Writing kernel-doc comments

The Linux kernel source files may contain structured documentation comments in the kernel-doc format to describe the functions, types and design of the code. It is easier to keep documentation up-to-date when it is embedded in source files.

> **Note**
>
> The kernel-doc format is deceptively similar to javadoc, gtk-doc or Doxygen, yet distinctively different, for historical reasons. The kernel source contains tens of thousands of kernel-doc comments. Please stick to the style described here.

The kernel-doc structure is extracted from the comments, and proper Sphinx C Domain function and type descriptions with anchors are generated from them. The descriptions are filtered for special kernel-doc highlights and cross-references. See below for details.

Every function that is exported to loadable modules using `EXPORT_SYMBOL` or `EXPORT_SYMBOL_GPL` should have a kernel-doc comment. Functions and data structures in header files which are intended to be used by modules should also have kernel-doc comments.

It is good practice to also 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. This is lower priority and at the discretion of the maintainer of that kernel source file.

How to format kernel-doc comments

The opening comment mark `/**` is used for kernel-doc comments. The `kernel-doc` tool will extract comments marked this way. The rest of the comment is formatted like a normal multi-line comment with a column of asterisks on the left side, closing with `*/` on a line by itself.

The function and type kernel-doc comments should be placed just before the function or type being described in order to maximise the chance that somebody changing the code will also change the documentation. The overview kernel-doc comments may be placed anywhere at the top indentation level.

Running the `kernel-doc` tool with increased verbosity and without actual output generation may be used to verify proper formatting of the documentation comments. For example:

```
scripts/kernel-doc -v -none drivers/foo/bar.c
```

The documentation format is verified by the kernel build when it is requested to perform extra gcc checks:

```
make W=n
```

Function documentation

The general format of a function and function-like macro kernel-doc comment is:

```
/**
 * function_name() - Brief description of function.
 * @arg1: Describe the first argument.
 * @arg2: Describe the second argument.
 *        One can provide multiple line descriptions
 *        for arguments.
 *
 * A longer description, with more discussion of the function function_name()
 * that might be useful to those using or modifying it. Begins with an
 * empty comment line, and may include additional embedded empty
 * comment lines.
 *
 * The longer description may have multiple paragraphs.
 *
 * Context: Describes whether the function can sleep, what locks it takes,
 *          releases, or expects to be held. It can extend over multiple
 *          lines.
 * Return: Describe the return value of function_name.
 *
 * The return value description can also have multiple paragraphs, and should
 * be placed at the end of the comment block.
 */
```

The brief description following the function name may span multiple lines, and ends with an argument description, a blank comment line, or the end of the comment block.

Function parameters

Each function argument should be described in order, immediately following the short function description. Do not leave a blank line

between the function description and the arguments, nor between the arguments.

Each `@argument:` description may span multiple lines.

> **Note**
>
> If the `@argument` description has multiple lines, the continuation of the description should start at the same column as the previous line:
>
> ```
> * @argument: some long description
> *            that continues on next lines
> ```
>
> or:
>
> ```
> * @argument:
> *         some long description
> *         that continues on next lines
> ```

If a function has a variable number of arguments, its description should be written in kernel-doc notation as:

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

Function context

The context in which a function can be called should be described in a section named `Context`. This should include whether the function sleeps or can be called from interrupt context, as well as what locks it takes, releases and expects to be held by its caller.

Examples:

```
* Context: Any context.
* Context: Any context. Takes and releases the RCU lock.
* Context: Any context. Expects <lock> to be held by caller.
* Context: Process context. May sleep if @gfp flags permit.
* Context: Process context. Takes and releases <mutex>.
* Context: Softirq or process context. Takes and releases <lock>, BH-safe.
* Context: Interrupt context.
```

Return values

The return value, if any, should be described in a dedicated section named `Return`.

> **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:
>    * 0 - OK
>    * -EINVAL - invalid argument
>    * -ENOMEM - out of memory
>    ```
>
>   this will all run together and produce:
>
>    ```
>    Return: 0 - OK -EINVAL - invalid argument -ENOMEM - out of memory
>    ```
>
>   So, in order to produce the desired line breaks, you need to use a ReST list, e. g.:
>
>    ```
>    * Return:
>    * * 0             - OK to runtime suspend the device
>    * * -EBUSY        - Device should not be runtime suspended
>    ```
>
> 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 probably won't produce the desired effect.

