The Annals of the UK-TEX Users' Group
uk.tug.org
ISSN 1354-5930

Vol. 10.3 | Edited by Jonathan Webley | November 2010

Contents

UK-TUG Committee 2010

- Alun Moon (Chair)
- John Peters (Treasurer)
- Joseph Wright (Secretary)
- Jonathan Webley (Baskerville Editor)
- Simon Dales
- Jonathan Fine
- David Saunders

The committee can be contacted at:

uktug-committee@uk.tug.org

1 Editorial

Welcome to my third edition of *Baskerville*, published a good year after the second. I had hoped to be producing issues more frequently and more regularly than I have so far, unfortunately my time has been limited. However, this is a good issue with some interesting content – many thanks to those who have supplied articles for inclusion.

UK-TUG's AGM was held in October. As a result there are some changes to the committee, and in particular a new chair. Included in this issue are the valediction from the outgoing chair, Jonathan Fine, and some words from the new chair, Dr Alun Moon. Further details of the AGM can be found on our website.

Jonathan Webley
baskerville@uk.tug.org

2 Chair's Report

I have been chair of UK-TUG for four years now and am not standing for re-election. Although not a founding member of UK-TUG I joined it in its first year (1990), have served on the committee for several years, and in particular organised several effective and well-attended meetings and written many articles for our journal *Baskerville* during the 1990s.

In this report, in addition to recent news, I will take a longer view. In 1990 there were 12.4 million mobile (then called cellular) phones. Last year there were approximately 4.6 billion, 370 times as many. In 1989 the USA had its first commercial dial-up access Internet service provider, and in 1992 Congress allowed the National Science Foundation funded network to interconnect with commercial networks. In 1990 there were 313,000 Internet host computers. In 2009 there were 681 million, which is 2,175 times as many.

In 1990 and 1991 Tim Berners-Lee started the World Wide Web while at CERN. In 1993 there was Mosaic, the first widespread graphical web browser and approximately 600 websites. By 1996 there were about 100,000 of which half were dot-com. Also in 1996 two PhD students at Stanford (Larry Page and Sergey Brin) started the research project that became Google, which

last year made a profit of $6.5 billion from a revenue of $23.6 billion.

Facebook was launched in February 2004 and by July this year it had over 500 million active users and estimated revenues of $0.8 billion. It is estimated that there are 6.9 billion people alive now, so about 1/14 are on Facebook and about 2/3 have a mobile phone.

In short, over the past 20 years humanity has built an electronic communication network that reaches most of the globe and is used by perhaps a majority of the world's population. This system now embraces person-to-person communication (as in the telephone), broadcast communication (as in newspapers, radio and television) and also distribution of books, film and recorded music.

All this is not possible without agreed behaviour, without standards. The world's oldest international organisations are the Central Commission for Navigation on the Rhine (1816), the International Telecommunication Union (1869) and the Universal Postal Union (1874). For example, prior to the UPU a letter sent abroad often needed stamps of several countries on it.

In 1977, when Don Knuth started working on TEX, paper was by far the dominant medium for written communication. Books, letters, bills, newspapers, timetables, tickets, advertisements, diaries, logbooks are all examples. Punched cards and paper were used for textile looms (1725, Jacquard 1801), ticker tape (1870), 1890 US census (Hollerith) and player pianos (flourished 1896–1930). Hollerith was a founder of what became IBM. In 1977 paper was, in libraries, the dominant media for data storage, along with vinyl for music. Around then the Betamax and VHS video tape formats were introduced.

At that time academic, scientific, government and commercial data processing were major users of electronically stored written information. Now ordinary people are major users. In 1980 IBM produced the first gigabyte capacity hard drive, the size of a fridge, 550 lbs and $40,000. Today £50 will buy a 1 terabyte hard drive, and £5 a 2 gigabyte USB drive. The source file `tex.web` for TEX occupies about 1 megabyte.

Typesetting is an early example of this move from paper to digital media. Prior to the rise of phototypesetting (shining light through negative images of characters onto photographic film) in the 1970s, hot metal typesetting was often used to create a single original, which could be photographed and used to produce offset litho-plates. Phototypesetting was, in turn, replaced by digital typesetters (in important ways similar to modern laser printers) driven by a computer.

