



# Как сгенерировать документацию из исходного кода в Linux

*Последнее обновление 9 сентября 2020 г. от Дэна Нанни*

Если вы являетесь разработчиком с открытым исходным кодом и хотите опубликовать свой проект для широкой публики, вы можете рассмотреть возможность публикации документации с исходным кодом для проекта. В случаях, когда вы пытаетесь прочитать исходный код, написанный другими пользователями, также будет полезно, если вы сможете получить *обзор с высоты* полета на зашифрованный исходный код.

В Linux `doxygen` является де-факто стандартным инструментом для автоматической генерации документации с перекрестными ссылками из аннотированного исходного кода. Он поддерживает основные языки программирования, включая C / C ++, Objective-C, C #, PHP, Java и Python. Существует более 350 проектов с открытым исходным кодом (например, Drupal, Gaim, GNU C ++ library, KDE), которые полагаются на `doxygen` автоматическое документирование своего исходного кода.

Используя `doxygen`, вы можете автоматически сгенерировать справочную документацию API по исходному коду в формате HTML или LaTeX. Вы также можете визуализировать зависимость от класса и взаимосвязь между различными исходными файлами.

В этом руководстве я объясню, как **автоматически сгенерировать документацию из исходного кода в Linux** с помощью `doxygen`.

# Установите Doxygen в Linux

## Для Ubuntu, Debian или Linux Mint:

Для установки `doxygen` в Ubuntu, Linux Mint или Debian:

xterm

```
$ sudo apt-получить установку doxygen  
$ sudo apt-получить установку graphviz
```

## Для CentOS, Fedora или RHEL:

Для установки `doxygen` на CentOS, Fedora или RHEL:

xterm

```
$ sudo yum устанавливает doxygen  
$ sudo yum устанавливает graphviz
```

## Сгенерируйте документацию из исходного кода с помощью `doxygen`

Чтобы сгенерировать документацию по исходному коду, выполните следующие действия.

Сначала сгенерируйте файл конфигурации для конкретного проекта `doxygen`:

xterm

```
$ doxygen -g my_proj.conf
```

Приведенная выше команда сгенерирует файл конфигурации шаблона для конкретного проекта, который вы можете дополнительно настроить, как описано ниже.

Среди прочего, вы можете отредактировать следующие параметры в файле конфигурации.

```
# документировать все объекты в проекте.  
EXTRACT_ALL = ДА  
  
# документировать все статические элементы файла.  
EXTRACT_STATIC = ДА  
  
# указать корневой каталог, содержащий исходные файлы  
проекта.  
INPUT = /home / xmodulo /source  
  
# найдите подкаталоги для всех исходных файлов.  
RECURSIVE = ДА  
  
# включите тело функций и классов в документацию.  
INLINE_SOURCES = ДА  
  
# сгенерируйте график визуализации с помощью программы  
dot (часть пакета graphviz).  
HAVE_DOT = ДА
```

Теперь продолжайте и запустите `doxygen` с файлом конфигурации.

xterm

```
$ doxygen my_proj.conf
```

Документация генерируется как в форматах HTML, так и Latex и хранится в `./html` и `./latex` каталогах соответственно.

Чтобы просмотреть документацию в формате HTML, вы можете использовать любой веб-браузер и открыть индексный файл HTML.