Structure, union, and enumeration documentation

The general format of a struct, union, and enum kernel-doc comment is:

```
/**
 * struct struct_name - Brief description.
 * @member1: Description of member1.
 * @member2: Description of member2.
 *           One can provide multiple line descriptions
 *           for members.
 *
 * Description of the structure.
 */
```

You can replace the `struct` in the above example with `union` or `enum` to describe unions or enums. `member` is used to mean struct and union member names as well as enumerations in an enum.

The brief description following the structure name may span multiple lines, and ends with a member description, a blank comment line, or the end of the comment block.

Members

Members of structs, unions and enums should be documented the same way as function parameters; they immediately succeed the short description and may be multi-line.

Inside a struct or union 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
 * @d: fourth member
 *
 * Longer description
 */
struct my_struct {
    int a;
    int b;
/* private: internal use only */
    int c;
/* public: the next one is public */
    int d;
};
```

Nested structs/unions

It is possible to document nested structs and unions, like:

```
/**
 * struct nested_foobar - a struct with nested unions and structs
 * @memb1: first member of anonymous union/anonymous struct
 * @memb2: second member of anonymous union/anonymous struct
 * @memb3: third member of anonymous union/anonymous struct
 * @memb4: fourth member of anonymous union/anonymous struct
 * @bar: non-anonymous union
 * @bar.st1: struct st1 inside @bar
 * @bar.st2: struct st2 inside @bar
 * @bar.st1.memb1: first member of struct st1 on union bar
 * @bar.st1.memb2: second member of struct st1 on union bar
 * @bar.st2.memb1: first member of struct st2 on union bar
 * @bar.st2.memb2: second member of struct st2 on union bar
 */
struct nested_foobar {
  /* Anonymous union/struct*/
  union {
    struct {
      int memb1;
      int memb2;
    };
    struct {
      void *memb3;
      int memb4;
    };
  };
  union {
    struct {
      int memb1;
      int memb2;
    } st1;
    struct {
      void *memb1;
      int memb2;
    } st2;
  } bar;
};
```

> **Note**

1. When documenting nested structs or unions, if the struct/union `foo` is named, the member `bar` inside it should be documented as `@foo.bar:`
2. When the nested struct/union is anonymous, the member `bar` in it should be documented as `@bar:`

In-line member documentation comments

The structure members may also be documented in-line within the definition. There are two styles, single-line comments where both the opening `/**` and closing `*/` are on the same line, and multi-line comments where they are each on a line of their own, like all other kernel-doc comments:

```
/**
 * struct foo - Brief description.
 * @foo: The Foo member.
 */
struct foo {
      int foo;
      /**
       * @bar: The Bar member.
       */
      int bar;
      /**
       * @baz: The Baz member.
       *
       * Here, the member description may contain several paragraphs.
       */
      int baz;
      union {
              /** @foobar: Single line description. */
              int foobar;
      };
      /** @bar2: Description for struct @bar2 inside @foo */
      struct {
              /**
               * @bar2.barbar: Description for @barbar inside @foo.bar2
               */
              int barbar;
      } bar2;
};
```

Typedef documentation

The general format of a typedef kernel-doc comment is:

```
/**
 * typedef type_name - Brief description.
 *
 * Description of the type.
 */
```

Typedefs with function prototypes can also be documented:

```
/**
 * typedef type_name - Brief description.
 * @arg1: description of arg1
 * @arg2: description of arg2
 *
 * Description of the type.
 *
 * Context: Locking context.
 * Return: Meaning of the return value.
 */
 typedef void (*type_name)(struct v4l2_ctrl *arg1, void *arg2);
```

Highlights and cross-references

The following special patterns are recognized in the kernel-doc comment descriptive text and converted to proper reStructuredText markup and Sphinx C Domain references.

> **Attention!**
>
> The below are **only** recognized within kernel-doc comments, **not** within normal reStructuredText documents.

`funcname()`

Function reference.

`@parameter`

Name of a function parameter. (No cross-referencing, just formatting.)

`%CONST`

Name of a constant. (No cross-referencing, just formatting.)

