LaTeX Dogwagger

[A different approach to documentation]

Version 2.1.1

J.M. van Schalkwyk

February 27, 2009

Contents

1 Introduction

LaTeX Dogwagger is a solution to a pernicious problem. All too often programmers write a magnificent program, and then *document* their creation as a sort of 'addendum'. Put another way, the documentation is separate and looks tacked on, something like the stumpy, customarily docked tail of a large rottweiler. Dogwagger tries to address this by integrating the documentation and the program. The program becomes something which is pulled out of the documentation, rather than the other way around.

For example, in the wonderful programming language Perl, there's a variety of conventions that allow you to mark sections of the program with an equals sign, followed by a name. All of the subsequent code is ignored by Perl until a magic line beginning with the expression `=cut` is encountered. A separate program can then be used to pull out the *cut* sections and assemble them into some sort of documentation. This approach is called POD, or 'Plain Old Documentation'. Dogwagger, although it is written in Perl, is somewhat more sophisticated.[1]

Here's a short list of Dogwagger features:

- Complete code can be generated for a variety of languages, including C, C++, Perl, and so forth;
- Multiple files can be produced on demand, for example C++ .CPP files, and .H header files;
- Binary code can be turned into files, where required.
- The tail wags the dog.

The last feature simply means that the source code which magically generates these files is *also* the documentation. Documentation is always written in LaTeX, because we believe LaTeX is the best way of producing elegant documentation. We simply place all relevant *code* inside the LaTeX source code, within `verbatim` statements. When you submit this source code to LaTeX, it makes an appropriate document (we normally create PDF documents, for great Web portability). When you submit the same source code to DogWagger, it magically creates all of the relevant Perl, C++ or other files.

[1] In the sense of being smart and somewhat elegant, not in the original meaning of the word which is 'mixed up'. Well, come to think of it, we do mix things up slightly, but that's another story!

1.1 Advantages of Dogwagger

These are many:

- All program code and documentation is seamlessly integrated;
- Updates are concurrent. You can update program and documentation at one go, and generate program *or* documentation by simply submitting the same file to either LaTeX or Dogwagger;
- Binary code is integrated (as required) with other code, without needing fancy tools or many different types of file;
- As Dogwagger is available under the GNU public licence, it'll aways be freely available without charge.

The source code of this file (Dogwagger21.tex) is available under the GNU public licence for download from anaesthetist.com.

1.2 Disadvantages of Dogwagger

There are a few:

- If you can't write LaTeX, you're stuffed. (Learn LaTeX)!
- You have to know how to run a Perl program. At the command line say:
  `perl dogwagger21.pl`
- Your expensive word processor may give you more trouble with editing the .TEX source than an inexpensive (WinEdt) or free (MSDos Edit, vim, Notepad) word processor.[2]
- If you wish to generate binary files and include them in your documentation/program, you'll have to find a program which can UUencode. Gosh. Here's one!
- The pressure is now on you to produce good documentation. Is this a disadvantage? Microsoft might think so!
- You have to include all of the code for the program. Is this a disadvantage?

Now let's explore how Dogwagger actually works. But first, an important note...

[2] This is sure to bring out the inadequacies of people who *simply must* drive big cars, fast yachts, or fancy word processors!

1.3 GNU Public Licence

This program is distributed under the Gnu Public Licence (GPL). A copy should accompany any distribution. For details of the GPL, see Appendix A, at the end of this document.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

2 How to use Dogwagger

If you read the LaTeX source for *this file*, then you'll see Dogwagger in action! But to get a bit more than a flavour read the following documentation thoroughly. The basic idea is something like this:

1. Create your LaTeX documentation;
2. Insert arbitrary chunks of program within the documentation as `\verbatim` comments — these chunks will eventually all be concatenated into one long program file!
3. Dress things up a little.

That's really it. The dressing up is very easy indeed. The only *really important* piece of dressing up is something which must be inserted in the first four lines of the file. We call it the *title line*, and it runs like this:

```
% LaTeX DogWagger version='2.1.1' fileTarget='foo.pl'
```

If such a line isn't present, and you submit a file for Dogwagger to parse, then she will complain bitterly. You'll also get warnings if the version number is wrong. See how, in LaTeX-like style, we submit parameters in between a 'backtick' and a conventional single quote! The `fileTarget` instruction is self-explanatory.

There are a two other parameters you will almost certainly use on this important title line. They are:

- `startComment='#'`
- `noWarn='yes'`

The `startComment` option specifies how lines will be commented out. This is important because DogWagger normally writes a few lines at the start of a generated file saying where the file came from and when it was created. In addition, separate sections in the generated program code are separated by comments.

Different programming languages use different character sequences to signal a comment line — for example, C++ uses a double slash `//`, and Perl uses a hash character #, otherwise known as a 'pound' or even a 'tictactoe'! As most grown up C compilers also now permit the `//` convention, we've not implemented the old-fashioned, ugly C `/*` comment style `*/`. The default comment character is the Perl one, which is worse than useless if you happen to be creating C++ files.

By default, Dogwagger kindly warns you before it overwrites files, but you can override this behaviour using `noWarn`.

2.1 Making multiple files

Part of the way through your code, you may wish to terminate the current program file you're creating (and documenting), and start a new file. For example, you may have discussed (and created) the main CPP program, and now wish to do the same for the .H header file. This is pretty easy. In the line *immediately preceding* the next `verbatim` section, insert a commented line like the following:

```
% DogWagger newTarget='foobar.h' newComment='//'
```

We will refer to such a line as *preamble*. Remember to comment the line out in LaTeX using a percentage character as the *first character* of the preamble line. See how we use `newComment` and `newTarget` to remind ourselves that this isn't the first file specified in the title line, but a subsequent one.

Files within directories

We deliberately don't encourage files generated by DogWagger to be written to obscure locations. They should generally be placed in the current directory, but if a subdirectory exists, then the files can be written to that subdirectory thus:

```
% DogWagger newTarget='foo/bar.c' newComment='//'
```

This statement creates the file *bar.c* in the subdirectory *foo*.

2.2 Suppressing unwanted *verbatim* sections

Within your normal LaTeX documentation, you may wish to include a `verbatim` section which *must not* appear within the final program you will generate. Here's how:

```
% DogWagger dogsAllowed='no'
```

Simply insert the above line immediately prior to the verbatim section you wish to suppress!

2.3 Minor frills and restrictions

Each `verbatim` section begins with `\begin{verbatim}` and ends with `\end{verbatim}`. In between there *must* be at least one whole line of data, otherwise DogWagger will choke. Another minor irritation is that even if verbatim statements have been commented out in LaTeX, they will still be seen by DogWagger (we might consider revising this)!