This was the situation when Don Knuth started working on TEX in 1977. There were computers and phototypesetters, and a large software gap. TEX and METAFONT admirably filled this gap, particularly for mathematical content. PostScript was developed by John Warnock and released by Abode in 1982. PDF followed in 1993. Digital typesetting is now taken for granted. We generate our PDF file, send it to a print supplier, who then returns thousands of printed copies.

Today many people prefer to receive written communication electronically, as text or chat message, email, webpage or PDF. Much typeset material is read on-screen and is seldom printed. And the webpage is a major medium for written communication. Last year Google bought a disused paper mill in Finland, for conversion into a data centre, at a total cost of $260 million. All for storage and transmission of digital information.

Although paper is far from dead, this enormous shift from paper to electronic media is of immense importance for the TEX community. Sadly, we are barely coping. Translation of LATEX to XML and vice versa is not straightforward. The problems of mathematical content on webpages have hardly been solved. Installation and running of TEX requires a long download and many technical skills. The LATEX3 project, started in 1993, is still far from completion.

Scalable Vector Graphics (SVG) provides us with a new opportunity. I describe it as PDF for webpages, with some elements of Flash. Although the W3C adopted SVG as a standard in 2001, it is not yet widely used due to non-adoption by Microsoft. But that is changing. All modern browsers support SVG, including Internet Explorer 9, but not IE7 and IE8.

SVG, together with web fonts, allows TEX quality typesetting to be displayed as a scalable part of a webpage on many modern browsers.

For IE7 and IE8 emulations are available. (For graphical material Google's svgweb translates SVG in the browser to Flash. For typeset matter HTML, CSS and web fonts provide a better emulation. The MathJax software gives an excellent example of what can be done now.)

There are many important challenges and opportunities facing us, besides SVG. Improved documentation and training, translation to and from XML, simplified installation, Unicode support are examples. But SVG is special for two reasons. First, it gives us an opportunity to establish TEX as the definitive means of rendering mathematics both for display on webpages and for print. Second, SVG will become the major medium for reading typeset material on webpages and elsewhere. For example, every EPUB reader must support SVG.

This, then, is my view of the past 20 years, which roughly speaking encompasses the life of UK-TUG, and of some of the challenges facing us now.

In the past four years we have made steady progress. When I became Chair things were so bad that there was open talk of dissolving the organisation. We instituted a subscription holiday and set up a projects fund to reduce our considerable surplus. Broadly speaking both have done well. We have shown ourselves able to spend money on supporting TEX in the UK and more widely. The AGM is being asked to end the subscription holiday.

We have funded two projects in 2007–8. We provided Jonathan Kew £1,600 for the TeXworks integrated development environment, which was completed and is now part of TEX distributions.

We also provided £1,700 (with a second equal instalment on receipt of a progress report) to add Unicode maths support to Latin Modern and the TeXGyre font collection (a project led by Hans Hagen). Here there seems to be no progress and no expenditure.

We have replaced our previous rather quirky constitution with something that will serve us better. We have held some fairly successful meetings, and earlier this year we organised LATEX training.

Particular thanks are due to David Saunders, who has been an excellent Treasurer, a steady and reliable voice, and who led greatly on the new constitution; to Joseph Wright, who has ably managed our website, handled membership and much administration, ran LATEX training with Nicola Talbot (who is is also thanked); and to Jonathan Webley who with much independent effort has restarted our magazine *Baskerville*.

I wish all new and continuing committee members and our new chair, Alun Moon, all the best for the coming years.

Jonathan Fine

3 New Chair

Let me introduce myself, I'm Alun and I've been a LATEX user since 1982 ('Eee when I were a lad we 'ad t' make do wit' teletype'). Don't worry this isn't an introduction for TEXies Anonymous. TEX and friends have been a useful tool through my time in higher education, though in my institution I'm a rarity. A colleague once likened the circle of TEX users to the 'escape committee'.

