The `dozenal` Package, v5.2

Donald P. Goodman III

January 27, 2015

Abstract

The `dozenal` package provides some simple mechanisms for working with the dozenal (duodecimal or "base 12") numerical system. It redefines all basic LaTeX counters, provides a command for converting arbitrary decimal numbers into dozenal, and provides new, real METAFONT characters for ten and eleven, though the commands for producing them can be redefined to produce any figure. As of v2.0, it also includes Type 1 versions of the fonts, selected (as of v5.0) with the `typeone` package option. This package uses the `\basexii` algorithm by David Kastrup.

Contents

1 Introduction

While most would probably call it at best overoptimistic and at worst foolish, some people (the author included) do still find themselves attracted to the dozenal (base-twelve) system. These people, however, have been pretty hard up[1] in the LaTeX world. There is no package file available which produces dozenal counters, like page and chapter numbers, nor were there *any* (I made a pretty diligent

[1] This is an Americanism for "out of luck" or "in difficult circumstances," for those who do not know.

search) dozenal characters for ten and eleven, leaving dozenalists forced to use such makeshift ugliness as the "X/E" or "T/E" or "*/#" or whatever other standard they decided to use. While this sort of thing may be acceptable in ASCII, it's absolutely unacceptable in a beautiful, typeset document.

Enter the `dozenal` package. This package automates all the messiness of being a dozenalist and using LaTeX. It redefines all the counters (though you'll have to redefine them yourself if you're using your own), provides an algorithm (generously donated by the intrepid David Kastrup) for converting arbitrary positive whole numbers into dozenal (this is eTeX, but all modern distributions will compile that), and finally, it includes original dozenal characters, specifically designed to blend in well with Knuth's Computer Modern fonts, though they should do fine with the more common body fonts, as well.

This document was typeset in accordance with the LaTeX DOCSTRIP utility, which allows automatic extraction of source code and documentation from the same source.

2 Basic Functionality

The `dozenal` package performs several basic tasks, which are the core of its functionality. A brief listing of them will help the user understand the options available, which are explained later on in this document.

- Provides commands for converting decimal numbers to dozenal and back again. (The "back again," conversion of dozenal back to decimal, only works in limited circumstances.)
- Provides default characters for the two transdecimal digits, "↊" for ten and "↋" for eleven; these correspond to the accepted Unicode standard digits "turned digit two" and "turned digit three," which have been approved for inclusion in Unicode, and are expected to be part of the official standard sometime in 1200 (2016.). These characters copy-paste as "X" and "E," the (somewhat) standard ASCII representations of these two digits. However, other characters can easily be substituted if desired.
- Redefines the counters in standard LaTeX document classes (such as `article`, `book`, and so forth) to use dozenal rather than decimal. This behavior can be shut off if desired.

That covered, we can now move on to how these features are exploited by the user.

3 Base Conversion

The `dozenal` package provides several new commands for base conversion. The first, and by far the most important given the purpose and content of this package, is `\basexii`. This is a very simple command which takes the following structure:

`\basexii`

`\basexii{`⟨*number*⟩`}{`⟨*ten symbol*⟩`}{`⟨*eleven symbol*⟩`}`

What the above means is that the command is `\basexii` and it takes three mandatory arguments: first, the number to be converted into dozenal; second, the symbol that should be used for ten; and third, the symbol that should be used for eleven. This number should be positive and whole; that is, it should be zero or higher, and it should not contain a fractional part. TEX is a typesetting program, after all; if you want a robust decimal to dozenal converter, there are many options that any dozenalists caring enough to use this package will already know about.

This `\basexii` algorithm was produced by David Kastrup, well known and admired in the TEX world for his many useful packages and other contributions. He posted this algorithm on comp.text.tex; it is included here with his kind and generous permission.

That one would want to use the same ten and eleven symbols throughout a document seems a reasonable assumption; therefore, I have provided a simplified
`\dozens` version of the `\basexii` command, `\dozens`. `\dozens` takes only a single argument, the number to be converted; the ten and eleven symbols used are those produced by the commands `\x` and `\e`, to which we'll get in a moment.