There are several little conveniences in DogWagger. You can label the comment at the start of each section using a line like the following *immediately preceding* a `verbatim` section:

```
% DogWagger sectionTitle='Fred'
```

Again, the line preceding the verbatim section is commented out in LaTeX. We then use `sectionTitle` to provide a label. We can do slightly more fancy things:

```
% DogWagger sectionTitle='Fred: Section $[SECTION]'
```

...which actually uses Dogwagger's internal section counter to replace `$[SECTION]` with the relevant section number.

Writing code as a single line

Occasionally it's convenient to write several lines of verbatim text as a single line of output. Dogwagger to the rescue with the `oneLine='yes'` command! Note that in this mode, trailing spaces count, but the leading spaces on the next line are removed. All other whitespace is preserved *as is*.

2.4 Deferred code

In version 2.0 of Dogwagger, we introduced the ability to move code sections down below other sections. In other words, we can now *defer* writing of a code section until other sections on which it *depends* have been written to the output file.

For example, when discussing SQL code, we might wish to talk about the main table first, but in the final code we will first want to define the minor tables on which the main table depends!

To make use of this facility, use the following commands:

```
% DogWagger dependsOn='alpha'
```

...or even, if something depends on several other sections:

```
% DogWagger dependsOn='alpha,beta,gamma'
```

The name of the section depended on is then given by:

```
% DogWagger myName='alpha'
```

When a dependsOn statement is encountered the assumption is made that *all*

of the names depended on have *not yet been defined*! If any of them had already, then the smart user would simply leave them out!

Note that an item can have both a myName and dependsOn values. In this case, the name is kept pending until all dependsOn values have been fulfilled, at which point the item is written to output, and then only is the name of the item itself resolved. (Self-dependence will necessarily result in an unresolved dependency, and thus an error).

2.5 Including binary files

Occasionally it may be necessary to include a binary file with your source code. Because such files are inscrutable (and potentially harmful) this is a practice to avoid, but unfortunately it's sometimes vital to include small binary files. Dogwagger meets this need by allowing UUencoded files to be included in `verbatim` sections thus:

```
% DogWagger newTarget='uudecode'
```

In other words, the only 'filename' which is reserved is `uudecode`. If this name is specified, then (as is usual for uunecoded files) the filename is picked out of the subsequent uuencoded information (located within the verbatim statement). You'll find a (uuencoded) uunecoding program in Appendix B!

Note that for reasons best know to us (if at all) that the first file specified (in the header line) *cannot* be a uuencoded file. Regard this as a feature rather than a bug!

2.6 Making a debug version!

Here's a command which you can *only* include in the title line:

- `include='everything'`

This wrinkle allows us to create two versions of code, a *debug* version, and a production version. By default, if you *omit* the above command from the title line, then the production version is created. If you include it, then we make a debug version. And what's the difference? Well, if Dogwagger encounters a line within a `verbatim` section which begins with the sequence:

```
+OPTIONAL
```

...then by default all of the code (including the OPTIONAL statement itself) is *omitted* until the terminating sequence:

```
-OPTIONAL
```

is encountered later on. ‘Including everything’ forces Dogwagger to include the optional code contained between these two keywords.

In many languages, there are ways of creating debug versions, for example the C++ `#define` followed by `#ifdef` and so on, but our way is more explicit and simpler.[3]

[3] We considered having an optional parameter after the OPTIONAL statement to allow multiple versions, but rejected this as extremely silly.

3 The source code of Dogwagger

Here's the Dogwagger version 2.1 code. Run the program from the command line using

```
perl Dogwagger21.pl
```

First we have the conventional shebang line, followed by the packages required. We use the Tk toolkit to give us a simple graphical user interface.

```
#!/usr/local/bin/perl -w

use strict;
use Tk;
require Tk::Dialog;
require Tk::Toplevel;
# require Tk::Font;

  # This program is distributed under the Gnu Public Licence (GPL).
  # You should have received a copy of the GNU General Public License
  # along with this program; if not, write to the Free Software
  # Foundation, Inc., 59 Temple Place - Suite 330,
  # Boston, MA  02111-1307, USA.

my $ERR;               # ugly global error;
my $ERRCOUNT;          # similarly nasty;
my $LINECOUNT;         # this is also a nasty global;
my $SECTIONTITLE;      # me too.

my $MAJORVERSION = 2;
my $MINORVERSION = 1;
my $TINYVERSION  = 1;   # version 2.1.1
my $BUG = 0;            # are we debugging (0=no)
```

3.1 Initialisation

We keep a record of what happened in a file *WAGLOG.LOG*. After getting the time (and printing it), we create version 2.0 arrays for deferred writing of code (See section 2.4).

```
my $filelog;
$filelog="WAGLOG.LOG";
open FILELOG, ">$filelog" or
     die "*CRASH* Could not open LOG $filelog :$!\n";
print FILELOG "LaTeX DogWagger, Version \
  $MAJORVERSION.$MINORVERSION.$TINYVERSION\n";
```

```
my $TODAY = &GetLocalTime();
print ("\n TODAY: $TODAY\n");
my @CHILDREN;
my @DEPENDENCIES;
my @PENDINGNAME;
```

The deferred arrays are CHILDREN (an array of sections which depend on other named sections, and haven't yet been written to file), DEPENDENCIES which stores the corresponding names *depended on*, and PENDINGNAME, which stores the name of each child, if that child has a name.

Each child contains multiple lines which will only be written to file when all of the dependencies of the child have been satisfied. As each name is encountered, it is removed from each dependency list where it occurs. If a dependency list becomes EMPTY in this process, the corresponding item is written to output.

4 Creating the main window

Next, we use Tk to open up a user interface. The ugly global variable `fred` is used for filename input — how imaginative!

```
my $fred;  # file name
$fred = "";

my $MAINW = new MainWindow;
   $MAINW->geometry('300x200');  # dimensions
   $MAINW->geometry('+80+30');   # screen offset!
   $MAINW->title(
     "DogWagger Version $MAJORVERSION.$MINORVERSION.$TINYVERSION");
   $MAINW->focusFollowsMouse;    # change focus mode

my $FIRSTARG;
$FIRSTARG = $ARGV[0]; # allow command line for simple stuff!
if ((defined $FIRSTARG) && ((length $FIRSTARG) > 0))
   {  # &Alert ($MAINW, "First argument is $FIRSTARG");
     $fred = $FIRSTARG;
   }
```

