The `dozenal` Package, v7.1

Donald P. Goodman III

July 1, 2017

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 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 a few other 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 (as of June 11↋↋) are now part of the Unicode standard. 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.
- Provides macros for converting dozenal numbers to words; e.g., "3↋" to "three dozen eleven."
- Provides macros for converting numbers to "doman" numerals; that is, a dozenal version of Roman numerals.

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

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

2.1 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,
`\basexii` is `\basexii`. This is a very simple command which takes the following structure:

`\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,
`\basex` if we 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 a pretty harsh restriction, however: the only tokens allowed in the number are 0–9, `X`, and `E`; putting in anything else will cause violent choking with "expected a number"-type errors.

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 `dozenal` allow us to do lovely things like the following:

`\thetestcount` = 3Ɛ

It's tempting to try to put that number into `\basex` to get it in decimal; but don't try it; `\basex{\thetestcount}` doesn't work because it contains expanded versions of `\x` and `\e`. Fortunately, you don't need it; LaTeX already has the value of the `testcount` counter in its innards, and is quite used to outputting it in decimal:

`\arabic{testcount}` = 47

On the other hand, if you have an actual string you want converted, you can send it directly to `\basex`:

`\basex{3E}` = 47

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

2.2 Numbers to Words

`dozenal` allows you to easily convert counters into *words*, by supplying said counter's name to the `\doznumtoword` macro. It takes as its argument the name of a *counter* (not a number itself!) and converts this into words:

```
This page's number is ``\doznumtoword{page}.''
```
This page's number is "four."

This macro works with much larger numbers, as well. Purely for exemplary purposes, let's define a counter `somecount` with `\newcounter{somecount}`, and set it equal to (decimal) 851 with `\setcounter{somecount}{851}`. "851" in dozenal is "5ƷƐ." Therefore, `\doznumtoword{somecount}` will yield "five biqua ten unqua eleven." If you want to change the capitalization, use other macros; e.g., `\DOZnumtoword{somecount}` gives "FIVE BIQUA TEN UNQUA ELEVEN," while `\Doznumtoword{somecount}` gives "Five Biqua Ten Unqua Eleven."

`\doznumtoword` (and friends) do act correctly when there are zeroes in the middle of the number, e.g., when `somecount` is equal to decimal 6977, which is dozenal 4055, it will output "four triqua zero biqua five unqua five." When there's a zero at the *end* of the number, that zero is still output: `\doznumtoword{somecount}` where `somecount` is equal to 144 gives "one biqua zero unqua zero," not simply "one biqua." I haven't decided yet if this is a bug or a feature; when I do, I'll act accordingly.

The rank words ("unqua," "biqua," and so forth) are all customizable by user-level commands, which are named `\dozrankoneword` for "unqua," `\dozranktwoword` for "biqua," and so forth. Simply redefine them like so:

```
\def\dozrankoneword{dozen}
\setcounter{somecount}{51}
\doznumtoword{somecount} = four dozen three
```

This type of macro is useful for putting page numbers in both digits and words, for example.

2.3 Doman Numerals

Dozenalists have also come up with some ideas for how to use Roman numerals in a decimal way; therefore, the `dozenal` package provides some macros to assist with that, as well.

To form "Doman" numerals, we simply alter the values of the traditional Roman characters into more dozenal-friendly alternatives. So "v" is 6, "x" is 10, and so on. Then, to avoid stringing four of the same character together, we extend the subtractive principle to allow up to *two* lower characters prior to a higher-value character. So, e.g., "iiv" is $6 - 2$, or 4, while "iv" is $6 - 1$, or 5.

`\Doman` `\doman` The macros `\Doman` and `\doman` are equivalent to `\Roman` and `\roman`, giving either capitalized or lowercase dozenal Roman numerals.