I've seen TEX grow with more and more packages; powerful graphics, presentations, PDF support and more. In this age of the wiki, just about every wiki system I've looked at has the capability to use LATEX as a back-end formatter for mathematics, many publishers support it for submissions. A Google search for material will show the range of organisations that use it. There are even apps for smart phones to write TEX. In this Internet age we could be looking at a golden age of LATEX.

There are three things I would like to encourage. Advocacy: we know LATEX is good, but we still have to convince a sometimes sceptical world. Training: we can continue with the already excellent training and support material that we have produced. Usability: can we make TEX usable? My kids have been exposed to WYSIWYG words from day one at school, what can we do to make LATEX usable for them?

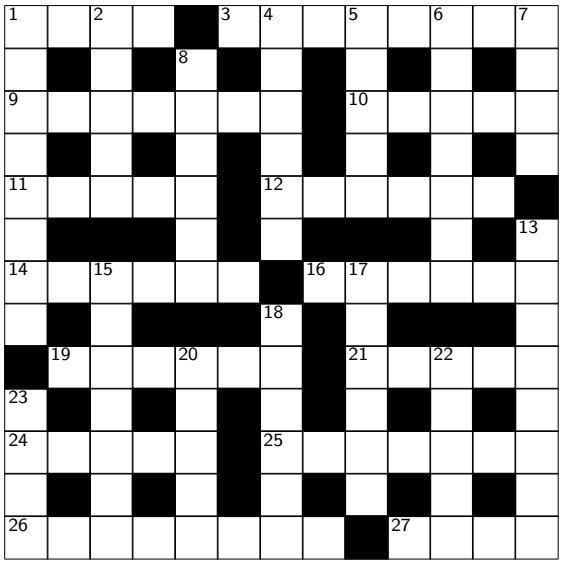
Dr Alun Moon

* * * * *

There are 10 kinds of people – those who know binary and those who don't.

4 The Hound

Jonathan Webley



Across

1 Parasites found in the police. (4)
3 Can be heard, but nothing caustic, unluckily. (8)
9 Can smut, nasty, be found in this holy place? (7)
10 Flower cluster for bumble without bee. (5)
11 You nine, rouse yourselves, else suffer from boredom. (5)
12 I decry fizzy gas. (6)
14 Reveal and I am sunk hopelessly. (6)
16 Exactness is not correct for our rig. (6)
19 Puzzle found in game, unfortunately. (6)
21 In colossal volley see gunfire. (5)
24 Regarding dirty mud, I object. (5)
25 I cut rice, endlessly cooked, on my lap. (7)
26 These things are abused after illegal vent loss. (8)
27 In regency stable find a pouch. (4)

Down

1 Hear this, when he puts line over. (6,2)
2 Howitzer, not north, but one found in church. (5)
4 My code can be broken for a laugh. (6)
5 Unknown to us, you are lending money. (5)
6 Hundreds on leaky boat, see nothing but this shag. (8)
7 On my leg! The little cow! (4)
8 Studies what's right and wrong, because it itches so.
13 Threadbare coats try, but it still makes you cold. (8)
15 City's ugly minim has the least. (7)
17 Terrible rent is put in between. (6)
18 Overfed jazz fan earns too much. (3,3)
20 Spiky bush gores badly. (5)
22 Infested and useless. (5)
23 Without ends, just a miscellany of stuff. (4)

5 Plea for an Online TUG Image Service

Steve Mayer

In the past few years there has been an explosion in the use of LaTeX online; this isn't for use in documents nor for presentations but for small images in forums, blogs and wikis. This allows users to write about mathematics without having to have a TeX distro on their own computers or get involved in arcane setup procedures as the LaTeX code is compiled on a server. This is a popular and effective system but not available to many who want to use their own hosts or a non-configurable system like Google's Blogger [1]. mimeTeX [2] was developed by John Forkosh to allow users to emulate a LaTeX system but it may require compiling on the server and assumes CGI programs can be run there, which is not always possible.

