



# **XRADIO Console Command Developer Guide**

---

**Revision 1.1**

**Oct 11, 2019**

## Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO. THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

## Revision History

Version	Date	Summary of Changes
1.0	2019-9-30	Initial Version
1.1	2019-10-11	格式调整

**Table 1- 1 Revision History**

## Contents

Declaration.....	2
Revision History.....	3
Contents.....	4
Tables.....	5
1 概述.....	6
1.1 实现效果.....	6
2 Console Command 添加步骤.....	7
2.1 添加步骤介绍.....	7
2.1.1 使能 console 的功能.....	7
2.1.2 添加 command 文件.....	7
2.1.3 添加命令.....	8
3 使用示例.....	11

---

## Tables

表 2-1 命令格式说明.....	9
-------------------	---

# 1 概述

---

Console Command 是 SDK 中常用的测试及调试手段。通过串口的命令的输入，可以简单地实现各种功能，例如使用 mem 命令去获取内存地址的内容等。

本文主要介绍了如何往工程里面添加 Console Command，以便于高效的开发。

## 1.1 实现效果

工程支持 Console Command 的功能后，可以往工程里面添加各种命令。在具体实现时，通过串口传入命令到控制台，然后控制台解析命令后将信息传给内部模块去执行相对应操作，最后将处理结果响应给控制台。如下所示的命令操作：

\$ mem r32 0x40040000 4	#console 输入的命令，获取地址 0x40040000 的内容
40040000: 00000001	#console 响应的命令，返回地址 0x40040000 的内容

## 2 Console Command 添加步骤

本章节将介绍在工程中添加 console command 的步骤说明。

### 2.1 添加步骤介绍

#### 2.1.1 使能 console 的功能

- 1) 打开所在工程的 prj\_config.h，将宏 PRJCONF\_CONSOLE\_EN 修改为 1。
- 2) 在平台初始化时，由于 console 的功能使能，系统会创建一个控制台的任务，控制台任务默认的栈大小为 2KB。

#### 2.1.2 添加 command 文件

- 1) 往工程添加 command.c、command.h 文件，文件目录结构如下：

```
.
├── command.c           # 本工程的控制台命令入口和命令定义
├── command.h
├── gcc
│   ├── localconfig.mk  # 本工程的配置规则，用于覆盖默认配置
│   └── Makefile        # 本工程的编译选项，可覆盖默认配置
├── image
│   └── xr872
│       └── image.cfg    # 本工程的镜像分区配置
├── main.c             # 本工程的入口，进行 JPEG 图像捕捉到的示例说明
└── prj_config.h       # 本工程的配置规则
```

- 2) command.c 文件内容填充如下：

```
#include "common/cmd/cmd_util.h"
#include "common/cmd/cmd.h"

/*
 * main commands
```

```
*/  
  
static const struct cmd_data g_main_cmds[] = {  
};  
  
void main_cmd_exec(char *cmd)  
{  
    enum cmd_status status;  
  
    if (cmd[0] != '\0') {  
#if (!CONSOLE_ECHO_EN)  
        if (cmd_strcmp(cmd, "efpg"))  
            CMD_LOG(CMD_DBG_ON, "$ %s\n", cmd);  
#endif  
        status = cmd_exec(cmd, g_main_cmds, cmd_nitems(g_main_cmds));  
        if (status != CMD_STATUS_ACKED) {  
            cmd_write_respond(status, cmd_get_status_desc(status));  
        }  
    }  
#if (!CONSOLE_ECHO_EN)  
    else { /* empty command */  
        CMD_LOG(1, "$\n");  
    }  
#endif  
#if CONSOLE_ECHO_EN  
    console_write((uint8_t *)"$ ", 2);  
#endif  
}
```

### 2.1.3 添加命令

添加命令的方式有多种，可以选择添加 SDK 已提供的测试命令，也可以添加自定义命令。命令的格式如下：

```
/* command format: <command-name> <arg>... */  
struct cmd_data {
```



```
char *name;

enum cmd_status (*exec)(char *);

};
```

Reference	Comments
<b>name</b>	命令的名称，可自定义
<b>exec</b>	命令所对应的执行函数，可以传递参数

表 2-1 命令格式说明

往串口输入的命令格式为: <command-name> <arg>...

### 1) 添加已有的命令

例如为了固件烧写方便，可以添加 upgrade 命令。在 command.c 文件中，往 g\_main\_cmds 数组中添加 cmd\_upgrade\_exec 命令。

```
static const struct cmd_data g_main_cmds[] = {
    { "upgrade", cmd_upgrade_exec },
};
```

其中，upgrade 是命令的名称，cmd\_upgrade\_exec 是命令所对应的执行函数。因此，往串口输入“upgrade”将会执行 cmd\_upgrade\_exec 的函数。

### 2) 添加自定义命令

例如为了测试 FatFs 文件的读写操作，可以自定义 FatFs 的命令操作集。

定义 FatFs 的命令操作集：

```
static const struct cmd_data g_fs_cmds[] = {
    { "open", cmd_fs_open_exec },
    { "read", cmd_fs_read_exec },
    { "write", cmd_fs_write_exec },
    { "close", cmd_fs_close_exec },
};

enum cmd_status cmd_fs_exec(char *cmd)
{
    return cmd_exec(cmd, g_fs_cmds, cmd_nitems(g_fs_cmds));
}
```