1	i	2	ii	3	iii	4	iiv	5	iv	6	v
7	vi	8	vii	9	viii	↊	iix	↋	ix	10	x
11	xi	12	xii	13	xiii	14	xiiv	15	xiv	16	xv
17	xvi	18	xvii	19	xviii	1↊	xiix	1↋	xix	20	xx
21	xxi	22	xxii	23	xxiii	24	xxiiv	25	xxiv	26	xxv
27	xxvi	28	xxvii	29	xxviii	2↊	xxiix	2↋	xxix	30	xxx
31	xxxi	32	xxxii	33	xxxiii	34	xxxiiv	35	xxxiv	36	xxxv
37	xxxvi	38	xxxvii	39	xxxviii	3↊	xxxiix	3↋	xxxix	40	xxl
41	xxli	42	xxlii	43	xxliii	44	xxliiv	45	xxliv	46	xxlv
47	xxlvi	48	xxlvii	49	xxlviii	4↊	xxliix	4↋	xxlix	50	xl
51	xli	52	xlii	53	xliii	54	xliiv	55	xliv	56	xlv
57	xlvi	58	xlvii	59	xlviii	5↊	xliix	5↋	xlix	60	l

In the table above, the Doman numerals 1–60 are displayed, along with their Hindu-Arabic equivalents. This table was produced entirely using a new LaTeX counter, `testdoman`; `\thetestdoman` was defined by saying

```
\renewcommand\thetestdoman{\doman{\value{testdoman}}}
```

The Hindu-Arabic columns were displayed with `\dozens{\value{testdoman}}`. Every two table cells, `testdoman` is increased by one. The whole table was produced without manually entering a single number in either form.

Most commonly, Roman numerals are seen in part numbers and in the page numbering of frontmatter. To achieve that result, do:

```
\renewcommand\thepage{\doman{\arabic{page}}}
```

You can reset this to normal dozenal numerals (or whatever else you'd prefer) when you reach your mainmatter.

3 Dozenal Characters and Fonts

3.1 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.

3.2 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.

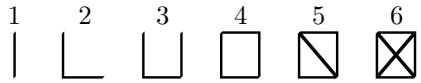
	Roman	*Italic*	**Boldface**
Footnotesize	↊ ↋	*↊ ↋*	**↊↋**
Normalsize	↊ ↋	*↊ ↋*	**↊↋**
LARGE	↊ ↋	*↊ ↋*	**↊↋**
Huge	↊ ↋	*↊ ↋*	**↊↋**

They will work in paragraph or math mode without distinction.

3.3 Tally Marks

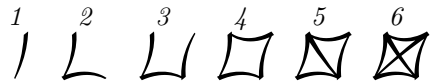
As of v4.0, `dozenal` also includes fonts for tally marks specifically designed for use in the dozenal base. In many European countries tallies are kept in a very similar way; this font demonstrates a way that such tally marks can be made consistent as well as dozenal.

[2] See supra, Section 2.1, at page 3.



`\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 "↊" or "↋" respectively. Other characters will produce nothing.

As of v6.0, there are separate, hand-written versions of the tally marks, accessed by putting tallies in italic:



In other words, to get these shapes, say `\textit{\tally{6}}`; this will give you "↋."

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

3.4 Dozenal Radix Point

Lastly, `dozenal` allows the use of the common dozenal radix point, ";", to work properly in math mode. Some dozenalists prefer to use ";" rather than "." (or ",") to mark the transition from integers to fractions in numbers; e.g., three-and-a-half is "3;6." In math mode, however, this doesn't work; a semicolon in math mode is punctuation, and a spurious space is inserted (quite correctly) afterward. We don't want this space when we're using it as a radix point. So `dozenal` (using code adapted from Walter Schmidt's `icomma` package) fixes this:

`$3;6$` = 3;6

And all is right with the world.

4 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` option.

`nocounters` 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 document classes 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` 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.

typeone

5 Implementation

Make sure that we have `fixltx2e` loaded, so that the `\TextorMath` magic will work. Also, as of v6.0, we also require `xstring` to use `\IfStrEq` in the number-to-word code.

```
\RequirePackage{fixltx2e}
\RequirePackage{xstring}
```

Now we ensure that `ifpdf` is loaded, so that we can test for pdf or dvi modes. We also make sure we have `ifluatex`, so that `dozenal` won't choke when you use it with the `typeone` option. We'll take care of the `luatex` stuff right away, while we're at it:

```
\RequirePackage{ifpdf}
\RequirePackage{ifluatex}
\ifluatex
\protected\def\pdfmapfile {\pdfextension mapfile }
\fi
```