One solution to this is to allow the compilation to take place on an external server. The

LaTeX code is sent to the providing server which returns an image that can then be inserted into the blog or forum etc. For example,

```
<img src="http://www.forkosh.dreamhost
                    .com/mathtex.cgi?x^2">
```

would give a GIF image of x^2 in an HTML context. This service is provided by at least three providers[1]: mathTeX [3], CodeCogs [4] and WordPress.org [5]. WordPress's service is meant for use in its own blogs, but it can be accessed from anywhere, provided the user encodes the text (so, for example ^ is replaced by %5E).

$$\mathrm{M}_{\mathrm{t}}^{\mathrm{a}}(\mathrm{h})/\mathrm{T}_{\mathrm{a}}^{\mathrm{r}}(\mathrm{n})$$

If you'd like to see how the system works with these three services I have written a simple webpage which shows the image code and URL as well as the image. It's at `http:/sixthform.info/tug/onlinelatex.html`. Since I don't intend this as a universal service, you will need to log in using:

```
username:  uktug
password:  baskerville
```

Although you can run it immediately from the website please feel free to download the page and play with the JavaScript code.

The advantages of the services are that nothing needs to be installed by the user; the code is accepted by virtually any HTML editor and the result is high-quality LaTeX. But the big disadvantage is that the user is relying on an external service and there is absolutely no guarantee that one day they will cease to offer that service. They are within their rights to withdraw their service completely or restrict it to certain uses or may wish to start charging. They could add advertisements or deliberately slow down the compilation.

John Forkosh, the author and provider of the mimeTeX and mathTeX programs and services, has suggested a solution to the latter problem. He says that if TUG were to provide the service then users would regard it as a trusted system and know that it would be reliable, consistent and long-lasting in the same way as they trust TUG to provide a service such as Tex Live.

I would support this idea. Online use of mathematics has increased dramatically since such rendering systems started being used about seven years ago and has raised the profile of LaTeX quite considerably. Many students meet LaTeX for the first time in forums: usage there persuades some of them to go on to learn more and use it for typesetting, which I'm sure we would all want to encourage.

Such a service does not require large facilities or expensive servers. Once set up it requires very little maintenance. So how about it TUG? Can we look forward to such a service in the near future?

References

[1] Blogger, `http://www.blogger.com`

[2] mimeTeX, `http://www.forkosh.dreamhost.com/source_mimetex.html`

[3] mathTeX, `http://www.forkosh.dreamhost.com/source_mathtex.html`

[4] CodeCogs, `http://www.codecogs.com/latex/eqneditor.php?lang=en-en`

[5] WordPress, `http://en.support.wordpress.com/latex/`

[6] MathTran, `http://www.mathtran.org`

6 LaTeX word count

Steve Mayer

I often hear colleagues in faculties of arts and humanities discuss the *word count* of essays from students and how appalled they are if the students are under or over the limit by some margin or other. I also remember being asked, as a student, to write a 10,000 word essay but not having

[1] mimeTeX also provides such a service but the output using mathTeX is superior; MathTran [6] offers a similar service for plain TeX users.

a clue as to what that might look like. I could understand 'pages' but 'words' seems to be too small a unit of measurement.

Fortunately, as a mathematician, I haven't had to worry about word counts (just as well as my PhD thesis and papers were bashed out on a sit-up-and-beg typewriter – old even then – before the days of TEX or word processors) so it seems a rather arcane subject to me (it will become real if the editor starts complaining about the length of this article). Thus I am intrigued by the LATEX word count site `http://folk.uio.no/einarro/Services/texcount.html/`. This uses the TeXcount Perl script [1] to count the words in LATEX documents. It's a very nice program which displays:

Words in text
Words in headers
Words in float captions
Number of headers
Number of floats
Number of math inlines
Number of math displayed

And subcounts: `text+headers+captions` (`#headers/#floats/#inlines/#displayed`).

Submitting part of this article gave six float captions and zero floats – not sure how it came to that conclusion.

Of course, defining the number of words in a mathematical expression is difficult (impossible?) and the program just skirts round this by excluding any inline or displayed formulae in the word count. Using text mode for x=1 and x^2=1 gives two and three words respectively, which just shows how using word count for this is hopeless.