Finally, as of v5.0, we can convert numbers back to decimal from dozenal, if we
`\basex` wish. We do this with the `\basex` macro, which takes a single argument, which is the dozenal number you wish to convert to decimal. This is subject to some pretty harsh restrictions, however. First, the only tokens allowed in the number are 0–9, `\x`, and `\e`. Second, these will only work with *unexpanded* `\x` and `\e`. If you don't know what this means, then good for you; TEX programmers will envy you.

To illustrate these limitations, let's define a new counter and dozenize it. Here, we define the counter and give it a nice value which will ensure that its dozenal value will have an `\e` in it:

```
\newcounter{testcount}\setcounter{testcount}{47}
```

In dozenal, of course, "47" is "3Ɛ." Now, let's redefine that counter so that its results will be dozenal:

```
\renewcommand\thetestcount{\basexii{\value{testcount}}{\x}{\e}}
```

Now we get to do lovely things like the following:

`\thetestcount` = 3Ɛ

Now we can try to get that number in decimal with `\basex`. But don't try it; `\basex{\thetestcount}` doesn't work because the `\x` and `\e` are already expanded. Instead, use LATEX's built-in functions for chores like this:

`\arabic{testcount}` = 47

On the other hand, if you have an actual string you want converted, you can send it directly to `\basex` without worrying about expansion, because through much trial and error and banging head against wall, the expansion issues have already been resolved:

`\basex{3\e}` = 47

So `\basex` is of limited utility, but it's a nice tool to add to the box.

4 Dozenal Characters and Fonts

4.1 Shorthands for Dozenal Characters

To make use of the `\dozens` shorthand discussed earlier,[2] you need to have the commands `\x` and `\e` defined. Fortunately, this package does that for you.

`\x` `\e` `\x` and `\e` are the commands used to quickly and easily access the symbols for ten and eleven. They default to using the special dozenal characters that are part of this package; they could be easily redefined if for some reason you don't like the Pitman characters (which are soon to be included in Unicode) in the following manner:

```
\renewcommand\x{X}
```

Or whichever characters you like to use. If you prefer the Dozenal Society of America's proposed characters (a stylized X and E), then this package will disappoint you. May I suggest `$\chi$` (χ) and `$\xi$` (ξ) as a stopgap while you locate or produce real characters of your own? Sorry; I'm an American myself, but I much prefer the Pitman characters for a variety of reasons (feel free to email me if you care), and creating fonts in METAFONT, even small and inconsequential ones like this, is too much work for characters that I don't even like.

4.2 The `dozenal` Fonts

The fonts provided by the dozenal package are essentially complete fonts which contain only the Pitman dozenal characters; these are ↊ for ten and ↋ for eleven. These characters are designed to blend well with the Computer Modern fonts; they work passably well with Times-type fonts and with kpfonts, and possibly with others.

The characters also come in all the appropriate shapes and sizes; a few examples follow.

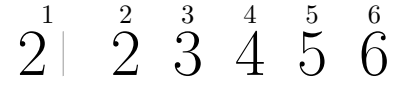
	Roman	*Italic*	**Boldface**
Footnotesize	↊ ↋	*↊ ↋*	**↊↋**
Normalsize	↊ ↋	*↊ ↋*	**↊↋**
LARGE	↊ ↋	*↊ ↋*	**↊↋**
Huge	↊ ↋	*↊ ↋*	**↊↋**

They will work in paragraph or math mode without distinction.

As of v4.0, `dozenal` also includes fonts for tally marks specifically designed for use in the dozenal base. In many European countries tallies are kept in a

[2] See supra, Section 3, at page 3.

very similar way; this font demonstrates a way that such tally marks can be made consistent as well as dozenal.



`\tally` These are accessed by the `\tally` command, which takes one argument: the number, 1–6, which you want to put in tallies. Entering "X" or "E" will yield "Ƨ" or "Ɛ" respectively. Other characters will produce nothing.