然后将 cmd\_fs\_exec 添加到 g\_main\_cmds 数组中去：

```
static const struct cmd_data g_main_cmds[] = {  
    { "upgrade",cmd_upgrade_exec },  
    { "fs",cmd_fs_exec },  
};
```

最后，依次实现 cmd\_fs\_open\_exec 等函数实现即可。

## 3 使用示例

---

本章节通过自定义 FatFs 命令来演示 console command 的添加流程，步骤如下：

1) 按照章节 2.1.2 的文件目录结构创建组织好相关文件。

2) prj\_config.h 文件的内容修改如下：

```
#ifndef _PRJ_CONFIG_H_
#define _PRJ_CONFIG_H_

#ifdef __cplusplus
extern "C" {
#endif

/*
 * project base config
 */

/* stack size for IRQ service */
#define PRJCONF_MSP_STACK_SIZE          (1 * 1024)

/* main thread priority */
#define PRJCONF_MAIN_THREAD_PRIO        OS_THREAD_PRIO_APP

/* main thread stack size */
#define PRJCONF_MAIN_THREAD_STACK_SIZE  (2 * 1024)

/*
 * project hardware feature
 */

/* uart enable/disable */
#define PRJCONF_UART_EN                  1

#define PRJCONF_CONSOLE_EN              1
```

```
/* mmc enable/disable */
#define PRJCONF_MMC_EN                1

/* mmc detect mode */
#define PRJCONF_MMC_DETECT_MODE        CARD_ALWAYS_PRESENT

#ifdef __cplusplus
}
#endif

#endif /* _PRJ_CONFIG_H_ */
```

3) command.c 文件如下:

```
#include "common/cmd/cmd_util.h"
#include "common/cmd/cmd.h"
#include "fs/fatfs/ff.h"
#include "common/framework/fs_ctrl.h"

static FIL fp;

static enum cmd_status cmd_fs_open_exec(char *cmd)
{
    int ret;
    char *path = cmd;
    ret = f_open (&fp, path, FA_OPEN_ALWAYS);
    if (ret != FR_OK) {
        CMD_ERR("open fail, %d\n", ret);
        return CMD_STATUS_FAIL;
    } else {
        CMD_DBG("open success\n");
    }
    return CMD_STATUS_OK;
}
```

```
static enum cmd_status cmd_fs_write_exec(char *cmd)
{
    int ret;
    uint32_t bw;
    char *buff = (char*)cmd;
    uint32_t len = strlen(buff);

    ret = f_write(&fp, buff, len, &bw);
    if (ret != FR_OK || bw < len) {
        CMD_ERR("write fail, %d\n", ret);
        return CMD_STATUS_FAIL;
    } else {
        CMD_DBG("open success\n");
    }
    return CMD_STATUS_OK;
}

static enum cmd_status cmd_fs_close_exec(char *cmd)
{
    int ret;
    if ((ret = (f_close(&fp) != FR_OK))) {
        CMD_ERR("close fail, %d\n", ret);
        return CMD_STATUS_FAIL;
    }
    return CMD_STATUS_OK;
}

static const struct cmd_data g_fs_cmds[] = {
    { "open", cmd_fs_open_exec },
    { "write", cmd_fs_write_exec },
    { "close", cmd_fs_close_exec },
};
```

```
enum cmd_status cmd_fs_exec(char *cmd)
{
    return cmd_exec(cmd, g_fs_cmds, cmd_nitems(g_fs_cmds));
}

/*
 * main commands
 */
static const struct cmd_data g_main_cmds[] = {
    { "upgrade", cmd_upgrade_exec },
    { "fs",      cmd_fs_exec },
};

void main_cmd_exec(char *cmd)
{
    enum cmd_status status;

    if (cmd[0] != '\0') {
#ifdef !CONSOLE_ECHO_EN
        if (cmd_strcmp(cmd, "efpg"))
            CMD_LOG(CMD_DBG_ON, "$ %s\n", cmd);
#endif
        status = cmd_exec(cmd, g_main_cmds, cmd_nitems(g_main_cmds));
        if (status != CMD_STATUS_ACKED) {
            cmd_write_respond(status, cmd_get_status_desc(status));
        }
    }
#ifdef !CONSOLE_ECHO_EN
    else { /* empty command */
        CMD_LOG(1, "$\n");
    }
#endif
#ifdef CONSOLE_ECHO_EN
    console_write((uint8_t *)"$ ", 2);
#endif
}
```

```
#endif  
}
```

4) command.h 文件如下:

```
#ifndef _COMMAND_H_  
#define _COMMAND_H_  
  
#ifdef __cplusplus  
extern "C" {  
#endif  
  
void main_cmd_exec(char *cmd);  
  
#ifdef __cplusplus  
}  
#endif  
  
#endif /* _COMMAND_H_ */
```

5) 编译运行，在串口控制台可以通过命令执行以上的命令功能:

```
$ fs open test.txt      #打开创建文件 test.txt  
$ fs write abcdef       #往文件 test.txt 写入数据“abcdef”  
$ fs close              #关闭文件 test.txt
```