My question is: is this site useful in any meaningful context? Do universities, publishers and others insist on word counts for highly mathematical contributions and are they happy that pages of long mathematical expressions may not be included in that word count?

[*LaTeX Word count* claimed there were 495 words in a draft of this article. Will they now have a copy on their server and what will they do with it?]

References

[1] TeXcount Perl Script, `http://tug.ctan.org/cgi-bin/ctanPackageInformation.py?id=texcount`

7 Detexify

Steve Mayer

A common question, particularly from students, is 'How do I put such and such a symbol in my LATEX document?' to which the answer is often 'Trawl through The Comprehensive LATEX Symbol List' [1]. For some students who haven't a clue what the symbol is called, this is asking too much (or so they think). You can instead refer them to Detexify[2] – a LATEX symbol classifier [2]. This site allows you to draw the symbol using a mouse and then offers the symbols it thinks you might want.

CTAN

the Comprehensive Tex Archive Network

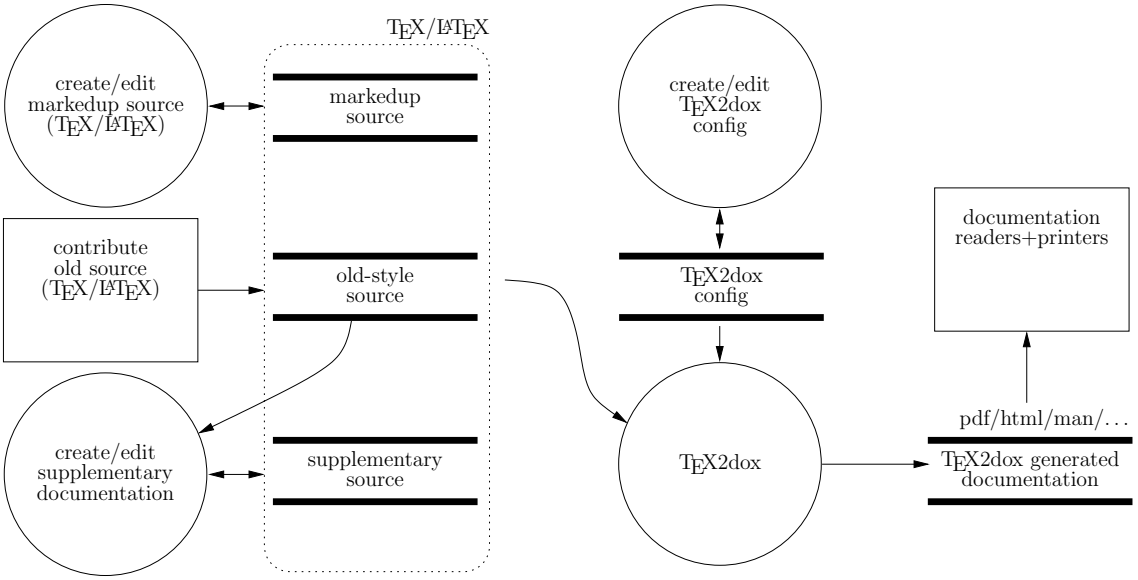
Unfortunately, it may not work in some versions of Internet Explorer as there's no box to draw in, but there's no problem in Firefox. I don't know about you, but my drawing skills would score zero in any contest, so the results for me, are to say the least, hilarious.

The site hopes to learn from users' input but is it just a curio to be played with and then forgotten? Can it prove useful and what does it say for optical character recognition (OCR) in mathematics?

References

[1] The Comprehensive LATEX Symbol List, `http://tug.ctan.org/cgi-bin/ctanPackageInformation.py?id=comprehensive`

[2] Detexify[2] – LaTeX symbol classifier, `http://detexify.kirelabs.org/classify.html`



8 TEX2dox

Simon Dales

TEX2dox is an auto-documentation tool for TEX. We want to document TEX sources in an efficient way, so we let the computer do the work.

8.1 Introduction

Documenting TEX and LATEX sources is always difficult. We are programmers trying to make things work.