The fonts are all prefixed `dozch`, if for some reason direct access to them is needed.

5 Package Options

The `dozenal` package redefines all the standard LaTeX counters, such as `section` and `enumii`. If you've defined your own counters, you'll need to dozenize them yourself; however, this is an easy matter:

```
\renewcommand\thecounter{\basexii{\arabic{counter}}{\x}{\e}}
```

For example. Of course, you can fill in the `\x` and `\e` with whatever you want (though it would make more sense to simply redefine `\x` and `\e`, so that all the counters would use the same characters), or you could use the `\dozens` command instead. Whatever your pleasure might be.

If you *don't* want all the counters to be redefined, or if you're using a class which doesn't include basic LaTeX counters, you'll want to use the `nocounters` `nocounters` option. The `nocounters` option to the package prevents the redefinition of these counters. The effect of this is that the commands of the package (`\basexii`, `\dozens`, etc.) are made available, but all the counters will still be in decimal. This permits using dozenal characters in an otherwise decimal document; it also proves useful in dozenal document in which these counters are undefined (e.g., `minimal`).

The `dozenal` fonts were designed in METAFONT, and they are distributed in both METAFONT-generated bitmaps and autotraced Postscript Type1 fonts. The `typeone` `typeone` option forces `dozenal` to provide Postscript Type 1 fonts rather than METAFONT bitmaps to TeX. Both of these are produced from the same font files, though, so the difference is very slight. However, the Type1 fonts do generally look better on screen; the `typeone` option will probably be used most of the time that `dozenal` itself is used.

6 Implementation

Make sure that we have fixltx2e loaded, so that the `\TextorMath` magic will work.

```
1 \RequirePackage{fixltx2e}
```

Now we ensure that `ifpdf` is loaded, so that we can test for pdf or dvi modes.

```
\RequirePackage{ifpdf}
```

Now we declare the option "nocounters", which prevents `dozenal` from redefining all the counters. This prevents errors in document classes which don't have these counters, such as `minimal`. Defines the command `\nocounters` if and only if the options is named.

```
\DeclareOption{nocounters}{%
\def\nocounters{}%
}%
```

Now we define the `typeone` option, which forces the use of the Type 1 versions of the dozenal fonts.

```
\newif\iftypeone\typeonefalse
\DeclareOption{typeone}{\typeonetrue}
\ProcessOptions\relax
```