We put the control buttons in a separate frame in the main Tk window.

```
my $bottomFrame = $MAINW->Frame();
   $MAINW->Label( -text => 'Enter source file name')->pack();
my $txt = $MAINW->Entry( -textvariable => \$fred)->pack(-padx => 50,
                                                        -pady => 15,
                                                        -ipadx => 5);
   $txt->configure (-validatecommand => [ \&CheckFred, $MAINW],
```

```
                                -validate => 'focusout');
my $goBut = $MAINW->Button( -text => 'Wag',
                            -command => [ \&WagTheDog, $MAINW ] );
   $goBut->configure(-background => 'green');
   $goBut->configure(-width => 20);
my $quitBut = $bottomFrame->Button(-text => 'Quit',
              -command => [ \&ByeForNow, $MAINW ] );
   $quitBut->configure(-background => 'red');
   $quitBut->configure(-width => 20);
$quitBut->pack();
$goBut->pack();
$goBut->focus(); # version 2.1
$bottomFrame->pack(-side => 'bottom', -fill => 'both',
                   -pady => 20);
MainLoop;
```

The controls are really simple — a text Entry box for the file name, an execution (‘Wag’) button, and a red quit button. And that’s really that for the main section. Next we have the main function which does the ‘wagging work’.

5 The principal function

As the name suggests, `WagTheDog` does the work. There are several ugly features apart from its length, including use of the nasty global `fred`. We submit the current window, `thisW`.

```
sub WagTheDog
{
  my($thisW);
    ($thisW)=@_;
  my ($MANDATORY, $OPTN);
  $OPTN = 0; # default is capture
  print FILELOG "\n ------------------------";
  &Debug($thisW, "\n\n You specified <$fred>\n");
  if (length $fred < 1)
     { &Alert($thisW, "First enter file name, e.g. PerlPgm.tex");
       return;
     };
  my($FRED, $hotline, $i);
  $FRED = $fred;
  $ERRCOUNT = 0;
```

5.1 Preliminaries

After some debugging statements and a check for the presence of a filename string, we clear the various arrays, and set up `myNam`, which stores a pending name about to be resolved. We only 'resolve' myNam once the code associated with the name *has been* written, at which point all of the dependent code is written, if indicated.

```
  @CHILDREN = ();
  @DEPENDENCIES = ();
  @PENDINGNAME = ();
    $CHILDREN[0]='';
    $DEPENDENCIES[0]='';
    $PENDINGNAME[0]='';
  my($myNam);
```

Let's open the source file, failing if this opening fails:

```
  $LINECOUNT = 0;
  $ERR = 0; #hideous
    open FRED, $FRED
         or &GlobalError("Could not open source $FRED :$!");
    if ($ERR)
       { $ERRCOUNT ++; # bump error count
         &Alert($thisW, $ERR);
         return;
       };
```

5.2 Finding the header line

Next, scan through the first four lines for the header line, failing if the header line isn't found.

```
    $i = 4;
    while ($i > 0)
      { $_ = <FRED>;
        $LINECOUNT ++;
        &Debug($thisW, "$_");
        if ( /\%.*LaTeX DogWagger/ )
           { $i = 0;  # force end
           };
        $i --;
      };
if (! $i)  # if DogWagger found, $i should be -1.
   { &Caution($thisW, "DogWagger data not found in <$fred>");
     close FRED;
     return;
   };
$hotline = $_;  # redundant
```

Okay, we could even look for sequences such as \%, but we won't get too anal.

5.3 Checking the version

We check for version compatibility, and also extract the name of the target file, comment character sequence, and warning/mandatory flags. The MANDATORY variable tells us whether to insert debug code (bracketed by +/- OPTIONAL statement lines).

```
my ($version, $DOGFILE, $startComment, $nowarn,
    $startFileTxt, $endFileTxt, $sft, $eft,
    $endComment);
my($majorVersion, $minorVersion);
$startFileTxt = '';
$endFileTxt = '';
```

The four variables $startFileTxt, $endFileTxt, $sft and $eft were added in version 2.1. They allow start and end text to be inserted into a file. The way we work things, the values are reset to the null string after the closure of the current file, so for each file with starting and/or ending text, the values must be specified anew!

```
($version, $DOGFILE, $startComment, $nowarn, $MANDATORY,
 $sft, $eft, $endComment) = &ReadHeader($hotline);
if (! $MANDATORY)
   { &Debug($thisW, "\n Optional text NOT included");
   };
$_ = $version;
/(.+)\.(.+)\.(.+)/;  # pull out major and minor version numbers:
$majorVersion = $1;
$minorVersion = $2;  # ignore trivial version number = $3

if ($majorVersion > $MAJORVERSION)
   { &Caution($thisW,
   "Warning: DogWag(V$MAJORVERSION.$MINORVERSION \
   won't support all features of V$majorVersion.$minorVersion");
   } else
   { if (  ($majorVersion == $MAJORVERSION)
         &&($minorVersion > $MINORVERSION)
        )
        { &Caution($thisW, "Caution: minor version switch.\
          Problems may abound!");
        };
   };
```

We give appropriate warnings if the major or minor version numbers of Dogwagger and the file being translated aren't compatible. Trivial version numbers (the third part of the dotted version number) are ignored.

5.4 Opening the target file

We open the target file, using the name provided, and fail if this fails.

```
my ($c, $ok, $wagline, $ec);
$c  = $startComment; # shorter. hmm. clumsy.
$ec = $endComment;   # version 2.1
if (! OpenTargetFile($thisW, $DOGFILE, $c, $FRED,
                     $nowarn, $sft, $ec))
     { return;  #fail
     };
```

5.5 Read source and process

Now we're ready to read in the source file, and process it. There is a small 'bug' in that a verbatim statement which has been commented out will still trigger action. Hmm. A biig `while` statement surrounds everything, within which we read each line in turn and process it. Several startup flags control interpretation, the most