xterm

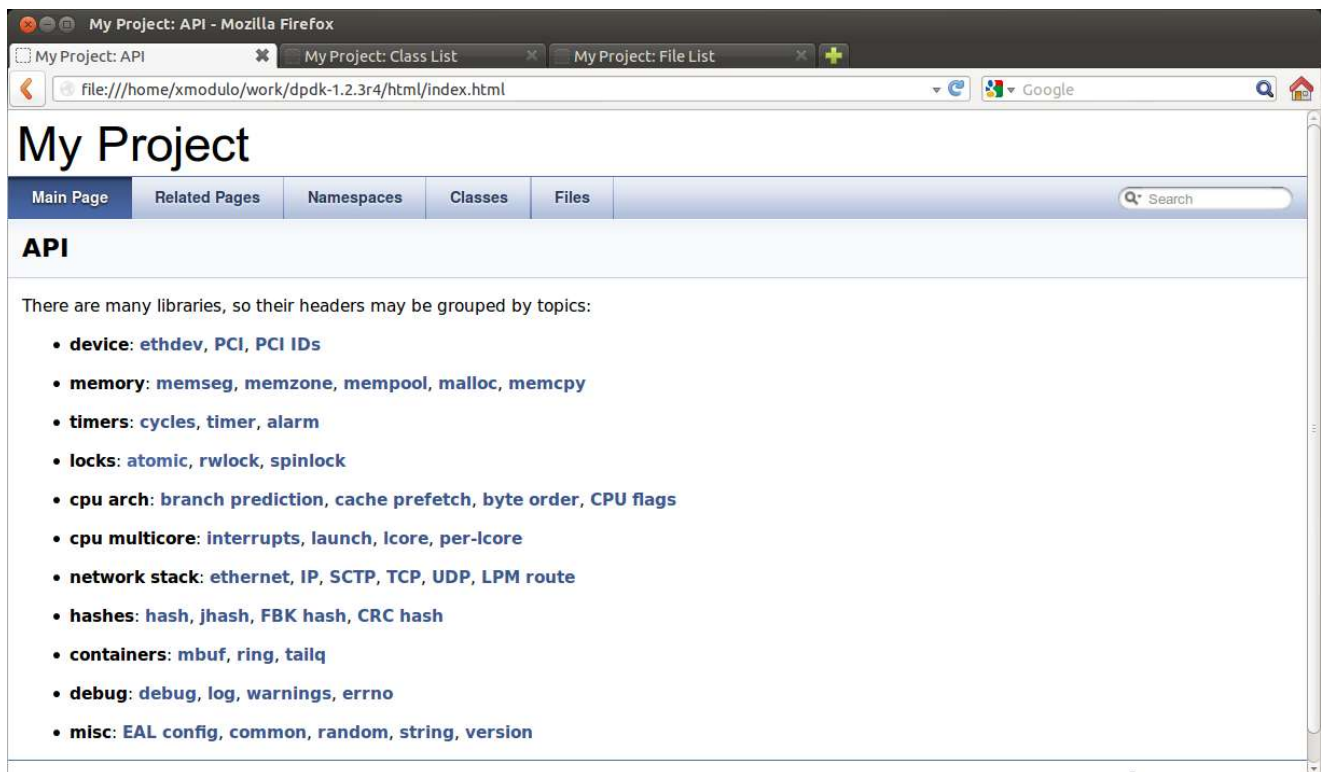
```
$ cd html
```

```
$ firefox index.html
```