We can use literate programming techniques but they can be antagonistic to other programming practices, and mostly don't get done.

We may want to document our own code with a minimum of work. We may have undocumented code that we wish to retrospectively document.

TEX2dox attempts to address these issues.

8.1.1 Literate Programming

If one has the foresight and desire one can use literate programming to create one's packages. We write a document that is a mixture of documentation and raw code. This is fed to some program that splits the document to make final documentation and runtime code.

The official LATEX tool is `DocStrip` [1]. It is a widely used system, but it forces one to originate one's TEX in a particular way, however desirable that might be.

Because all editing has to be done on the combined document it cannot be applied retrospectively. But it does produce very high quality output.

8.1.2 Auto-documentation tools

There are auto-documentation tools. Unlike with literate programming they can be applied to finished source code to extract some documentation. They can even be applied to unprepared source code.

They parse the source looking for interesting strings and build documentation from what they find. These tools typically look for function declarations possibly supported by 'magic comments' (which contain special markup that distinguishes them from ordinary ones).

The common tools for the C-like languages are `Doxygen` [2] and LuaDoc [3] for Lua [4].

Having gathered the data it can then output it on some appropriate format. In the case of `Doxygen` it outputs HTML and LATEX. Also MAN, RTF and XML (but I have never used these myself).

The documentation typically contains descriptions of each function and where it is found

and referenced. This lets one retrospectively examine old code. The output is generally oriented to the API. If we can apply filters to these function names we can separate the internal implementation functions from the public API.

In practise, one can be editing one's source and testing it with little direct thought to the documentation. Every so often you run your auto-documentation tool to keep the HTML up to date.

LuaDoc: This is an equivalent tool to `Doxygen` for Lua. It has very similar input and output. The exact HTML style may be different but the content is similar.

TEX2dox: Why don't we use an auto-documentation tool for TEX? We would be able to document our sources as we develop them and retrospectively. The documentation may not be as eloquent as if we had used literate programming but it is better than not doing it at all.

8.2 Doxygen

`Doxygen` is the de facto tool in the C-family world.

As you program you insert 'magic comments' into your code. Even if you don't bother with this, the tool can find where you declared each function. It can warn you of undocumented functions so you can go back and add some descriptive text.

'Magic comments' are comments that are real comments but are specially formatted internally to be identifiable to an application. For example the `#!` at the beginning of UNIX executable script files. In the case of `Doxygen` we use `//!`, amongst others. Any ordinary comments will be ignored by the application.

Documentation can be applied retrospectively. It will find the function declarations but will complain a lot about missing documentation. Even if the aged source must remain read-only then you can still add documentation using external files. The resulting documentation is the same as if it had been inserted in the natural place.

This C fragment:

```
87 // this is a boring comment
88 //
89 //! \brief makes things bold
90 //!
91 //! Takes an input string and
92 //! outputs it bolded.
93 //! I suppose you could say more.
94 TErr Bold(TeX & Text){
95   return Text.bold();
96   }
```

produces this documentation output:

TErr Bold(TeX & Text) – makes things bold
Takes an input string and outputs it bolded. I suppose you could say more.

Defined on line 94 of wibble.cpp

8.2.1 Pas2Dox

Somebody else has got there first: by making `Doxygen` understand Pascal. `Pas2Dox` [5] is a filter that reads Pascal source and outputs a minimal pseudo-C equivalent. Note that `Doxygen` does not need to be a full C parser, it need only read it well enough to extract the interesting parts. So `Pas2Dox`'s output need only be a minimal C.

8.2.2 TEX2dox

Inspired by `Pas2Dox` I decided to explore a similar system for letting `Doxygen` read TEX.

The idea is to write your own TEX source with some magic comments and let the tool make the API documentation.

This TEX input:

```
87 %  this is a boring comment
88 %
89 %%! \brief makes things bold
90 %%!
91 %%! Takes an input string and
92 %%! outputs it bolded.
93 %%! I suppose you could say more.
94 \def\Bold#1{%
95   {\bfseries #1}%
96   }
```

produces this output:

> ***\Bold{}*** – makes things bold
> Takes an input string and outputs it bolded. I suppose you could say more.
>
> Defined on line 94 of wibble.sty

8.3 TEX2dox Mk. 1

TEX2dox is a tool to auto-document TEX and LATEX sources. Mk. 1 is an experiment to see if it could be done.