We also require `mfirstuc`, because it's *great*:

```
\RequirePackage{mfirstuc}
```

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}}%
\DeclareSymbolFont{dozens}{T1}{dozch}{m}{n}
\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}}%
\DeclareSymbolFont{dozens}{OT1}{dozch}{m}{n}
\fi%
\newcommand\x{\TextOrMath{\protect\doz{{X}}}{\doz@X}}%
\newcommand\e{\TextOrMath{\protect\doz{{E}}}{\doz@E}}%
\DeclareMathSymbol{\doz@X}{\mathord}{dozens}{88}%
\DeclareMathSymbol{\doz@E}{\mathord}{dozens}{69}%
```

Put in some additional code for the tally marks.

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

```
\doz{#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. I use `xstring` here, replacing a messy macro mesh from the last version (5.3).

```
\newcount\doz@countchar
\def\doz@charcount#1{%
\StrLen{#1}[\doz@filler]%
\doz@countchar=\doz@filler%
}%
```

Now we develop our conversion routines for `\basex`. For v6.0, these were hugely simplified by using the `xstring` package instead of trying to bash through in plain TEX, which eventually worked but was *not* pretty. We start by defining a few counters to help us out:

```
\newcount\doz@lfiller\doz@lfiller=-1%
\newcount\doz@total\doz@total=0%
\newcount\doz@loopi\doz@loopi=0%
\newcount\doz@multiplier\doz@multiplier=1%
```

Next we adopt a macro from TeX.SE question 140476, from user "Dan", which works like a charm even when using counter values rather than simple integers. These macros let us grab an individual character from a string; in this case, from the argument of `\basex`.

```
% macro from TeX.SE question 140476, posted by user "Dan"
\def\ninthofmany#1#2#3#4#5#6#7#8#9{#9\gobbletorelax}
\def\gobbletorelax#1\relax{}
\def\doz@CharAt#1#2{%
\expandafter\ninthofmany\romannumeral\numexpr(9000-\number#1000)#2\relax}
% end "Dan" macro
```

Here's where the money happens. We loop through each digit of the argument, multiplying it by the appropriate factor of 10 (the dozen, of course), and then add that to a rolling total. At the end, we output the number.

```
\def\doz@ten{X}%
\def\doz@elv{E}%
\def\doz@basex#1{%
\doz@total=0%
\doz@loopi=\doz@countchar%
```

```
\doz@multiplier=1%
\def\doz@wholenum{#1}%
\loop\ifnum\doz@loopi>0%
\def\doz@currchar{\doz@CharAt{\number\doz@loopi}{\doz@wholenum}}%
\if\doz@currchar\doz@ten%
\doz@lfiller=10%
\else\if\doz@currchar\doz@elv%
\doz@lfiller=11%
\else%
\doz@lfiller=\doz@currchar%
\fi\fi%
\multiply\doz@lfiller by\doz@multiplier%
\multiply\doz@multiplier by12%
\advance\doz@total by\doz@lfiller%
\advance\doz@loopi by-1%
\repeat%
\the\doz@total%
}%
\def\basex#1{%
\doz@charcount{#1}%
\doz@loopi=0%
\doz@basex{#1}%
\doz@multiplier=1%
\doz@total=0%
}%
```

Finally, we define the macros for creating "Doman" (dozenal Roman) numerals. One, of course, is defined in terms of the other. First, though, we need a modulus operator:

```
\newcount\doz@modulus%
\def\doz@modulo#1#2{%
\doz@modulus=#1%
\divide\doz@modulus by#2%
\multiply\doz@modulus by#2%
\multiply\doz@modulus by-1%
\advance\doz@modulus by#1\relax%
}%
\newcount\doz@quotient%
\def\doz@quot#1#2{%
\doz@quotient=#1%
\divide\doz@quotient by#2%
}%
```

