The T scheme

FIX: for what?

8 The standard copystyle and generic bitstyle

```
\@for\bt@e:=\bt@tasks \do {%
  \PromulgateTaskAcrossBitstyles{\bt@e}{Z}{standard}
}
```

9 Bit Styles

9.1 Story

This is the same as generic.

```
\NewBitstyle{generic}
\NewBitstyle{story}
```

9.2 Poem

```
\NewBitstyle{poem}
\PromulgateTaskAcrossCopystyles{begin@bit}{P}{poem}
\PromulgateTaskAcrossCopystyles{end@bit}{P}{poem}
\PromulgateTaskAcrossCopystyles{settitle}{P}{poem}
\PromulgateTaskAcrossCopystyles{setfirsttitle}{P}{poem}
```

9.3 Prose

```
\NewBitstyle{prose}
\PromulgateTaskAcrossCopystyles{settitle}{R}{prose}
```

10 Copy Styles

```
\NewCopystyle{titlepage}

\newcommand\bt@make@setfirsttitle@T [2] {% args: bitstyle copystyle
  \DefName{bt@setfirsttitle@#1@#2}{} {%
    \thispagestyle{empty}%
    \vspace*{\fill}%
    \begin{center}
      {\relsize{5}\BitTitle \\}%
      \ifx\BitSubtitle\ShortEmpty\else
        {\relsize{3}\BitSubtitle \\}%
      \fi
    \end{center}
```

```
    \vfill
%      \CoverNotes
%      \vfill
    \par
    \begingroup
      \sffamily
      \hfill
      \begin{tabular}{r@{\hspace{\@ne em}}r}
        Copy of:&\todayabbrev   \\%
        Version of:&\Lastchange \\%
        Date:&\Date             \\%
      \end{tabular}
      \par\bigskip\bigskip
      \begin{raggedleft}
        \def\and{\\}%
        \larger\scshape\Author
```

A final \par is needed to close the paragraph in raggedleft mode.

```
        \par
      \end{raggedleft}
    \endgroup
    \newpage
    \setcounter{page}{\@ne}%
  }%
}
\PromulgateTaskAcrossBitstyles{setfirsttitle}{T}{titlepage}
```

11 Bit counters and interbit skips

\c@bit
\c@bitbit
\c@bitbitbit

```
\providecounter{bit}
\providecounter{bitbit}[bit]
\providecounter{bitbitbit}[bitbit]
```

\bt@bitskip
\InterBitBreak
\InterBitBitBreak
\InterBitBitBitBreak

\medbreak is penalty -100; \smallbreak is 50. FIXME: what's going on here?

```
\newcommand\bt@bitskip[1] {%
  \par
  \ifdim \lastskip < \bigskipamount
    \removelastskip
    \penalty -500%
    #1%
  \fi
}
\newcommand\InterBitBreak {%
  \bt@bitskip
  \bigskip
}
\newcommand\InterBitBitBreak {%
  \bt@bitskip
  \medskip
}
\newcommand\InterBitBitBitBreak {%
  \bt@bitskip
  \smallskip
}
```

\bitbit
\bitbitbit
\bt@bitbit
\bt@bitbitbit

FIXME: These should be made attributes of the Generic Bit Style.

```
\newcommand\bitbit {%
  \@ifstar {%
      \bt@bitbit{}%
    }{% ELSE
    \refstepcounter{bitbit}%
    \bt@bitbit{\thebitbit\ }%
  }%
}
\newcommand{\bt@bitbit} [2] {%
  \ifnum\value{bitbit}=\@ne \else
     \InterBitBitBreak
  \fi
  \noindent #1{\relsize{-1}\bfseries #2}%
  \par\nobreak
  \smallskip
}
\newcommand\bitbitbit {%
  \@ifstar {%
      \bt@bitbitbit{}%
    }{% ELSE
    \refstepcounter{bitbitbit}%
    \bt@bitbitbit{\thebitbitbit\ }%
  }%
}
\newcommand\bt@bitbitbit [2] {%
  \ifnum\value{bitbitbit}=\@ne \else
    \InterBitBitBitBreak
  \fi
  \noindent #1{\relsize{-1}\bfseries #2}%
  \par\nobreak
  \smallskip
}
```

12 Userinfos

\Author
\author
\Lastchange
\lastchange
\Date
\date
\title
\BitTitle
\BitSubtitle
\Subtitle
\subtitle

When we redefine `\author`, `\date`, and `\title`, we add a new action to the original meaning. This means that standard commands like `\maketitle` will still perform as expected, if someone prefers to use that instead of the `Frankenstein` commands.

```
\SaveCS\date

\newcommand*\Date {%
  \todayabbrev
}
\defcommand\date [1] {%
  \MDSaveddate{#1}%
  \def\Date{#1}%
}
\SaveCS\title
\ReserveCS\BitTitle

\defcommand\title [1] {%
```

```
  \MDSavedtitle{#1}%
  \def\BitTitle{#1}%
}
```