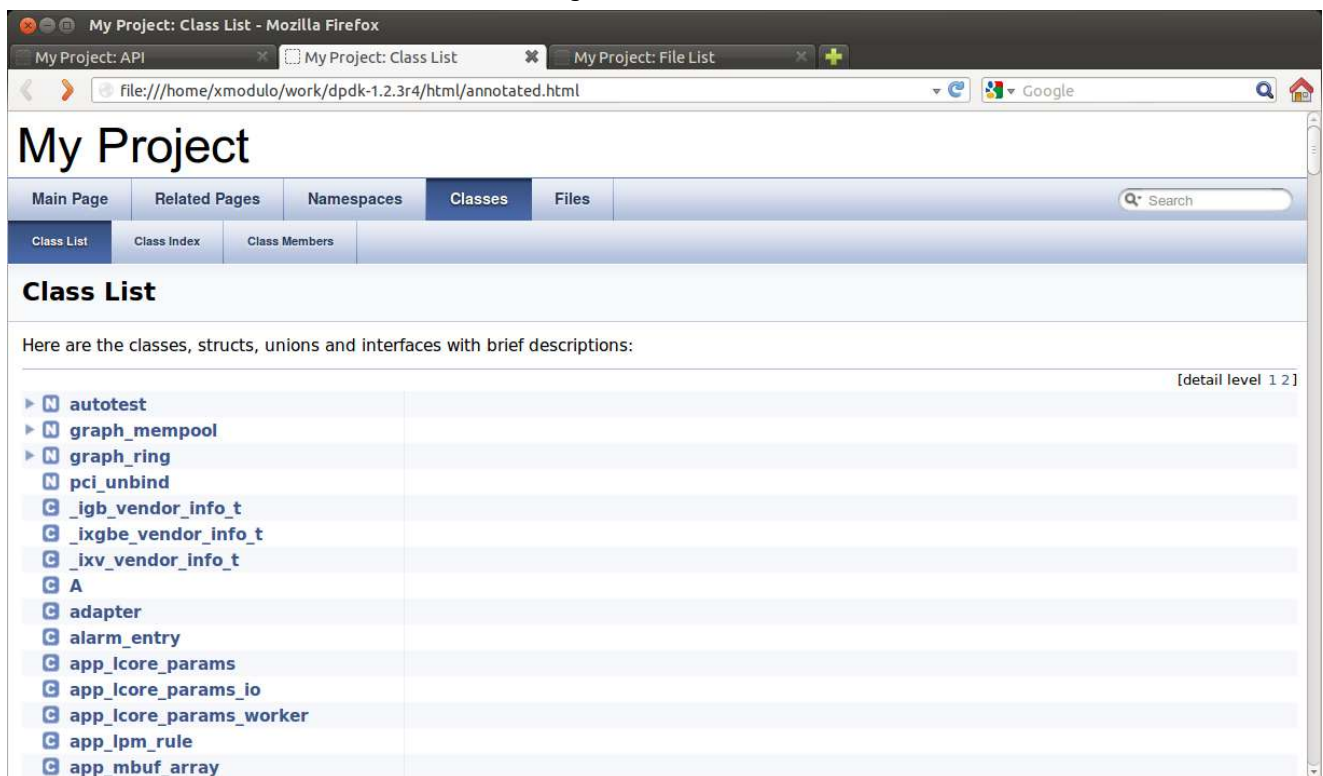
## doxygen Скриншоты документации

Вот несколько скриншотов с примерами документации, сгенерированной **doxygen**.

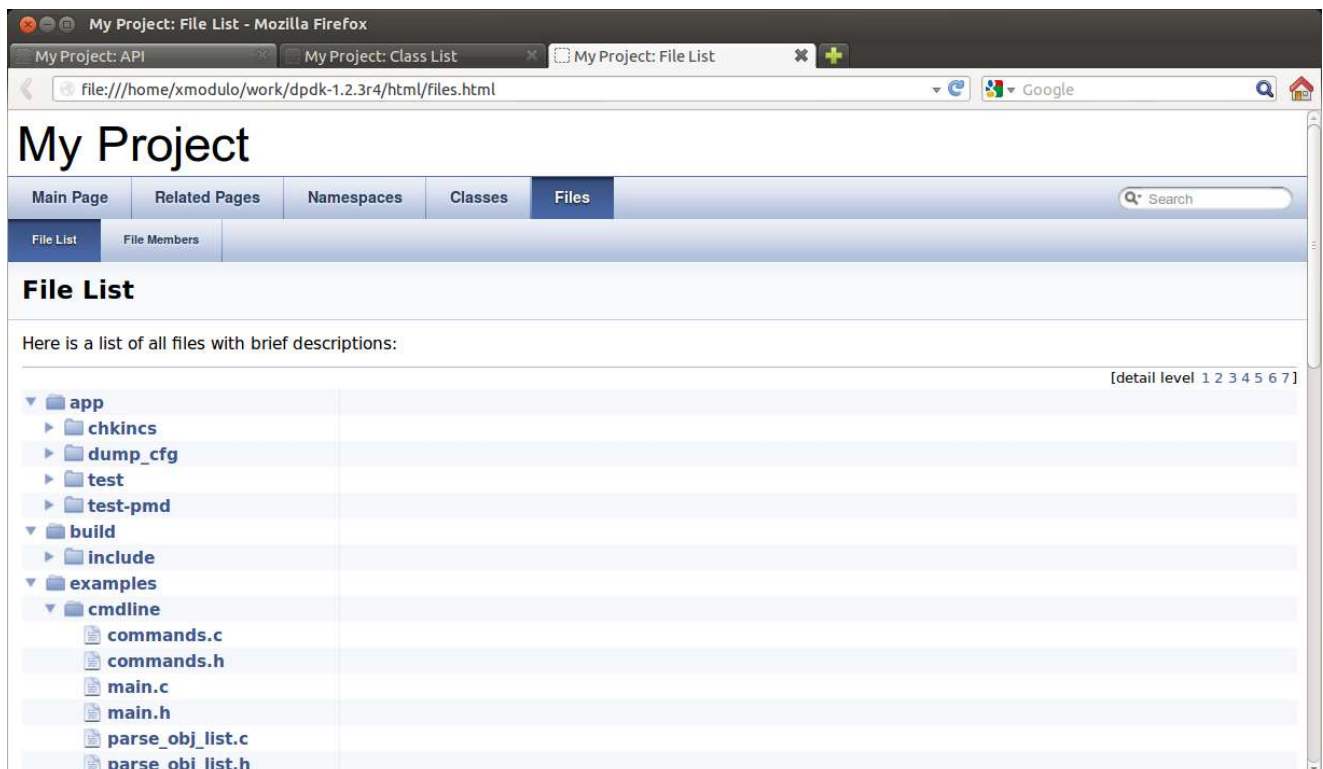
Здесь показан список заголовочных файлов, которые автоматически классифицируются по темам.



