Tip 67: Document your code with Doxygen

Ben Klemens

13 February 2012

level: Your list of functions is getting longer
purpose: make it harder to be lazy

You need documentation. You know this, and you know that you need to keep it current when the code changes. Yet, somehow, documentation is so often the first thing to fall by the wayside. It is so very easy to say *it runs; I'll document it later.*

So you need to make writing the documentation as easy as physically possible. The immediate implication is that you have the documentation for the code in the same file as the code, as close as possible to the code being documented. The immediate implication of this is that you're going to need a means of extracting the documentation from the code file.

I'll present two means of doing this: Doxygen[1] and CWEB[2].

Doxygen is a simpler system with simpler goals. It works best for attaching a description to each function, struct, or other such block. This is the case of documenting an interface for users who will never care to look at the code itself. The description will be in a comment block right on top of the function, struct, &c., so it is easy to write the documentation comment, then write the function to live up to the promises you just made.

CWEB is based on Donald Knuth's WEB program, which implemented what Knuth calls *literate programming*. The intent here is to take that *write the documentation first* advice to the extreme, and allow you to write a free-form description of your code, with code interspersed here and there to formalize the description for the computer. I'll show you some CWEB next time.

Doxygen The syntax for Doxygen is simple enough, and a few bullet points will have you well on your way to using it:

- If a comment block starts with two stars, `/** like so */`, then Doxygen will parse the comment. One-star comments, `/* like so */` are ignored.
- If you want Doxygen to parse a file, you will need a `/** \file */` comment at the head of the file; see example below. If you forget this, Doxygen won't

[1]`http://www.doxygen.info`
[2]`http://www-cs-staff.stanford.edu/~uno/cweb.html`

produce output for your file and won't give you much of a hint as to what went wrong.

- Put the comment right before the function, struct, et cetera.
- Your function descriptions can (and should) include `\param` segments describing the input parameters and a `\return` line listing the expected return value. Again, see below.
- You can use an `@` anywhere I used a backslash above: `@file`, `@mainpage`, et cetera. This is in emulation of JavaDoc, which seems to be emulating WEB. As a LaTeX user, I am more used to the backslash.

To run it, you will need a configuration file, and there are a lot of options to configure. Doxygen has a clever trick for handling this: run

```
doxygen -g
```

and it will write a configuration file for you. You can then open it and edit as needed; it is of course very well documented. After that, run `doxygen` by itself to generate the outputs, including HTML, PDF, XML, or manual pages, as per your specification.

The narrative Your documentation should contain at least two parts: the technical documentation describing the function-by-function details, and a narrative explaining to users what the package is about and how to get their bearings.

Start with a comment block with the header `\mainpage`. This can be anywhere in your code. You could put it close to the code itself, or the narrative may make sense as an separate file consisting entirely of Doxygen comment blocks, maybe named `documentation.h`. If you are producing HTML output, this will be the `index.html` of your web site—the first page readers should see. From there, add as many pages as you'd like. Subsequent pages have a header of the form

```
/** \page onewordtag The title of your page
  */
```

Back on the main page (or any other, including function documentation), add `\ref onewordtag` to produce a link to the page you wrote. You can tag and name the mainpage as well, if need be.