We then define the font that we're using for our METAFONT-produced Pitman characters. Incidentally, we also define the command `\doz`, though I can't foresee any decent use for it except in packages and preambles; it is then used to define `\x` and `\e`, which provide the ten and eleven symbols for all the counter redefinitions. This includes definitions for both T1 and OT1 encodings, so it will work with either.

```
\iftypeone%
\ifpdf
\pdfmapfile{=dozenal.map}
\fi
\DeclareFontFamily{T1}{dozch}{}
\DeclareFontShape{T1}{dozch}{m}{n}{<-6> dozchars6
<7> dozchars7 <8> dozchars8 <9> dozchars9 <10-11>
dozchars10 <12-16> dozchars12 <17-> dozchars17  }{}
\DeclareFontShape{T1}{dozch}{b}{n}{<-> dozchb10 }{}
\DeclareFontShape{T1}{dozch}{bx}{n}{<-6> dozchbx6
<7> dozchbx7 <8> dozchbx8 <9> dozchbx9 <10-11>
dozchbx10 <12-> dozchbx12 }{}
\DeclareFontShape{T1}{dozch}{m}{sl}{<-8> dozchsl8
<9> dozchsl9 <10-11> dozchsl10 <12-> dozchsl12 }{}
\DeclareFontShape{T1}{dozch}{bx}{sl}{<-> dozchbxsl10 }{}
\DeclareFontShape{T1}{dozch}{m}{it}{<-7> dozchit7
<8> dozchit8 <9> dozchit9 <10-11> dozchit10
<12-> dozchit12 }{}
\DeclareFontShape{T1}{dozch}{bx}{it}{<-> dozchbxi10 }{}
\def\doz#1{{\fontfamily{dozch}\fontencoding{T1}\selectfont #1}}%
\else%
\DeclareFontFamily{OT1}{dozch}{}
\DeclareFontShape{OT1}{dozch}{m}{n}{<-6> dozchars6
<7> dozchars7 <8> dozchars8 <9> dozchars9 <10-11>
dozchars10 <12-16> dozchars12 <17-> dozchars17  }{}
\DeclareFontShape{OT1}{dozch}{b}{n}{<-> dozchb10 }{}
\DeclareFontShape{OT1}{dozch}{bx}{n}{<-6> dozchbx6
```

```
<7> dozchbx7 <8> dozchbx8 <9> dozchbx9 <10-11>
dozchbx10 <12-> dozchbx12 }{}
\DeclareFontShape{OT1}{dozch}{m}{sl}{<-8> dozchsl8
<9> dozchsl9 <10-11> dozchsl10 <12-> dozchsl12 }{}
\DeclareFontShape{OT1}{dozch}{bx}{sl}{<-> dozchbxsl10 }{}
\DeclareFontShape{OT1}{dozch}{m}{it}{<-7> dozchit7
<8> dozchit8 <9> dozchit9 <10-11> dozchit10
<12-> dozchit12 }{}
\DeclareFontShape{OT1}{dozch}{bx}{it}{<-> dozchbxi10 }{}
\def\doz#1{{\fontfamily{dozch}\fontencoding{OT1}\selectfont #1}}%
\fi%
\newcommand\x{\TextOrMath{\protect\doz{{X}}}{\X}}%
\newcommand\e{\TextOrMath{\protect\doz{{E}}}{\E}}%
\DeclareSymbolFont{dozens}{OT1}{dozch}{m}{n}
\DeclareMathSymbol{\X}{\mathord}{dozens}{88}
\DeclareMathSymbol{\E}{\mathord}{dozens}{69}
```

Put in some additional code for the tally marks.

```
\newcommand\tally[1]{%
\usefont{OT1}{dozch}{m}{n}\selectfont{#1}%
}%
```

Then we define our command which will produce the dozenal numbers from decimal sources. This algorithm was taken directly from the publicly available archives of comp.text.tex, where it was posted by the well-known and redoubtable David Kastrup. We also define the `\dozens` command, a simplified `\basexii` (which, in fact, depends utterly upon `\basexii`), just to make it easy for everyone.

```
\def\basexii#1#2#3{\ifcase\numexpr(#1)\relax
0\or1\or2\or3\or4\or5\or6\or7\or8\or9\or#2\or#3\else
\expandafter\basexii\expandafter{\number\numexpr((#1)-6)/12}{#2}{#3}\expandafter\basexii\expand
\newcommand\dozens[1]{\basexii{#1}{\x}{\e}}
```

Now that we can convert numbers *to* dozenal, let's set it up so that we can convert them *from* dozenal. This is pretty ugly stuff here, because it's mostly straight TEX (and e-TEX) without higher-level conveniences, and because it's an expansion nightmare. If we didn't have to account for `\x` and `\e` because included in such strings, it's fairly easy; but it took a great deal of troubleshooting to make it work with them.