important being `ishot`, which determines whether we are actively writing lines, or just throwing away LaTeX text.

```
     my($ishot, $hotdata, $chomper, $chomped);
     my($nodogs);
     my($SECTION);
     $SECTION = 1;

     $ishot = 0;
     $chomper = 0;  # default is OFF
     $chomped = 0;
     $SECTIONTITLE = ’’; # default is empty
     $ok=1;
     $nodogs=0; # default

while($ok)
  { $_ = <FRED>;
    $LINECOUNT ++;

    if (! defined)
       { $ok = 0;
       } else
       { if (! $ishot) # if not writing
            { if ( /\\begin\{verbatim\}(.*)/ )
                 { if (! $nodogs)
                      { $ishot = 1; # turn on
                        $hotdata = $1;
                        $SECTION = &PrintSectionHeader($c, $SECTION, $ec);
                        print DOGFILE $hotdata; # clumsy but explicit
                      };
                 } else
# see comment [1] below
{   my($depOn);
    $myNam = ’’;
    $depOn = ’’;
    $nodogs = 0;
    if (/^\%.*DogWagger/)
       { if ( /dogsAllowed=\‘no\’/)
            { $nodogs = 1;
            } else
            { $wagline = $_;
              if (/dependsOn=\‘(.+?)\’/)
                 { $depOn = $1;
                   print FILELOG "\n Section dependencies <$depOn>";
                 };
              if (/myName=\‘(.+?)\’/)
                 { $myNam = $1;
                   print FILELOG "\n Section name: >$myNam";
```

```
   };
if (/noWarn=\‘(.+)\’/)
    { if ($1 eq ’yes’)
         { $nowarn = 1;
         } else
         { $nowarn = 0; # default (safer)
         };
    };
if (/oneLine=\‘yes\’/)
   { $chomper = 1; # turn on!
     print FILELOG " (chomp)";
   };
if (/sectionTitle=\‘(.+?)\’/) # self-explanatory
   { $SECTIONTITLE = $1;
   };
if (/newComment=\‘(.+?)\’/) # new comment string!
   { $startComment = $1; # note usage!
   };
if (/endComment=\‘(.*?)\’/)
   { $endComment = $1;
   };
if (/startFile=\‘(.*?)\’/) # new file start, can be null!
   { $startFileTxt = $1;   # ver 2.1
     $startFileTxt =~ s/\\n/\n/mg; # CR’s !!
   };
if (/endFile=\‘(.*?)\’/)   # similar, end file
   { $endFileTxt = $1;   # ver 2.1
     $endFileTxt =~ s/\\n/\n/mg; # CR’s !!
   };
if (/newTarget=\‘(.+?)\’/)
   { $_ = $1;
     if (/uudecode/) # if uudecoding ?!...
      { my ($ufile, $umode, $uout) = Uudecode($MAINW);
        if (length $ufile > 0)
           {
             print FILELOG ("\n Uudecoding <$ufile> mode $umode");
             open UFILE, ">$ufile" or &GlobalError("UU");
             binmode UFILE; # NB otherwise DOS stuffup!
             print UFILE $uout;
             # hmm what about the unix mode (opening?)
             close UFILE;
           }; # ??? also print to FILELOG?
      } else  # close current, open new!
      { $DOGFILE = $_; # retain new name
        &CloseDogFile($thisW, $c, $eft, $ec); # close previous
        $sft = $startFileTxt;
        $eft = $endFileTxt;
        $startFileTxt = ’’; #
```

```
                    $endFileTxt = '';   # reset for next.
                    $c = $startComment; # only now alter comment!
                    $ec = $endComment;  # v2.1 likewise
                    $endComment = '';   # reset!!
                    print FILELOG ("\n Comment format is <$c" . "COMMENT" . "$ec
                    if (! OpenTargetFile($thisW, $DOGFILE, $c, $FRED,
                           $nowarn, $sft, $ec))
                       { return;  #fail
                       };
                 };
              };
            # ----------------------------------------
            # here if more tests, use $wagline, not $_ !
            # ----------------------------------------
          };
      };
   if (length $depOn > 0)  # if dependency
      { if (! &StoreChild ($myNam, $depOn)) # keep whole
          { &Caution($thisW, "WARNING: \
          Input file <$FRED> terminated unexpectedly!");
            close FRED;
            close DOGFILE;
            return; #fail!
          };
        $myNam = ''; # cannot YET resolve (stored not printed)!
      };
# END AMENDMENT V2.0 9/9/2005.
                    };
```

Comment 1 In the above, if we're not 'hot' (writing) we look for the 'begin' verbatim statement. If this verbatim statement is present, we turn on the heat. Otherwise, we check to see whether we're dealing with a DogWagger line *immediately preceding* a begin verbatim statement, in other words, we check for a preamble line. See how, for a preamble line to be detected, the first character on the line must be a percentage character.

The above code performs a variety of checks, including for `dogsAllowed`, `dependsOn`, `myName`, `noWarn`, `newComment`, `newTarget`, and the obscure `oneLine`. See how within this section we have the ability to UUdecode a whole section of many lines, writing the file and then just bashing on with the current file!

If we *are* busy writing lines to output (are hot) we check for the end of a verbatim statement. If this is the case but an OPTION statement is still active we fail but otherwise we go cold after resolving the name and writing code (FixName), if appropriate.

```
} else # are hot!
```

```
 { if ( /(.*)\\end\{verbatim\}/ ) # end verbatim?
      { if ($OPTN) # OPTION still on?
           { &Alert ($MAINW,
              "Optional text not closed. See log!");
             &GlobalError(
              "\n ERROR: NO option closure, line $LINECOUNT");
             $OPTN = 0;
           };
        $hotdata = $1;
        print DOGFILE $hotdata; #last chunk
        if (length $myNam > 0) # if name defined
                  { $SECTION = FixName($myNam, $c, $SECTION, $ec);
                  };
        $ishot = 0;   # turn off.
        $chomper = 0; # back to default
        $chomped = 0; # redundant.
        $SECTIONTITLE = '';
      } else
      # see Comment[2] below
      { if ($chomped)
           { / *(.*)/;  # even allow null line ??
             $_ = $1;   # remove leading spaces!
           };
        if ($chomper)  # v2.0 (23/8/2005): chomp line feed if indicated!
           { chomp;
             $chomped = 1; # signal we've just chomped
           };
        if ( /^\s*\+OPTIONAL/)
           { $OPTN = 1;
             $_ = "";
           };
        if ( /^\s*-OPTIONAL/)
           { $OPTN = 0;
             $_ = "";
           };
        if ($MANDATORY || ! $OPTN)
           { print DOGFILE $_; # write to output
           };
      };
 };
            };
       }; # end of biig while stmt.
  close FRED;
  &CloseDogFile($thisW, $c, $eft, $ec);
  &Caution($thisW, "Done!");
  return;
}
```

Comment 2 In the case where we are hot but it's not the end, we need to check

some conditions. If the last line was chomped as part of a oneLine, we remove leading spaces. If we are busy chomping terminal carriage returns, we chomp. And we process OPTIONAL statements as appropriate.

We write the line to output (as appropriate), and then bash on. (If MANDATORY is on, we print regardless; if it's off then we only print non-optional lines).