An example Here goes. This program gets a lot done, and the documentation via Doxygen looks good. Read the documentation to see what it does and how to run it.

```
/** \file */
#include <stdio.h>
#include <curl/curl.h>
#include <libxml2/libxml/xpath.h>
```

```
/** \mainpage
  The front page of the Grey Lady's web site is as gaudy as can be, including se
  headlines and sections trying to get your attention, various formatting scheme
  and even photographs--in <em>color</em>.

 This program reads in the NYT headlines RSS feed, and writes a simple list in p
 HTML. You can then click through to the headline that modestly grabs your atten

 For notes on compilation, see the \ref compilation page.
*/

/** \page compilation Compiling the program

Save the following code to \c makefile.

Notice that cURL has a program, \c curl-config, that behaves like \c pkg-config,
but is cURL-specific. I do not know why they did this.

\code
CFLAGS =-g -Wall -std=gnu99 -O3  `curl-config --cflags`
LDLIBS=`curl-config --libs ` -lxml2
\endcode

Having saved your makefile, if you name this here code file to \c headlines.c, t
use <tt>make headlines</tt> to compile.

Of course, you have to have the development packages for libcurl and libxml2 ins
for this to work.
*/

//These have in-line Doxygen documentation. The < points to the prior text
//being documented.
char *rss_url = "http://feeds.nytimes.com/nyt/rss/HomePage"; /**< The URL for an
char *rssfile = "nytimes_feeds.rss";        /**< A local file to write the RSS t
char *outfile = "now.html";                 /**< The output file to open in your

/** Check an assertion; print a message to stderr and exit if the assertion fail
#define stopifnot(assertion, ...) if (!(assertion)){fprintf(stderr, __VA_ARGS__)

/** Print a list of headlines to the outfile, which is overwritten without apolo

  \param urls The list of urls. This should have been tested for non-NULLness
  \param titles The list of titles, also pre-tested to be non-NULL. If the lengt
      the \c urls list or the \c titles list is \c NULL, this will crash.
*/
void print_to_html(xmlXPathObjectPtr urls, xmlXPathObjectPtr titles){
    FILE *f = fopen(outfile, "w");
    for (int i=0; i< titles->nodesetval->nodeNr; i++)
        fprintf(f, "<a href=\"%s\">%s</a><br>\n"
                , xmlNodeGetContent(urls->nodesetval->nodeTab[i])
                , xmlNodeGetContent(titles->nodesetval->nodeTab[i]));
    fclose(f);
}

/** Parse an RSS feed on the hard drive. This will parse the XML, then find all
  matching the XPath for the title elements and all nodes matching the XPath for
  Then, it will write those to the outfile.

  \param infile The RSS file in.
*/
void parse(char *infile){
    xmlDocPtr doc = xmlParseFile(infile);
    stopifnot(doc, "Error: unable to parse file \"%s\"\n", infile);

    xmlXPathContextPtr context = xmlXPathNewContext(doc);
    stopifnot(context, "Error: unable to create new XPath context\n");

const xmlChar *titlepath= (xmlChar*)"//item/title";
const xmlChar *linkpath= (xmlChar*)"//item/link";
    xmlXPathObjectPtr titles = xmlXPathEvalExpression(titlepath, context);
    xmlXPathObjectPtr urls = xmlXPathEvalExpression(linkpath, context);
    stopifnot(titles && urls, "either the Xpath '//item/title' or '//item/link'

    print_to_html(urls, titles);

    xmlXPathFreeObject(titles);
    xmlXPathFreeObject(urls);
    xmlXPathFreeContext(context);
    xmlFreeDoc(doc);
}

/** Use cURL's easy interface to download the current RSS feed.

\param url The URL of the NY Times RSS feed. It may be that any of the ones list
           \url http://www.nytimes.com/services/xml/rss/nyt/ will work.
\param outfile The headline file to write to your hard drive. I'll first save th
        RSS feed to this location, then overwrite it with the short list of link

 \return 1==OK, 0==failure.
 */
int get_rss(char *url, char *outfile){
    FILE *feedfile = fopen(outfile, "w");
    if (!feedfile) return -1;

    CURL *curl = curl_easy_init();
    if(!curl) return -1;
    curl_easy_setopt(curl, CURLOPT_URL, url);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, feedfile);
    CURLcode res = curl_easy_perform(curl);
    if (!res) return -1;

    curl_easy_cleanup(curl);
    fclose(feedfile);
    return 0;
}

int main(void) {
    stopifnot(get_rss(rss_url, outfile), "failed to download %s to %s.\n", rss_u
    parse(rssfile);
}
```