kernel-doc nano-HOWTO

How to format kernel-doc comments

In order to provide embedded, 'C' friendly, easy to maintain,
but consistent and extractable documentation of the functions and
data structures in the Linux kernel, the Linux kernel has adopted
a consistent style for documenting functions and their parameters,
and structures and their members.

The format for this documentation is called the kernel-doc format.
It is documented in this Documentation/kernel-doc-nano-HOWTO.txt file.

This style embeds the documentation within the source files, using
a few simple conventions. The scripts/kernel-doc perl script, some
SGML templates in Documentation/DocBook, and other tools understand
these conventions, and are used to extract this embedded documentation
into various documents.

In order to provide good documentation of kernel functions and data
structures, please use the following conventions to format your
kernel-doc comments in Linux kernel source.

We definitely need kernel-doc formatted documentation for functions
that are exported to loadable modules using EXPORT_SYMBOL.

We also look to provide kernel-doc formatted documentation for
functions externally visible to other kernel files (not marked
"static").

We also recommend providing kernel-doc formatted documentation
for private (file "static") routines, for consistency of kernel
source code layout. But this is lower priority and at the
discretion of the MAINTAINER of that kernel source file.

Data structures visible in kernel include files should also be
documented using kernel-doc formatted comments.

The opening comment mark "/**" is reserved for kernel-doc comments.
Only comments so marked will be considered by the kernel-doc scripts,
and any comment so marked must be in kernel-doc format. Do not use
"/**" to be begin a comment block unless the comment block contains
kernel-doc formatted comments. The closing comment marker for
kernel-doc comments can be either "*/" or "**/", but "*/" is
preferred in the Linux kernel tree.

Kernel-doc comments should be placed just before the function
or data structure being described.

Example kernel-doc function comment:

```
/**
 * foobar() - short function description of foobar
 * @arg1:	Describe the first argument to foobar.
 * @arg2:       Describe the second argument to foobar.
 *              One can provide multiple line descriptions
 *              for arguments.
 *
 * A longer description, with more discussion of the function foobar()
 * that might be useful to those using or modifying it.  Begins with
 * empty comment line, and may include additional embedded empty
 * comment lines.
 *
 * The longer description can have multiple paragraphs.
 */
```

The short description following the subject can span multiple lines
and ends with an @argument description, an empty line or the end of
the comment block.

The @argument descriptions must begin on the very next line following
this opening short function description line, with no intervening
empty comment lines.

If a function parameter is "..." (varargs), it should be listed in
kernel-doc notation as:

```
 * @...: description
```

Example kernel-doc data structure comment.

```
/**
 * struct blah - the basic blah structure
 * @mem1:       describe the first member of struct blah
 * @mem2:       describe the second member of struct blah,
 *              perhaps with more lines and words.
 *
 * Longer description of this structure.
 */
```

The kernel-doc function comments describe each parameter to the
function, in order, with the @name lines.

The kernel-doc data structure comments describe each structure member
in the data structure, with the @name lines.

The longer description formatting is "reflowed", losing your line
breaks. So presenting carefully formatted lists within these
descriptions won't work so well; derived documentation will lose
the formatting.

See the section below "How to add extractable documentation to your
source files" for more details and notes on how to format kernel-doc
comments.

Components of the kernel-doc system

Many places in the source tree have extractable documentation in the
form of block comments above functions. The components of this system

are:

- scripts/kernel-doc

  This is a perl script that hunts for the block comments and can mark them up directly into DocBook, man, text, and HTML. (No, not texinfo.)

- Documentation/DocBook/*.tmpl

  These are SGML template files, which are normal SGML files with special place-holders for where the extracted documentation should go.

- scripts/basic/docproc.c

  This is a program for converting SGML template files into SGML files. When a file is referenced it is searched for symbols exported (EXPORT_SYMBOL), to be able to distinguish between internal and external functions.
  It invokes kernel-doc, giving it the list of functions that are to be documented.
  Additionally it is used to scan the SGML template files to locate all the files referenced herein. This is used to generate dependency information as used by make.

- Makefile

  The targets 'sgmldocs', 'psdocs', 'pdfdocs', and 'htmldocs' are used to build DocBook files, PostScript files, PDF files, and html files in Documentation/DocBook.

- Documentation/DocBook/Makefile

  This is where C files are associated with SGML templates.

How to extract the documentation

If you just want to read the ready-made books on the various subsystems (see Documentation/DocBook/*.tmpl), just type 'make psdocs', or 'make pdfdocs', or 'make htmldocs', depending on your preference. If you would rather read a different format, you can type 'make sgmldocs' and then use DocBook tools to convert Documentation/DocBook/*.sgml to a format of your choice (for example, 'db2html ...' if 'make htmldocs' was not defined).