6 Reading the header

As discussed in the introductory section, we must accommodate the various header line options. We obtain the version, fileTarget and startComment values in the following routine:

```
sub ReadHeader
{
  my ($hotline);
     ($hotline) = @_;
  my ($ver, $target, $comment, $nowarn, $mandatory,
      $sft, $eft, $endComment);

  $ver = 0;
  $target = '';
  $comment = '#';
  $nowarn = 0;
  $mandatory = 0;
  $sft = '';
  $eft = '';
  $endComment = '';

  $hotline =~ /version=\'(\d+\.\d+\.\d+)\'/;  # version
  $ver = $1;

  $hotline =~ /fileTarget=\'(.+?)\'/;         # file name
  $target = $1;

  $hotline =~ /startComment=\'(.+?)\'/;       # comment
  $comment = $1;

  if ($hotline =~ /include=\'everything\'/)
     { $mandatory = 1;
     };

  if ($hotline =~ /noWarn=\'yes\'/)
     { $nowarn = 1;
     };
  if ($hotline =~ /endComment=\'(.*?)\'/)
     { $endComment = $1;
     };
  if ($hotline =~ /startFile=\'(.*?)\'/)      # ver 2.1
     { $sft = $1;
       $sft =~ s/\\n/\n/mg; # CR's !!
     };
  if ($hotline =~ /endFile=\'(.*?)\'/)      # ver 2.1
     { $eft = $1;
       $eft =~ s/\\n/\n/mg; # CR's !!
```

```
    };

  return ($ver, $target, $comment, $nowarn, $mandatory,
          $sft, $eft, $endComment);
}
```

Straightforward, really. In version 2.1 we add the option to specify the very first few characters of the file using the *startFile* option. This is useful for HTML and PHP. See also the corresponding *endFile* option for terminating the last chunk of a file.

7 Miscellaneous routines

The following are rather trivial routines:

7.1 Confirm an action

Given a window and a message, obtain confirmation.

```
sub Confirm
{ my ($thisW, $msg);
     ($thisW, $msg) = @_;

    my $D = $thisW->Dialog(
                -title => "Confirm your choice",
                -text  => "$msg",
                -default_button => 'No',
                -buttons        => ['No','Yes'],
                      );
    $_ = $D->Show();  # use Show for Tk-b9.01
    if ($_ eq 'Yes')
       { return 1;
       };
    return (0);
}
```

7.2 Alert

Alert the user with a warning.

```
sub Alert
{ my ($thisW, $msg);
     ($thisW, $msg) = @_;

    my $D = $thisW->Dialog(
                -title => $msg,
                -text  => "$msg",
                -default_button => 'OK',
                -buttons        => ['OK'],
                      );
    $D->title('Note..');
    $D->Show;
}
```

A standard Tk Dialog.[4]

[4] For the pedant, part 7 has been removed!

7.3 Caution — Alert with print

‘Caution’ is similar to `Debug`, but always generates an alert, regardless of the debugging status.

```
sub Caution
{ my ($thisW, $msg);
     ($thisW, $msg) = @_;
  print FILELOG "\n$msg";
  &Alert($thisW, $msg);
}
```

7.4 Read the time

```
sub GetLocalTime
{  my ($sec, $min, $hour, $mday, $mon,
       $year, $wday, $yday, $isdst);
($sec, $min, $hour, $mday, $mon,
       $year, $wday, $yday, $isdst) = localtime(time);

  $year += 1900;      #fix y2k.
  $mon ++;            #january is zero!
  return ("$year-$mon-$mday $hour:$min:$sec");
}
```

7.5 Farewell

We simply close the file log and exit.

```
sub ByeForNow
{ my ($thisW);
     ($thisW) = @_; # unused at present.

close FILELOG;
exit;
}
```

7.6 A clumsy error-related hack

The following clumsy hack should be fixed. See (foul) usage!

```
sub GlobalError
{ my ($msg);
     ($msg) = @_;
  print FILELOG "$msg";
  $ERR = $msg; #ugly global ?!
}
```

7.7 Debugging

Debug simply logs a statement. If the BUG variable is set, then an alert message is displayed, but this is only used for detailed debugging.

```
sub Debug
{ my ($thisW, $msg);
     ($thisW, $msg) = @_;

  print FILELOG "$msg";

     if (! $BUG)
        { return;
        };
  &Alert($thisW, $msg);
}
```

7.8 More debugging

The following is only used for a clumsy debug where we displaying the value of the filename `fred` as an Alert, if you manually edit out the # from the relevant Perl line.

```
sub CheckFred
{
  my ($thisW);
     ($thisW) = @_;
#  &Alert($thisW, "Value is <$fred>");
}
```

7.9 Ask — datum input

Given a window, title and default text value, `Ask` gets user input. We might use this to acquire a file name, but at present this is an unused routine.[5]

```
sub Ask
{ my ($win, $title, $default);
     ($win, $title, $default) = @_;

  my ($db, $fred);
  my ($e);
  $fred = $default;
```

[5] Remove me!

```
  $db = $win->DialogBox( -title => $title,
                         -buttons => ["OK", "Cancel"]
                         );
  $e = $db->add('Entry',
                -textvariable => \$fred)->pack(-padx => 50,
                                               -pady => 15,
                                               -ipadx => 5);

  my $choice = $db->Show;
  if ($choice eq "Cancel")
     { return ("");
     };
  return ($fred);
}
```

7.10 Store array of lines

Here we keep 'child' (dependent) lines in an array element, to be resolved later when all dependencies are met. The index of the topmost child is given by `$#CHILDREN`. The list of dependencies contains elements separated by commas, and there are starting and terminal commas (!) too.

```
sub StoreChild
{  my ($pendingName, $dependencies);
      ($pendingName, $dependencies) = @_;
   my ($idx, $child);
   $idx = 1+$#CHILDREN;
   print FILELOG "\n Line $LINECOUNT: Storing child[$idx]";

   $_ = <FRED>; # first line *must* be begin verbatim
   if ( /\\begin\{verbatim\}(.*)/ )
      { $child = $1;  # keep rest of line
      } else
      { print FILELOG
     "\n ERROR at line $LINECOUNT: \
       no verbatim stmt on 1st child line!";
        $ERRCOUNT ++; # bump error
        print FILELOG "<$ERRCOUNT!>";
        print FILELOG "<$_>";
        return 1; # not fatal, per se.
      };

   $DEPENDENCIES[$idx] = ",$dependencies,";
   $PENDINGNAME[$idx] = $pendingName;
   $CHILDREN[$idx] = ''; # default nothing

     my($ishot, $hotdata);
```

```
    my($nodogs, $ok);
    my($SECTION);
    $SECTION = 1;

    $ishot = 0;
    $ok=1;
    $nodogs=0; # default

    while($ok)
      { $_ = <FRED>;
        $LINECOUNT ++;

        if (! defined)
           { return 0; # fail
           } else
           { if ( /(.*)\\end\{verbatim\}/ )
                { $ok = 0;
                } else
                { $child = "$child$_";  # concatenate, unchomped
                };
           };
      };
  $CHILDREN[$idx] = $child; # store away lines to be printed
  return 1; # success!
}
```

As things stand, there is a bug in the above, as optional (debug) code cannot be included in a child section!