Здесь показан список классов, структур, объединений и интерфейсов с описаниями.



Здесь показан браузер исходного кода, рекурсивно перечисляющий все исходные файлы в подкаталогах.



Если вы нажмете на конкретный исходный файл, вы увидите страницу, на которой показан график зависимостей для файла, а также документация для всех определенных функций.

My Project: examples/helloworld/main.c File Reference - Mozilla Firefox

file:///home/hyunseok/work/dpdk-1.2.3r4/html/helloworld\_2main\_8c.html

Include dependency graph for main.c:

```

graph TD
    main["examples/helloworld/main.c"]
    main --> stdio_h["stdio.h"]
    main --> string_h["string.h"]
    main --> stdint_h["stdint.h"]
    main --> errno_h["errno.h"]
    main --> sys_queue_h["sys/queue.h"]
    main --> rte_memory_h["rte_memory.h"]
    main --> rte_memzone_h["rte_memzone.h"]
    main --> rte_launch_h["rte_launch.h"]
    main --> rte_tailq_h["rte_tailq.h"]
    main --> rte_eal_h["rte_eal.h"]
    main --> rte_per_core_h["rte_per_core.h"]
  
```

### Functions

static int **lcore\_hello** ( **\_attribute\_\_((unused)) void \*arg** )  
 int **MAIN** (int argc, char \*\*argv)

### Function Documentation

**static int lcore\_hello ( **\_attribute\_\_((unused)) void \* arg** )** static

```

{
    unsigned lcore_id;
    lcore_id = rte_lcore_id();
    printf("hello from core %u\n", lcore_id);
    return 0;
}
  
```

**int MAIN ( int **argc**,  
 char \*\* **argv**  
 )**

```

{
    int ret;
  
```

Здесь показано подробное представление определений функций / макросов. Ниже на рисунке показан фактический фрагмент исходного кода, соответствующий этой документированной части.

My Project: lib/librte\_eal/common/include/rte\_debug.h File Reference - Mozilla Firefox

file:///home/hyunseok/work/dpdk-1.2.3r4/html/lib\_2librte\_eal\_2common\_2include\_2rte\_debug\_8h.html#a9fcat

### Detailed Description

Debug Functions in RTE

This file defines a generic API for debug operations. Part of the implementation is architecture-specific.

### Macro Definition Documentation

**#define rte\_panic ( ... ) **rte\_panic** ( **\_func\_**, **\_VA\_ARGS\_**, "dummy" )**

Provide notification of a critical non-recoverable error and terminate execution abnormally.

Display the format string and its expanded arguments (printf-like).

In a linuxapp environment, this function dumps the stack and calls abort() resulting in a core dump if enabled.

The function never returns.

**Parameters**

**format** The format string

**args** The variable list of arguments.

```

#define rte_panic( func,  

    format,  

    ...  

) __rte_panic(func, format "%0s", _VA_ARGS_)
  
```

```

/**
 * Provide notification of a critical non-recoverable error and terminate
 * execution abnormally.
 *
 * Display the format string and its expanded arguments (printf-like).
 *
 * In a linuxapp environment, this function dumps the stack and calls
 * abort() resulting in a core dump if enabled.
 *
 * The function never returns.
 *
 * @param format
 *   The format string
 * @param args
 *   The variable list of arguments.
 */
#define rte_panic(...) rte_panic(__func__, __VA_ARGS__, "dummy")
#define rte_panic(func, format, ...) __rte_panic(func, format "%.0s", __VA_ARGS__
__)

/**
 * Provide notification of a critical non-recoverable error and stop.

```

## Support Xmodulo

This website is made possible by minimal ads and your gracious donation via [PayPal](#) or [credit card](#)

Please note that this article is published by Xmodulo.com under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you would like to use the whole or any part of this article, you need to cite this web page at Xmodulo.com as the original source.





0 Comments

 Login

G

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name



Share

Best Newest Oldest

Be the first to comment.

[Subscribe](#) [Privacy](#) [Do Not Sell My Data](#)