For historic reasons I decided to write it in command line PHP [6]. I know this language so I could get something to work. I was writing my own package at the time and so used it to help me with that project and refine the TEX2dox application.

You may have spotted that TEX looks nothing like C, and this is the chief challenge.

8.3.1 Name Mangling

C's function names are of the form: <letterU> [<letterU>|<digit>]*, where <letterU> ::= <letter> | <underscore>, for example: `wibble`, `wibble123` or `_wibble1a`. TEX's equivalent are mostly a string of just letters. This may make it look like a simpler problem, but it's not.

Within packages `csname`s typically contain some other characters, especially @ signs. This would cause `Doxygen` indigestion, so the `csname`s[2] have to be heavily mangled to make them C-compatible. So `@a@b` becomes `__AT_a_AT_b`; readable by humans but only just.

Also C functions take parameters such as `int main(int argc, char**argv)`. Similarly for a TEX macro, e.g. `\def\Bold#1`, we might want to mangle it to `Bold(_hash_1)`. This is a legal C function declaration, so `Doxygen` can process it.

8.3.2 Running

We run TEX2dox twice: once to extract the documentation and build a database and appropriate intermediate files, and secondly to run `Doxygen` to do the gathering and outputting.

It parses the TEX sources and builds a 'Grand Unified Comment File', which eventually gets read by `Doxygen`.

The GUCF is partly built from the magic comments we put in our source and partly from some extra external ones. This enables us to, say, document the whole of `texmf` and `CTAN`. It says clearly in those files that they may not be modified without attribution and renaming, so how would we document them and retain functionality?

The solution is to build a separate directory tree of external documentation files. These need not map the structure of `texmf`, just that it refers to the `csname`s within those files.

8.3.3 TEX2dox internals

As `Doxygen` runs it walks through the source directories picking files. If a file is in the set of extensions for filtering then that filter is applied, in this case TEX2dox. It reads the TEX source and outputs some equivalent pseudo-C. This only has to be good enough to contain the tokens that `Doxygen` would use when constructing its internal database, from which it then outputs the documentation.

So, the sample code:

[2] In this context, they are the equivalent to function names in C.

```
%%%%%%%%%%%
% some sample code
%
%%! \brief bolds text
\def\Bold#1{%
  {\bfseries #1}%
  }
%%! \brief italicises text
\def\Italic#1{%
  {\itshape #1}%
  }
%
```

produces:

```

Bold(_hash_1){}


Italic(_hash_1){}
```

Note that the line numbers of the pseudo-C match the `\def` positions. This enables the source browser within the `Doxygen` output to point to the correct place in the source.

Note also that all the comments are missing. This is intentional, because TEX comments take up one line at least. What would we do if we found more than one `\def` on the same line, say `\def\a{...}\def\b{...}`? So we output `a(){}b(){}` and then deal with the documentation comments separately.

8.3.4 Implementation

I wrote it in command line PHP. Some have said I should have done it in Python [7], but I already knew PHP and so I decided that's that where I would start. Should TEX2dox mature I can address the issue of language-porting at a later stage.

It uses a TEX tokeniser and a hand-crafted parser. The tokeniser is a tweaked version of the one mentioned in the *TEXBook* but with enhancements and simplifications. Its `chardef/catcodes` are fixed and it must regard magic comments as tokens too.

Since most LATEX code uses the conventional `chardef/catcodes` this is not an issue for most of `texmf`. Where things get more confused is in the very low-level parts of the setup of plain TEX and LATEX format files. However, very few documenters will want to read these, so in the main this is not an issue.

In general the parser gets tokens and looks for interesting sequences of tokens. If it finds something like `<def><csname>` it will try to construct a `csname` object. It will also have gathered any magic comments to associate with that object.