7.11 Section labelling

FixName walks through all dependencies, resolves them (where relevant), and on resolution, writes the relevant child code to output. (A ‘child’ is a section which depends on other sections, and must not be written before these sections have been identified and written).

```
sub FixName
{ my ($fname, $morenames, $c, $SECTION, $ec);
  ($fname, $c, $SECTION, $ec) = @_; # get name argument
  $morenames = ",$fname,";

  my ($idx);
  while ( $morenames =~ /^(.*,)(.+),$/ ) # split off last name
    { $fname = $2;
      $morenames = $1;
      $idx = $#CHILDREN;
      while ($idx > -1)
```

```
        { if ($DEPENDENCIES[$idx] =~ /(.*,)$fname,(.*)/ )
             { $_ = "$1$2";  # if name in list, clip out
               print FILELOG " (dependency <$fname> resolved for child $idx)";
               $DEPENDENCIES[$idx] = $_;
               if ( /^,$/ ) # if all resolved
                  {  print FILELOG "\n Writing child[$idx] ";
                     $SECTION = &PrintSectionHeader($c, $SECTION, $ec);
                     print DOGFILE $CHILDREN[$idx];
                     $CHILDREN[$idx] = ''; # (might even remove)
                     # ....WAIT! HERE MUST RESOLVE THIS ONE:
                     if (length $PENDINGNAME[$idx] > 0)
                        { $morenames = "$morenames$PENDINGNAME[$idx],";
                          $PENDINGNAME[$idx] = '';  # clear me!
                        };
                  };
             };
          $idx --; # move down to next
        };
    };
  return $SECTION;
}
```

7.12 Check for unresolved dependencies

At the end, we have to make sure that all dependencies have been resolved, or signal an error. Errors are also written to the log.

```
sub CheckUnresolved
{ my($idx);
  $idx = $#CHILDREN;
  my ($errcnt);
  $errcnt = 0;

  while ($idx > -1)
    {
      if (length $CHILDREN[$idx] > 0)
         { print FILELOG "\n\n *** ERROR *** \n\n Unresolved code: \n ";
           print FILELOG "Dependencies: <$DEPENDENCIES[$idx]> \n";
           print FILELOG "Name: <$PENDINGNAME[$idx]> \n <Code: <---\n ";
           print FILELOG $CHILDREN[$idx];
           print FILELOG "$\n ---> Code ends> \n\n";
           $errcnt++;
         };
      $idx --;
    };
  return $errcnt;  # number of errors, 0=ok.
}
```

8 Handling of Multiple files

This section is is a consequence of the version 2 ability to generate multiple files from a single .TEX source. We chose the simple option of closing the first file and then opening and writing the next one, rather than having multiple dangling file handles. The sole exception to this rule is that if we are writing a UUdecoded binary file, we don't fiddle with the current open file.

8.1 Open the target

We open a target file. If opening fails, the very clumsy GlobalError invocation will be entered and the nasty global $ERR will then be altered, allowing us to detect the existence of a file. This ugly hack should be rewritten at some stage.

```
sub OpenTargetFile
{
  my ($thisW, $DOGFILE, $c, $FRED, $nowarn, $sft, $ec);
     ($thisW, $DOGFILE, $c, $FRED, $nowarn, $sft, $ec) = @_;
     my($ok);

  $TODAY = &GetLocalTime();
  $ERR = 0; # clumsy test for existence of file
     open DOGFILE, $DOGFILE or &GlobalError("OK");
     if (! $ERR)  # if file exists...
        { close DOGFILE;
          if ($nowarn)
             { $ok = 1;
             } else
             { $ok = &Confirm ($thisW,
                 "Overwrite <$DOGFILE>? Are you sure?");
             };
          if (! $ok)
             { &Caution($thisW, "File $DOGFILE NOT overwritten!");
               # amendment v2.1.1 (2008-08-09): write to junk file:
               $DOGFILE = 'JUNK.JUNK';
             };
        };

     $ERR = 0;
     open DOGFILE, ">$DOGFILE" or
        &GlobalError("Could not open target $DOGFILE :$!");
     if ($ERR)
        { $ERRCOUNT ++; # bump error count
          &Alert($thisW, $ERR);
          return 0; #fail
        };
```

```
  print FILELOG "\n\n Opened target file: <$DOGFILE>";
  print DOGFILE $sft; # very first text eg. for PHP.
  print DOGFILE
    "$c Generated by LaTeX DogWagger Version " .
    "$MAJORVERSION.$MINORVERSION.$TINYVERSION from file <$FRED>$ec\n";
  print DOGFILE "$c Date: $TODAY $ec\n";
  print DOGFILE "$c Do NOT edit this file. Edit the LaTeX source!!$ec\n";

  return 1; # success
}
```

On opening the target file, we write several lines to this file (DOGFILE) using the comment character(s) to rem out the lines.[6]

8.2 Close file

Simply close the output file handle DOGFILE.

```
sub CloseDogFile
{ my ($thisW, $c, $eft, $ec);
     ($thisW, $c, $eft, $ec) = @_;

  $ERRCOUNT += &CheckUnresolved();
  if ($ERRCOUNT > 0)
     { print FILELOG "<$ERRCOUNT!>";
       &Caution($thisW,
         "WARNING: Error count $ERRCOUNT. See WAGLOG.LOG!");
       print DOGFILE
         "\n\n$c -- WARNING: $ERRCOUNT ERROR(S). See log!$ec\n";
       $ERRCOUNT = 0; # clear me.
     };

     print DOGFILE "\n$c ---END OF FILE---$ec\n";
     print DOGFILE $eft; # version 2.1
     close DOGFILE;      #
     print FILELOG "\n <END OF FILE>";
}
```

[6] As things stand, we don't allow suppression of this header except in binary files, but it's conceivable that at some stage we will need to modify DogWagger to permit this option.