Now we can move on to the meat of the operation:

```
\newcount\doz@romct%
\newif\ifdoz@domancaps%
\def\doz@doman#1{%
\doz@romct=#1%
\doz@quot{\doz@romct}{1728}%
\loop\ifnum\doz@quotient>0%
```

```
\ifdoz@domancaps M\else m\fi%
\advance\doz@quotient by-1%
\advance\doz@romct by-1728%
\repeat
\ifnum\doz@romct>1440
\ifnum\doz@romct<1584
\ifdoz@domancaps CCM\else ccm\fi%
\advance\doz@romct by-1440
\else%\ifnum\doz@romct>1583
\ifdoz@domancaps CM\else cm\fi%
\advance\doz@romct by-1584
\fi
\fi
\ifnum\doz@romct>575%
\ifnum\doz@romct<719
\ifdoz@domancaps CCD\else ccd\fi%
\advance\doz@romct by-576
\else
\ifnum\doz@romct<864%
\ifdoz@domancaps CD\else cd\fi%
\advance\doz@romct by-720%
\fi
\fi
\else
\ifnum\doz@romct>719
\ifdoz@domancaps D\else d\fi%
\advance\doz@romct by-719%
\fi
\fi
\doz@quot{\doz@romct}{144}%
\loop\ifnum\doz@quotient>0%
\ifdoz@domancaps C\else c\fi%
\advance\doz@quotient by-1%
\advance\doz@romct by-144%
\repeat
\ifnum\doz@romct>119
\ifnum\doz@romct<132
\ifdoz@domancaps XXC\else xxc\fi%
\advance\doz@romct by-120
\else
\ifdoz@domancaps XC\else xc\fi%
\advance\doz@romct by-132
\fi
\fi
\ifnum\doz@romct>71
\ifdoz@domancaps L\else l\fi%
\advance\doz@romct by-72
\fi
\ifnum\doz@romct>47%
\ifnum\doz@romct>59%
```

```
\ifdoz@domancaps XL\else xl\fi%
\advance\doz@romct by-60%
\else
\ifdoz@domancaps XXL\else xxl\fi%
\advance\doz@romct by-48%
\fi
\fi
\doz@quot{\doz@romct}{12}%
\loop\ifnum\doz@quotient>0%
\ifdoz@domancaps X\else x\fi%
\advance\doz@quotient by-1%
\advance\doz@romct by-12%
\repeat
\doz@modulo{\doz@romct}{12}%
\ifnum\doz@modulus=10
\ifdoz@domancaps IIX\else iix\fi%
\advance\doz@romct by-10
\fi
\ifnum\doz@modulus=11
\ifdoz@domancaps IX\else ix\fi%
\advance\doz@romct by-11
\fi
\ifnum\doz@romct>5
\ifdoz@domancaps V\else v\fi%
\advance\doz@romct by-6
\fi
\ifnum\doz@romct>3%
\ifnum\doz@romct=4%
\ifdoz@domancaps IIV\else iiv\fi%
\advance\doz@romct by-4%
\else
\ifnum\doz@romct=5%
\ifdoz@domancaps IV\else iv\fi%
\advance\doz@romct by-5%
\fi
\ifnum\doz@romct=6%
\ifdoz@domancaps V\else v\fi%
\advance\doz@romct by-6
\fi
\fi
\fi
\doz@quot{\doz@romct}{1}%
\loop\ifnum\doz@quotient>0%
\ifdoz@domancaps I\else i\fi%
\advance\doz@quotient by-1%
\advance\doz@romct by-1%
\repeat
}
\protected\def\doman#1{%
\doz@domancapsfalse%
```

```
\doz@doman{#1}%
}%
\protected\def\Doman#1{%
\doz@domancapstrue%
\doz@doman{#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
```

Now we begin the number-to-word macros. First, we define the macros which allow the user to specify his own words for each rank:

```
\def\dozrankoneword{unqua}
\def\dozranktwoword{biqua}
\def\dozrankthreeword{triqua}
\def\dozrankfourword{quadqua}
\def\dozrankfiveword{pentqua}
\def\dozranksixword{hexqua}
\def\dozranksevenword{septqua}
\def\dozrankeightword{octqua}
\def\dozranknineword{ennqua}
\def\dozranktenword{decqua}
\def\dozrankelvword{elvqua}
```

Then we define some `\if`s to help us decide how we should capitalize the end result.

```
\newif\ifDOZcaps\DOZcapsfalse
\newif\ifDozcaps\Dozcapsfalse
```