If you want to see man pages instead, you can do this:

```
$ cd linux
$ scripts/kernel-doc -man $(find -name '*.c') | split-man.pl /tmp/man
$ scripts/kernel-doc -man $(find -name '*.h') | split-man.pl /tmp/man
```

Here is split-man.pl:

```
-->
#!/usr/bin/perl

if ($#ARGV < 0) {
   die "where do I put the results?\n";
}

mkdir $ARGV[0],0777;
$state = 0;
while (<STDIN>) {
    if (/^\.TH \"[^\"]*\" 9 \"([^\"]*)\"/) {
        if ($state == 1) { close OUT }
        $state = 1;
        $fn = "$ARGV[0]/$1.9";
        print STDERR "Creating $fn\n";
        open OUT, ">$fn" or die "can't open $fn: $!\n";
        print OUT $_;
    } elsif ($state != 0) {
        print OUT $_;
    }
}

close OUT;
<--
```

If you just want to view the documentation for one function in one
file, you can do this:

```
$ scripts/kernel-doc -man -function fn file | nroff -man | less
```

or this:

```
$ scripts/kernel-doc -text -function fn file
```

How to add extractable documentation to your source files

The format of the block comment is like this:

```
/**
 * function_name(:)? (- short description)?
(* @parameterx(space)*: (description of parameter x)?)*
(* a blank line)?
 * (Description:)? (Description of function)?
 * (section header: (section description)? )*
(*)?*/
```

All "description" text can span multiple lines, although the
function_name & its short description are traditionally on a single line.
Description text may also contain blank lines (i.e., lines that contain
only a "*").

"section header:" names must be unique per function (or struct,
union, typedef, enum).

Avoid putting a spurious blank line after the function name, or else the
description will be repeated!

All descriptive text is further processed, scanning for the following special
patterns, which are highlighted appropriately.

```
'funcname()' - function
'$ENVVAR' - environment variable
'&struct_name' - name of a structure (up to two words including 'struct')
'@parameter' - name of a parameter
'%CONST' - name of a constant.
```

NOTE 1: The multi-line descriptive text you provide does *not* recognize
line breaks, so if you try to format some text nicely, as in:

```
  Return codes
    0 - cool
    1 - invalid arg
    2 - out of memory
```

this will all run together and produce:

```
  Return codes 0 - cool 1 - invalid arg 2 - out of memory
```

NOTE 2: If the descriptive text you provide has lines that begin with
some phrase followed by a colon, each of those phrases will be taken as
a new section heading, which means you should similarly try to avoid text
like:

```
  Return codes:
    0: cool
    1: invalid arg
    2: out of memory
```

every line of which would start a new section. Again, probably not
what you were after.

Take a look around the source tree for examples.

kernel-doc for structs, unions, enums, and typedefs

Beside functions you can also write documentation for structs, unions,
enums and typedefs. Instead of the function name you must write the name
of the declaration; the struct/union/enum/typedef must always precede
the name. Nesting of declarations is not supported.
Use the argument mechanism to document members or constants.

Inside a struct description, you can use the "private:" and "public:"
comment tags. Structure fields that are inside a "private:" area
are not listed in the generated output documentation. The "private:"
and "public:" tags must begin immediately following a "/*" comment
marker. They may optionally include comments between the ":" and the
ending "*/" marker.

Example:

```
/**
 * struct my_struct - short description
 * @a: first member
 * @b: second member
 *
 * Longer description
 */
struct my_struct {
    int a;
    int b;
/* private: internal use only */
    int c;
};
```

Including documentation blocks in source files

To facilitate having source code and comments close together, you can include kernel-doc documentation blocks that are free-form comments instead of being kernel-doc for functions, structures, unions, enums, or typedefs. This could be used for something like a theory of operation for a driver or library code, for example.

This is done by using a DOC: section keyword with a section title. E.g.:

```
/**
 * DOC: Theory of Operation
 *
 * The whizbang foobar is a dilly of a gizmo.  It can do whatever you
 * want it to do, at any time.  It reads your mind.  Here's how it works.
 *
 * foo bar splat
 *
 * The only drawback to this gizmo is that is can sometimes damage
 * hardware, software, or its subject(s).
 */
```

DOC: sections are used in SGML templates files as indicated below.

How to make new SGML template files

SGML template files (*.tmpl) are like normal SGML files, except that they can contain escape sequences where extracted documentation should be inserted.

!E<filename> is replaced by the documentation, in <filename>, for functions that are exported using EXPORT_SYMBOL: the function list is collected from files listed in Documentation/DocBook/Makefile.

!I<filename> is replaced by the documentation for functions that are _not_ exported using EXPORT_SYMBOL.

```
!D<filename> is used to name additional files to search for functions
exported using EXPORT_SYMBOL.

!F<filename> <function [functions...]> is replaced by the
documentation, in <filename>, for the functions listed.

!P<filename> <section title> is replaced by the contents of the DOC:
section titled <section title> from <filename>.
Spaces are allowed in <section title>; do not quote the <section title>.

Tim.
*/ <twaugh@redhat.com>
```