9 Trivial amendments

9.1 Print a section header

We now have the ability to label sections with pre-defined text (commented out). Remember that LINECOUNT is an ugly global. We submit the section count SECTION, and return this value incremented by one. We also permit insertion of this section count.

```
sub PrintSectionHeader
{ my($c, $SECTION, $ec);
    ($c, $SECTION, $ec) = @_;

  if (length $SECTIONTITLE > 0)
     { $_ = $SECTIONTITLE;
       if (/\$\[SECTION\]/) # if contains section count
          { s/\$\[SECTION\]/$SECTION/;
          };
          print DOGFILE "\n$c$_$ec\n";
     } else
     { print DOGFILE "\n$c --- <Section $SECTION> --- $ec\n";
     };
  #  if (($SECTION % 10) == 1) # removed in ver 2.1
  #     { print FILELOG "\n";
  #     };
  print FILELOG "\n" . "line $LINECOUNT: written as section [$SECTION]";
  $SECTION ++;
  return $SECTION;
}
```

10 Binary encoding and decoding

Responding to the need to write binary code from our TEX source:

10.1 UUdecode

We read the global file handle FRED, mandating that the initial line is the 'begin verbatim' line. The line immediately after this must contain the UUencoded header line. The subsidiary routine UUdecodeLine returns not only decoded text, but also an error code. If the error code is less than zero, an error has occurred; if the error code is zero, then the subsequent line *must* be an `end` statement signalling the end of the UUencoded section!

```
sub Uudecode
{ my ($MAINW) = @_;

  my ($filename, $mode, @rslt);
  my ($line, $decoded, $err);
  $filename = "";
  $line = <FRED>; # this should be \begin{verbatim} line:
  if ($line !~ /\\begin\{verbatim\}/ )
     { &Alert ($MAINW, "Uudecode: no verbatim <$line>");
       return ("", "", "");
     };

  $line = <FRED>; # MUST be header!
  chomp($line);
  $line =~ /begin\s+(\d{3})\s+(.+)/;
  if (! defined $1)
     { # here write error!
       &Alert ($MAINW, "Uudecode: bad first UU line <$line>");
       return ("", "", "");
     };
  $mode = $1;
  $filename=$2;
  $err = 1; # -ve will signal failure

  while ( $line = <FRED> )
    { # hmm what if extra 0xD ?
      last if (! defined $line); # ??
      chomp($line);
      last if ($line =~ /^end/);
      if (! $err) # bad if err zero
         { &Alert ($MAINW, "Uudecode: end stmt not seen!<$line>");
           last;
         };
```

```
      ($decoded, $err) = UudecodeLine($line);
      # nb if $err is zero, next line must be /^end/!
      if ($err < 0)
         { # here could write error!
           if ($err == -1)
              { $err = "Bad line";
              }
           elsif ($err == -2)
              { $err = "silly length($decoded)";
              }
           elsif ($err == -3)
              { $err = "lengths don't match($decoded)";
              };

           &Alert ($MAINW, "Uudecode: error $err in <$line>");
           last; # terminate
         };
      push @rslt, $decoded;
    };
  return ($filename, $mode, join("",@rslt));
}
```

10.2 UUdecoding

Here's the routine that actually does the business of UUdecoding. We've kept this very simple, based on publicly available code.[7] On most UNIX/Linux systems, UUencoding and decoding should be readily available, but for DOS uuencoding, try e.g. this program.

```
sub UudecodeLine
{
  my ($line) = @_;
  my ($charlen);
  my ($decoded, $ld);

  $line =~ /(.).*\`*$/; # remove terminal backticks too!
  if (! defined $1)
     { return ("", -1); # dud line!
     };
   $charlen = (ord($1) - 32) & 077;
  if ($charlen == 0)
     { # ie terminal line with single backtick:
       # no error, but END!
       return ("", 0);
     };
```

[7] Source: http://search.cpan.org/src/ANDK/Convert-UU-0.52/lib/Convert/UU.pm

```
  if (($charlen > 45) || ($charlen <0))
     { return ("$charlen($1)", -2); # bad length
     };
  # convert to number, then count of characters encoded;
  $decoded = unpack("u", $line); #uudecode!
    # MUST CHECK ON HOW ROBUST unpack IS???
  $ld = length $decoded;
  if ($ld != $charlen)
     { return ("$ld:$charlen:$decoded", -3); # length doesn't match!
     };
  return ($decoded, 1); # success!
}
=thelastpage
```

See the archaic use of octal. But it works.

A brief note on uuencoding/decoding

This description assumes you understand hexadecimal and ASCII

A uuencoded file consists of:

1. A header line;
2. A body;
3. A trailer line.

All other lines must be ignored Lines may end with 0x0D, 0x0A, or any combination of the two (ie carriage return and/or line feed). From now on we'll call the end of line character(s) the 'endline'. Conventionally this should be *encoded* as simply 0x0A. Here are the details:

1. The header line. This contains three items *separated* by spaces (0x20):
   (a) The five character string 'begin' (no quotes around it)
   (b) Three digits, each in the range 0..7 i.e. an octal number
   (c) The file name in ASCII (potential for trouble here!)

   The header line terminates with an endline.
2. The body. This contains one or more lines, each ending with an endline. For all but the last data line, there should be 62 characters in a line:
   (a) A single character, usually the ASCII character M
   (b) 60 characters representing an encoded string

(c) The endline

For the last line, some variation is permitted: The first character can be in the range 0x20 to 0x5F. There's a FURTHER CHECK: the very last line of the body does NOT contain data and is simply made up of a single backtick character.

In all cases:

(a) The first character represents the number of encoded characters, with 0x20 added! This is why all lines but the last should start with M: they contain 45 encoded characters (hex 0x3D). The ASCII representation of M is 5D, ie 0x3D + 0x20.

(b) Encoding of characters is done three-at-a-time. If there are less than three characters, we pad with hex zero (0x0). Encoding is as follows:

i. Divide the 3 bytes (3*8 = 24 bits) into four groups of 6 bits, working from left to right;

ii. This gives us four numbers between 0x0 and 0x3F;

iii. To each number, add 0x20, giving numbers in the range of 0x20–0x5F;

iv. Write the numbers as four ASCII characters to the output file.

In other words, we output characters in the set of:
`!"#$%&'()*,-./:;<=>?@[\]^_` as well as plus, space, 0–9 and A–Z.

3. The trailer line. This starts with the three character string 'end' (No quotes).

There are some frills:

- Also permissible are ASCII characters > 95 (5Fh) but only the rightmost 6 bits are relevant.
- The three digit number on the first line is the file mode (read/write/execute) The first number is the octal representation of the read permission of the file, the next the write permission, and finally the execute permission.
- Because some mailers used to strip off terminal blanks, it is usual (and perhaps wise) to pad such lines (with at least one terminal blank) with supplementary junk characters. The usual character is the backtick, 0x60, which has the added advantage that it translates to the otherwise illegal value 0x0 when the high bits are masked off.

11 Change log

11.1 Changes in version 2.0

The major changes in version 2.0 were related to the ability to shift sections down below other sections (defer writing of certain sections), waiting for all dependencies to be fulfilled before writing the code. Names are only resolved when all of the dependencies of that section have been met.