First, we work up some macros to help us count the characters in an argument (the same, more or less, as is used in the `basicarith` package, adapted from those by "Florent" at `tex-and-stuff.blogspot.com`); then, we convert to decimal with `\basex`.

```
\newcount\b@charcount
\def\gobblechar{\let\char= }
\def\assignthencheck{\afterassignment\checknil\gobblechar}
\def\countunlessnil{%
\ifx\char\nil \let\next=\relax%
\else%
\let\next=\auxcountchar%
\advance\b@charcount by1%
```

```
\fi%
\ifx\char\backslash\advance\b@charcount by-1\fi%
\next%
}%
\def\auxcountchar{%
\afterassignment\countunlessnil\gobblechar%
}%
\def\countchar#1{\def\xx{#1}\b@charcount=0\expandafter\auxcountchar\xx\nil}
%end Florent code
\def\gobble#1{}%
\newcount\@numdigs%
\newcount\@decmult%
\newcount\@loopindex\@loopindex=1%
\newcount\@decnum\@decnum=0%
\newcount\@digit\@digit=1%
\def\basex#1{%
\countchar{#1}\@numdigs=\b@charcount%
\@decmult=1\@loopindex=1%
\loop\ifnum\@loopindex<\@numdigs%
\multiply\@decmult by12%
\advance\@loopindex by1%
\repeat%
\@loopindex=0%
\def\isten{x}\def\iselv{e}\def\isquote{\backslash}%
\edef\@doznum{\detokenize{#1}\relax}%
\@decnum=0%
\loop\ifnum\@loopindex<\@numdigs%
\def\@firstchar{\expandafter\@car\@doznum\@nil}%
\if\@firstchar\@backslashchar
\edef\@doznum{\@removefirstdig{\@doznum}}%
\else
\if\@firstchar\isten\def\@firstchar{10}\fi
\if\@firstchar\iselv\def\@firstchar{11}\fi
\advance\@loopindex by1%
\@digit=\@firstchar%
\edef\@doznum{\@removefirstdig{\@doznum}}%
\multiply\@digit by\@decmult%
\advance\@decnum by\@digit%
\divide\@decmult by12%
\fi
\repeat
\the\@decnum%
}
\def\@removefirstdig#1{%
\expandafter\expandafter\expandafter\gobble\expandafter\string#1%
}
```

Now, of course, we simply redefine all the counters. This covers only those counters included in the basic LaTeX document classes, however, so if you've written your own, you'll need to redefine them yourself.

This first bit ensures that the counters are redefined even if the command `\mainmatter` is not defined. We have to do this outside of the `\g@addto@macro` below; otherwise, in documents where `\mainmatter` is defined but not used, the counters will not be redefined. This way, they're redefined in all cases.

This also takes care of ensuring that the counters are only redefined if the "nocounters" options was *not* specified.

```
\@ifundefined{nocounters}{%
\@ifundefined{c@page}{}{%
\renewcommand\thepage{\basexii{\value{page}}{\x}{\e}}}
\@ifundefined{c@footnote}{}{%
\renewcommand\thefootnote{%
\basexii{\value{footnote}}{\x}{\e}}}
\@ifundefined{c@part}{}{%
\renewcommand\thepart{%
\basexii{\value{part}}{\x}{\e}}}
\@ifundefined{c@subparagraph}{}{%
\renewcommand\thesubparagraph{%
\basexii{\value{subparagraph}}{\x}{\e}}}
\@ifundefined{c@paragraph}{}{%
\renewcommand\theparagraph{%
\basexii{\value{paragraph}}{\x}{\e}}}
\@ifundefined{c@equation}{}{%
\renewcommand\theequation{%
\basexii{\value{equation}}{\x}{\e}}}
\@ifundefined{c@figure}{}{%
\renewcommand\thefigure{%
\basexii{\value{figure}}{\x}{\e}}}
\@ifundefined{c@table}{}{%
\renewcommand\thetable{%
\basexii{\value{table}}{\x}{\e}}}
\@ifundefined{c@table}{}{%
\renewcommand\thempfootnote{%
\basexii{\value{mpfootnote}}{\x}{\e}}}
\@ifundefined{c@enumi}{}{%
\renewcommand\theenumi{%
\basexii{\value{enumi}}{\x}{\e}}}
\@ifundefined{c@enumii}{}{%
\renewcommand\theenumii{%
\basexii{\value{enumii}}{\x}{\e}}}
\@ifundefined{c@enumiii}{}{%
\renewcommand\theenumiii{%
\basexii{\value{enumiii}}{\x}{\e}}}
\@ifundefined{c@enumiv}{}{%
\renewcommand\theenumiv{%
\basexii{\value{enumiv}}{\x}{\e}}}
\@ifundefined{c@chapter}{%
\renewcommand\thesection{%
\basexii{\value{section}}{\x}{\e}}
\renewcommand\thesubsection{%
```

```
\thesection.\basexii{\value{subsection}}{\x}{\e}}
\renewcommand\thesubsubsection{%
\thesubsection.\basexii{\value{subsubsection}}{\x}{\e}}
}{
\renewcommand\thechapter{%
\basexii{\value{chapter}}{\x}{\e}}
\renewcommand\thesection{%
\thechapter.\basexii{\value{section}}{\x}{\e}}
\renewcommand\thesubsection{%
\thesection.\basexii{\value{subsection}}{\x}{\e}}
\renewcommand\thesubsubsection{%
\thesubsection.\basexii{\value{subsubsection}}{\x}{\e}}
}
```