Then, we give some utility macros:

```
\def\doz@expandloop#1{\doz@xloop#1\relax}
\def\doz@xloop#1{%
\ifx\relax#1\else%
\doz@numword#1%
\expandafter\doz@xloop\fi%
}
\def\printdozrankword{}
\def\doz@rankword{%
\ifnum\doz@countchar=12
\def\printdozrankword{\dozrankelvword}%
\fi
\ifnum\doz@countchar=11
\def\printdozrankword{\dozranktenword}%
\fi
\ifnum\doz@countchar=10
\def\printdozrankword{\dozranknineword}%
\fi
\ifnum\doz@countchar=9
\def\printdozrankword{\dozrankeightword}%
\fi
\ifnum\doz@countchar=8
\def\printdozrankword{\dozranksevenword}%
\fi
\ifnum\doz@countchar=7
\def\printdozrankword{\dozranksixword}%
\fi
\ifnum\doz@countchar=6
\def\printdozrankword{\dozrankfiveword}%
\fi
\ifnum\doz@countchar=5
\def\printdozrankword{\dozrankfourword}%
\fi
\ifnum\doz@countchar=4
\def\printdozrankword{\dozrankthreeword}%
\fi
\ifnum\doz@countchar=3
\def\printdozrankword{\dozranktwoword}%
\fi
\ifnum\doz@countchar=2
\def\printdozrankword{\dozrankoneword}%
\fi
\ifnum\doz@countchar=1
\def\printdozrankword{\relax}%
\fi
\advance\doz@countchar by-1
\ifDOZcaps
```

```
\edef\printdozrankword{\uppercase{\printdozrankword}}%
\fi
\ifDozcaps
\edef\printdozrankword{\capitalisewords{\printdozrankword}}%
\fi
\ifnum\doz@countchar>0
\ \printdozrankword\ %
\else
\printdozrankword%
\fi
}%
\def\doz@numword#1{%
\IfStrEq{#1}{0}{\def\doz@numberword{zero}}{}%
\IfStrEq{#1}{1}{\def\doz@numberword{one}}{}%
\IfStrEq{#1}{2}{\def\doz@numberword{two}}{}%
\IfStrEq{#1}{3}{\def\doz@numberword{three}}{}%
\IfStrEq{#1}{4}{\def\doz@numberword{four}}{}%
\IfStrEq{#1}{5}{\def\doz@numberword{five}}{}%
\IfStrEq{#1}{6}{\def\doz@numberword{six}}{}%
\IfStrEq{#1}{7}{\def\doz@numberword{seven}}{}%
\IfStrEq{#1}{8}{\def\doz@numberword{eight}}{}%
\IfStrEq{#1}{9}{\def\doz@numberword{nine}}{}%
\IfStrEq{#1}{X}{\def\doz@numberword{ten}}{}%
\IfStrEq{#1}{E}{\def\doz@numberword{eleven}}{}%
\ifDOZcaps
\edef\doz@numberword{\uppercase{\doz@numberword}}%
\fi
\ifDozcaps
\edef\doz@numberword{\makefirstuc{\doz@numberword}}%
\fi
\doz@numberword%
\doz@rankword%
}%
\def\doznumtoword#1{%
\edef\thenumber{\basexii{\value{#1}}{X}{E}}%
\expandafter\doz@charcount\expandafter{\thenumber}%
\expandafter\doz@expandloop\expandafter{\thenumber}%
\doz@countchar=0%
}%
\def\DOZnumtoword#1{%
\DOZcapstrue%
\doznumtoword{#1}%
\DOZcapsfalse%
}%
\def\Doznumtoword#1{%
\Dozcapstrue%
\doznumtoword{#1}%
\Dozcapsfalse%
}%
```

Our last job is to make sure the semicolon (Humphrey point) works correctly as a radix point in math mode. This code is adapted from the `icomma` package by Walter Schmidt.

```
\AtBeginDocument{%
    \mathchardef\humphrey\mathcode`\;%
    \mathcode`\;="8000 %
}
{\catcode`;=\active
    \gdef;{\futurelet\@let@token\sm@rtsemi}
}
\def\sm@rtsemi{%
    \ifx\@let@token\@sptoken \else
    \ifx\@let@token\space \else
    \mathord\fi\fi \humphrey}
```

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