When it can be sure it has enough tokens it can output them. When running in `Doxygen` filter mode it outputs pseudo-C function declarations. When running in gather mode it will take these objects and output them to the Grand Unified Comment File.

A little while later `Doxygen` will read this GUCF and incorporate its contents within its internal tables.

8.3.5 Output

`Doxygen` outputs some quite pretty HTML and LATEX.

Typically it will document all the 'functions' within the code and provide a link to the source. This can be internal to the document, so in the case of HTML the source will be included as a webpage form of the actual source.

For development I find the HTML output the most useful. Keep a webpage open to your documentation. Just run `Doxygen` and TEX2dox each time you want an update and it will regenerate the HTML. Then browse in the usual way.

If you want some 'hard-copy' style output then you can run `pdflatex` on the outputted LATEX to get a finished PDF.

It does output other formats, including MAN, XML and RTF. Also Microsoft compressed HTML can be produced with filters. However, I haven't tested these myself.

Internal and end-user API: There is a trick we can use on `Doxygen` to filter the internals of a package from its public API. In a LATEX package we tend to make internal macros have @ signs within them. So TEX2dox declares these mangled `csnames` as `static` (in C terms), and

Doxygen can filter them out if we want. Therefore we can run TEX2dox with `static=on` and we see all our internal API documented. This would confuse end users, and so we can run it again with `static=off`, and just see the public API to our package.

8.3.6 Implementation issues

The main issue is that to keep Doxygen happy we have to do a lot of very ugly name-mangling. Whilst the mangled names are technically correct they are very unergonomic to read.

The GUCF is written out as a C source file, composed entirely of comments. This mechanically generated file is visible to the end users in the documentation. Ideally nobody should see it.

8.4 Conclusion

Mk. 1 works. It outputs usable-ish documentation. It fits in with a chaotic real-world programming workflow.

Mk. 2 has yet to be written and is very much 'vapourware'. This is intentionally so until we have a better specification.

It is critical that the magic comment format is decided beforehand. Implementations of TEX2dox can be improved over time but we don't want to have deal with backwards incompatibility. Hence, we want to fix this now.

8.4.1 Magic comments

For Mk. 1, I chose to use the Doxygen style of magic comments because I knew them already. Whether this is the right comment style for the long term, I don't know.

Should we use our own new proprietary one or tweak the Doxygen/Luadoc style? A number of us out there already know Doxygen, especially those of us that have developed applications, which will mostly be in the C-like family (C, C++, Java, PHP).

Luadoc's magic comments are very similar to those used by Doxygen, so maybe we should use this style because of the forthcoming luatex implementation of TEX.

Or should we go for an incompatible style that fits the needs of TEX better?

References

[1] DocStrip, http://www.tug.org/texmf-dist/doc/latex/base/docstrip.pdf

[2] Doxygen, http://www.doxygen.org

[3] LuaDoc – Documentation Generator Tool for the Lua language, http://luadoc.luaforge.net/

[4] Lua, http://www.lua.org/

[5] Pas2Dox, http://pas2dox.sourceforge.net/

[6] PHP, http://www.php.net/

[7] Python Programming Language, http://www.python.org/

9 The Hound Answers

Across

1. Iice, 3. Acoustic, 9. sanctum, 10. umbel, 11. ennui, 12. dry. ice, 14. unmask, 21. salvo, 24. demur, 25. circuit, 16. rigour, 19. enigma, 26. solvents, 27. cyst.

Down

1. listen up, 2. canon, 4. comedy, 5. usury, 6. tobacco, 7. calf, 8. ethics, 13. cryostat, 15. minimal, 17. insert, 18. fat cat, 20. gorse, 22. lousy, 23. odds.

10 Contributions

All contributions to *Baskerville* should be sent to the editor at:

baskerville@uk.tug.org

Articles on any area of TEX or its friends, UK-TUG or related topics are very welcome. The Committee is particularly keen to publish articles with a UK *flavour*. Send in your comments on this issue; your suggestions, letters, thoughts, tips and hints, articles, jokes, questions, requests for help, jobs, cartoons or puzzles – anything relevant will be considered for publication.