Finally, if the `\mainmatter` command is used, we need to make sure that it doesn't mess up our numbering scheme.

```
\@ifundefined{mainmatter}{}{%
\g@addto@macro\mainmatter{%
\@ifundefined{c@page}{}{%
\renewcommand\thepage{\basexii{\value{page}}{\x}{\e}}}
\@ifundefined{c@footnote}{}{%
\renewcommand\thefootnote{\basexii{\value{footnote}}{\x}{\e}}}
\@ifundefined{c@part}{}{%
\renewcommand\thepart{\basexii{\value{part}}{\x}{\e}}}
\@ifundefined{c@subparagraph}{}{%
\renewcommand\thesubparagraph{%
\basexii{\value{subparagraph}}{\x}{\e}}}
\@ifundefined{c@paragraph}{}{%
\renewcommand\theparagraph{%
\basexii{\value{paragraph}}{\x}{\e}}}
\@ifundefined{c@equation}{}{%
\renewcommand\theequation{%
\basexii{\value{equation}}{\x}{\e}}}
\@ifundefined{c@figure}{}{%
\renewcommand\thefigure{%
\basexii{\value{figure}}{\x}{\e}}}
\@ifundefined{c@table}{}{%
\renewcommand\thetable{%
\basexii{\value{table}}{\x}{\e}}}
\@ifundefined{c@table}{}{%
\renewcommand\thempfootnote{%
\basexii{\value{mpfootnote}}{\x}{\e}}}
\@ifundefined{c@enumi}{}{%
\renewcommand\theenumi{%
\basexii{\value{enumi}}{\x}{\e}}}
\@ifundefined{c@enumii}{}{%
\renewcommand\theenumii{%
\basexii{\value{enumii}}{\x}{\e}}}
\@ifundefined{c@enumiii}{}{%
\renewcommand\theenumiii{%
```

```
\basexii{\value{enumiii}}{\x}{\e}}}
\@ifundefined{c@enumiv}{}{%
\renewcommand\theenumiv{%
\basexii{\value{enumiv}}{\x}{\e}}}
\@ifundefined{c@chapter}{
\renewcommand\thesection{%
\basexii{\value{section}}{\x}{\e}}
\renewcommand\thesubsection{%
\thesection.\basexii{\value{subsection}}{\x}{\e}}
\renewcommand\thesubsubsection{%
\thesubsection.\basexii{\value{subsubsection}}{\x}{\e}}
}{
\renewcommand\thechapter{%
\basexii{\value{chapter}}{\x}{\e}}
\renewcommand\thesection{%
\thechapter.\basexii{\value{section}}{\x}{\e}}
\renewcommand\thesubsection{%
\thesection.\basexii{\value{subsection}}{\x}{\e}}
\renewcommand\thesubsubsection{%
\thesubsection.\basexii{\value{subsubsection}}{\x}{\e}}
} % end if it's defined
}
}
}{} % end redefinition of counters block
```

And that's the end. Thanks for reading, folks; please email me with any suggestions or improvements.

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.