Why did I originally want `\Subtitle` to be a user command instead of `\subtitle`?

```
\NewUserInfo*[\subtitle]\BitSubtitle
%\NewUserInfo*[\Subtitle]\BitSubtitle
%\newlet\subtitle\Subtitle
\NewUserInfo*\Lastchange
\lastchange{\todayabbrev}
```

`\Copystyle`
`\copystyle`

```
\NewUserInfo*\CopyStyle
\copystyle{standard}
```

13 Dates

`\today`
`\todayabbrev`

```
\def\today {%
  \number\day\space\ifcase\month\or
        January\or February\or
        March\or April\or May\or June\or July\or August\or September\or
        October\or November\or December\fi\space\number\year
}
\newcommand\todayabbrev {%
  \number\day\space\ifcase\month\or
        Jan\or Feb\or
        Mar\or Apr\or May\or Jun\or Jul\or Aug\or Sep\or
        Oct\or Nov\or Dec\fi\space\number\year
}
```

`\copywrite` This adds a copyright message to the beginning of `\Notes`. Its single argument should be a year.

```
\newcommand\copywrite [1] {%
  \edef\Notes {%
    \copyright#1 All rights reserved. \Notes
  }%
}
```

14 Ends

`\EndSign` It's very hard to attach this thing without allowing a page break before it. FIX.

```
\newcommand\EndSign {%
  \par\nobreak
  \vspace{2\bigskipamount}%  multiplication turns glue into dimen.
  \hfill
```

We do this so that `\Author` can have `\\`s in it.

```
  \begingroup
    \scshape
    \let\and\@tabularcr
    \begin{tabular}{c}
```

```
      \Author \\%
      \relsize{-1}\itshape\Date
    \end{tabular}%
  \endgroup
}
```

`\theend` User command which (almost) always makes an `\EndSign`:

```
\newcommand\theend {%
  \if@signed@
    \EndSign
  \fi
}
```

`\TheEnd` The end of a bit calls `\TheEnd`. The default definition is either nothing or `\EndSign`, depending on some condition.

```
\newcommand\TheEnd {%
  \def\sc@t@c {%
    prizes%
  }%
  \let\sc@t@a\EndSign
  \ifx\CopyStyle\sc@t@c
    \InitCS*\sc@t@a
  \else
    \if@signed@
      \if@ends@
% FIX: when you get the footer going...
%       \ifnum\value{page} < 2%
%         \InitCS*\sc@t@a
%       \fi
      \else
        \InitCS*\sc@t@a
      \fi
    \else
      \InitCS*\sc@t@a
    \fi
  \fi
  \sc@t@a
}
```

15 Version control

We use macros instead of boxes, because at this point some of the macros in the boxes may change their meaning.

Algorithm: We will maintain two `\hbox`es `to \textwidth` each time `\MakeCover` is called. Set the footboxes to `\v@idbox` here. Whenever `\MakeCover` is called, it should stack another `\vbox` with the right information in it into both boxes. The information is expanded, and thus the macro contents can change between calls to `\MakeCover`, but the box being built inserts two different `\vboxes` with their value at the moment.

`\bt@vcbox`
`\VersionControlString`
`\VersionControlAuthorString`

```
\newcommand\BTVersionWord {Version}
```

```
\newcommand\BTCopyWord {Copy}
\newcommand\bt@vcbox {%
  \footnotesize
  \ifx\Lastchange\ShortEmpty\else
    {\bfseries \BTVersionWord:} {\slshape \Lastchange}%
  \fi
  \hspace{\@ne em}%
  {\bfseries \BTCopyWord:} {\slshape \todayabbrev}%
}
\newcommand\VersionControlString {%
  \hfill \bt@vcbox \hfill
}
```

I should assign some names to the various meanings of and, like protect.

```
\newcommand\VersionControlAuthorString {%
  \bt@vcbox \hfill {\scshape\def\and{\unskip, \ignorespaces}\Author}%
}
```

\whitespace

```
\requirecommand\whitespace {%
  \bigbreak
}
```

Part III
Configuration

We leave the rest to a configuration file.

```
\InputIfFileExists{bits.cfg}{}{}
```

The contents of the distributed configuration file are below.

```
\def\fileinfo{Bits package configuration}
\def\fileversion{v1}
\def\filedate{1996/01/24}
\def\docdate{1996/01/24}

\ProvidesFile{bits.cfg}
\InputIfFileExists{monster.cfg}{}{}
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.