It would be possible to keep a record of 'names already resolved' but this is a little silly. We are only interested in deferring the writing of code, so there's little point in bookkeeping to this extent. Just leave out the dependency if it's already resolved!

If a child has no name, then the corresponding PENDINGNAME element is `''`. We could decrease memory requirements for CHILDREN by deleting array elements once written; we might benefit from even removing all corresponding elements entirely (as we will resolve the name immediately and the element in DEPENDENCIES is of course empty as well).

We also introduced the concepts of line concatenation, chomping off line feeds and subsequent leading spaces.[8]

Another amendment was allowing alteration of the initial comment string (newComment), the noWarn option to suppress irritating warnings (especially with multiple file writes), and OPTIONAL statements for a debugging version. (Note the bug with this described above).

We also introduced insertion of binary (uuencoded) files, by allowing the user to say `newTarget='uudecode'`. Usage of the Uudecode function is:

```
($filename, $mode, $outstring) = Uudecode($MAINW);
```

Several comments on the uudecoding option are (a) Should we check for and warn about executables? (b) At present permissions are disabled, so this isn't really an issue. (c) At some stage we should check on how robust the Perl unpack "u" option is!, and (d) the uuencoded line *cannot* start on the verbatim line but *MUST* start on the next line — there can be NO unused lines at the start.

11.2 Changes in version 2.1

1. We allow file start and end code. This is really for PHP, where such code is vital, but also for included HTML. The option startFile='<?php' (or what-

[8] When using the WinEdt text editor, the default is to trim trailing spaces, which can be rather irritating. You have to uncheck Options—Preferences—defaults—Trim spaces, or in already created documents uncheck Document—document settings—trim spaces.

ever) must be in the same line as the file name specification, but we can specify the endFile='?>' directive any time before we terminate the file for which we require such terminal code. The length of either can be specified as zero using startFile='' or endFile=''. **Note** that after each file is written, these text strings are reset to the null string, so the startFile and endFile values must be specified for *each file* in which they are used!

2. We remove any occurrence of `=cut` at the start of a verbatim comment, as we had problems with some Perl versions, notably v8.

3. We set the focus to the 'Wag' button, so on running the program with a file argument, things are ready to run.

4. We allow comment closure (as in HTML, older versions of C), along the lines of *endComment*='–>' for HTML. The default closing comment is an empty string. Note that unlike startFile and endFile, startComment/newComment persist until changed. *However*, endComment reverts to the empty string after the file using the current string has been closed!!

5. We fixed a problem with multiple dependencies, where a section was printed after just one dependency was resolved owing to an error in testing the remaining list of dependencies!

6. In minor version 2.1.1, if the user declines to overwrite a given file, we don't abort the whole Dogwagger process — instead, we write the data to the file 'JUNK.JUNK' and carry on! Note that we overwrite this file, and don't append, so if you decline to overwrite multiple files, only the last will be kept.

Appendix A: GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually

obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.) These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

(a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically

performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.) The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable. If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to

copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be

similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and ”any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

    NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO

OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

(END OF TERMS AND CONDITIONS)

Appendix B: A UUencoding program

Aww heck. Here's a UUencoding program we found on the 'Net at digitalriver.com:

```
begin 644 uuencode.com
MZV,-"DEN<'5T('!A=&@O9FEL93H@($EN<'5T(&9I;&4@97)R;W(N3W5T<'5T
M(&9I;&4@97)R;W(N8'T*96YD#0I.;R!A8W1I;VX@97AI<W1S(2'@06)O<G1I
M;F<A'''!'#@$''"T,,TA/')S#+JY!.E''>C$'>D]'>CB'7,QNOP#N;H'Z+,!
MN@(!N10'NP('M$#-(;]_',8%4(O7M'K-(>B['7,*M'&Z00&Y"0#KQ+KH'XOR
MN'']S2%S'^EP':-='8O/*\Y/L%S]\J[\='B+_H!]'3IU'D='B_>+UK^T'ZP*
MP'0#JNOXN'T*JU>+\K\X!(O7K'K'=0*P+JH\+G7TN'5UJ[AE'(D%,\F'/F0!
M_W0EM$[-(3P"=!T\$G0968O/*\J['@'#R[1'S2&Z2@&Y$P"P!>D]_[0\S2%9
M<P/IO'"C7P&ZJ@,KRNBA'.BV''1(L0:LBN#0Z-#HJJR*T-'HT>C1Z-'HJHKB
MK(K0T^BJBL*J@\4#.S9A'7('@#YC'0!U%X/]+74%Z#D'L08[-F$!<L&'/F,!
M'76S"^UT"2LV80$K[N@='+HY';D('.@^''(L>7P&T/LTAM$S-(5"T"<TA6.OT
MNJH#B\\KRE&+^HO%B'6T(+M@/XH%(L<"Q#K$=0**PZKB\5FX#0J)!4%!BQY?
M';1'S2%R!HOZ1S/MP[HG';D2'.LPNC@$N<BOBQY=';0_S2%R&HORB]H#V#O!
M='G'!P''Q@9C'0&)'F$!"\##NA8!N1$'Z','Z7S_4%)1NN($N0('Z!''65KH
M"P"ZX@2Y'@#H'@!8P[L"'+1'S2'#OH''O^@#_*P*P'0OM""L.L1V^SPO='0\
M+74<B]"+!#P_=!\D7SU/((O"=0OV%F0!1D:L.L1V!JJLZ_CYP\8%'/C#6+K\
M['!Z1'_D&)E9VEN(#8T-"!5545.0T]$12!V,BXP'$1A=FED(%'@2VER<V-H
M8F%U;2P@5&]A9"!(86QL+"!':79E;B!T;R!T:&4@<'5B;&EC(&1O;6%I;@!5
M545.0T]$12!;+6]=(%MD.EU;7'!A=&A<76)I;F%R>2YF:6P@/%)%5%523CX-
M"G!R;V1U8V5S(&)I;F%R>2Y5544@;VX@8W5R<F5N="!D<FEV95QP871H#0HH
M<')O=FED:6YG(&)I;F%R>2Y5544@9&]E<VXG="!A;')E861Y(&5X:7-T*2X-
M"BUO('-W:71C:"!F;W)C97,@;W9E<G=R:71E(&]F(&5X:7-T:6YG(&)I;F%R
M>2Y5544-"B14:&ES('!R;V=R86T@<F5Q=6ER97,@1$]3(%8R+C'@;W(@:&EG
':&5R+@T*)'''
'
end
```

Dogwagger will pull it out of the .TEX source, of course!