```
``literal``
```

A literal block that should be handled as-is. The output will use a `monospaced font`.

Useful if you need to use special characters that would otherwise have some meaning either by kernel-doc script or by reStructuredText.

This is particularly useful if you need to use things like `%ph` inside a function description.

`$ENVVAR`

Name of an environment variable. (No cross-referencing, just formatting.)

`&struct name`

Structure reference.

`&enum name`

Enum reference.

`&typedef name`

Typedef reference.

`&struct_name->member` or `&struct_name.member`

Structure or union member reference. The cross-reference will be to the struct or union definition, not the member directly.

`&name`

A generic type reference. Prefer using the full reference described above instead. This is mostly for legacy comments.

Cross-referencing from reStructuredText

No additional syntax is needed to cross-reference the functions and types defined in the kernel-doc comments from reStructuredText documents. Just end function names with `()` and write `struct`, `union`, `enum` or `typedef` before types. For example:

```
See foo().
See struct foo.
See union bar.
See enum baz.
See typedef meh.
```

However, if you want custom text in the cross-reference link, that can be done through the following syntax:

```
See :c:func:`my custom link text for function foo <foo>`.
See :c:type:`my custom link text for struct bar <bar>`.
```

For further details, please refer to the Sphinx C Domain documentation.

Overview documentation comments

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.

The general format of an overview or high-level documentation comment is:

```
/**
 * 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).
 */
```

The title following `DOC:` acts as a heading within the source file, but also as an identifier for extracting the documentation comment. Thus, the title must be unique within the file.

Including kernel-doc comments

The documentation comments may be included in any of the reStructuredText documents using a dedicated kernel-doc Sphinx directive extension.

The kernel-doc directive is of the format:

```
.. kernel-doc:: source
   :option:
```

The *source* is the path to a source file, relative to the kernel source tree. The following directive options are supported:

export: *[source-pattern ...]*

Include documentation for all functions in *source* that have been exported using `EXPORT_SYMBOL` or `EXPORT_SYMBOL_GPL` either in *source* or in any of the files specified by *source-pattern*.

The *source-pattern* is useful when the kernel-doc comments have been placed in header files, while `EXPORT_SYMBOL` and `EXPORT_SYMBOL_GPL` are next to the function definitions.

Examples:

```
.. kernel-doc:: lib/bitmap.c
   :export:

.. kernel-doc:: include/net/mac80211.h
   :export: net/mac80211/*.c
```

internal: *[source-pattern ...]*

Include documentation for all functions and types in *source* that have **not** been exported using `EXPORT_SYMBOL` or `EXPORT_SYMBOL_GPL` either in *source* or in any of the files specified by *source-pattern*.

Example:

```
.. kernel-doc:: drivers/gpu/drm/i915/intel_audio.c
   :internal:
```

identifiers: *[function/type ...]*

Include documentation for each *function* and *type* in *source*. If no *function* is specified, the documentation for all functions and types in the *source* will be included.

Examples:

```
.. kernel-doc:: lib/bitmap.c
   :identifiers: bitmap_parselist bitmap_parselist_user

.. kernel-doc:: lib/idr.c
   :identifiers:
```

no-identifiers: *[function/type ...]*

Exclude documentation for each *function* and *type* in *source*.

Example:

```
.. kernel-doc:: lib/bitmap.c
   :no-identifiers: bitmap_parselist
```

functions: *[function/type ...]*

This is an alias of the 'identifiers' directive and deprecated.

doc: *title*

Include documentation for the `DOC:` paragraph identified by *title* in *source*. Spaces are allowed in *title*; do not quote the *title*. The *title* is only used as an identifier for the paragraph, and is not included in the output. Please make sure to have an appropriate heading in the enclosing reStructuredText document.

Example:

```
.. kernel-doc:: drivers/gpu/drm/i915/intel_audio.c
   :doc: High Definition Audio over HDMI and Display Port
```

Without options, the kernel-doc directive includes all documentation comments from the source file.

The kernel-doc extension is included in the kernel source tree, at `Documentation/sphinx/kerneldoc.py`. Internally, it uses the `scripts/kernel-doc` script to extract the documentation comments from the source.

How to use kernel-doc to generate man pages

If you just want to use kernel-doc to generate man pages you can do this from the kernel git tree:

```
$ scripts/kernel-doc -man \
  $(git grep -l '/\*\*' -- :^Documentation :^tools) \
```

```
    | scripts/split-man.pl /tmp/man
```

Some older versions of git do not support some of the variants of syntax for path exclusion. One of the following commands may work for those versions:

```
$ scripts/kernel-doc -man \
  $(git grep -l '/\*\*' -- . ':!Documentation' ':!tools') \
  | scripts/split-man.pl /tmp/man

$ scripts/kernel-doc -man \
  $(git grep -l '/\*\*' -- . ":(exclude)Documentation" ":(exclude)tools") \
  | scripts/split-man.pl /tmp/man
```