



StarFive
赛昉科技

Software SDK Developer Guide for U-Boot

VisionFive 2

Version: 1.0

Date: 2022/11/24

Doc ID: JH7110-DGEN-001

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Copyright © Shanghai StarFive Technology Co., Ltd., 2022. All rights reserved.

Information in this document is provided "as is," with all faults. Contents may be periodically updated or revised due to product development. Shanghai StarFive Technology Co., Ltd. (hereinafter "StarFive") reserves the right to make changes without further notice to any products herein.

StarFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement.

StarFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

All material appearing in this document is protected by copyright and is the property of StarFive. You may not reproduce the information contained herein, in whole or in part, without the written permission of StarFive.

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: <http://www.starfivetech.com>

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers with the programming basics and debugging know-how for the U-Boot module of the StarFive next generation SoC platform - JH7110.

Audience

This document mainly serves the U-Boot relevant developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH7110.






Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0		First official release.

Notes and notices

The following notes and notices might appear in this guide:

-  **Tip:**
Suggests how to apply the information in a topic or step.
-  **Note:**
Explains a special case or expands on an important point.
-  **Important:**
Points out critical information concerning a topic or step.
-  **CAUTION:**
Indicates that an action or step can cause loss of data, security problems, or performance issues.
-  **Warning:**
Indicates that an action or step can result in physical harm or cause damage to hardware.

Contents

List of Tables.....	5
List of Figures.....	6
Legal Statements.....	ii
Preface.....	iii
1. Introduction.....	7
1.1. Function Introduction.....	7
1.2. Device Tree Overview.....	7
1.3. Device Tree Source Code.....	8
2. Configuration.....	9
2.1. Kernel Menu Deconfiguration.....	9
2.2. Kernel Menu Configuration.....	10
2.3. U-Boot DTS.....	11
3. Interface Description.....	12
3.1. FDT Interfaces.....	12
3.1.1. fdt_getprop.....	12
3.1.2. fdt_set_node_status.....	12
3.1.3. fdt_path_offset.....	13
3.2. ENV Interfaces.....	13
3.2.1. int env_set.....	13
3.2.2. env_get.....	14
3.2.3. env_save.....	14
3.3. U-Boot Commands.....	14
3.3.1. run_command_list.....	14
3.3.2. do_bootm.....	15
3.3.3. do_booti.....	15
3.3.4. do_tftp.....	15
3.3.5. do_mmcinfo.....	16
3.4. Flash Read and Write.....	16
3.4.1. spi_flash_read.....	16
3.4.2. spi_flash_write.....	16
3.5. Partition Information.....	17
3.5.1. part_get_info_by_name.....	17
3.6. GPIO Operations.....	17
3.6.1. gpio_get_value.....	17
3.6.2. gpio_set_value.....	18
4. Debug Methods.....	19
4.1. DM_DEBUG.....	19
4.2. DEBUG_DEVRES.....	19
4.3. TOOLS_DEBUG.....	19

List of Tables

Table 0-1 Revision History.....	iii
Table 2-1 Deconfiguration Options.....	11

StarFive

List of Figures

Figure 1-1 Device Tree Workflow.....	8
Figure 2-1 Deconfiguration File.....	9
Figure 2-2 Menu Configuration Dialog.....	10

StarFive

1. Introduction

In embedded systems, Bootloader/U-Boot runs before the Linux kernel. By initializing hardware devices and building mapping relations between memory spaces, U-Boot creates an appropriate hardware and software environment, as a preparation for building the operating system.

1.1. Function Introduction

For the JH7110 SoC platform, U-Boot is not only used for system bring-up, but also for burning and upgrading.

A U-Boot program usually has the following functions.

- Boot the kernel:
Load image from storage media `nand/mmc/spinor` to DRAM and run.
- Mass-production and upgrading:
Mass-production of cards, USB sticks, private data burning and firmware upgrade.
- Fast-boot:
Use standard fast-boot commands to flash images on boards.

1.2. Device Tree Overview

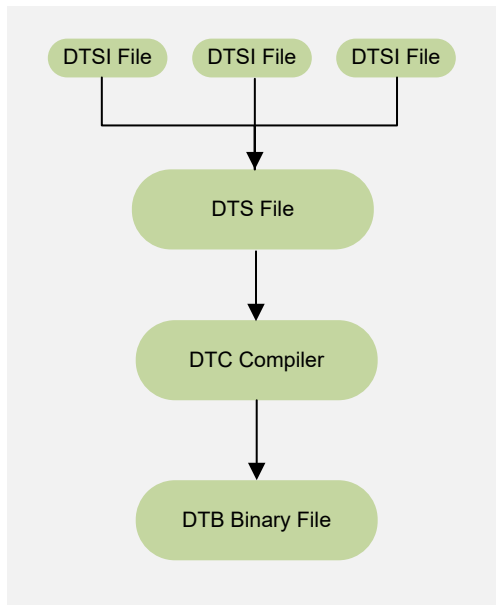
Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

A device tree is primarily represented in the following forms.

- *Device Tree Compiler (DTC)*: The tool used to compile device tree into system-readable binaries.
- *Device Tree Source (DTS)*: The human-readable device tree description file. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Source Information (DTSI)*: The human-readable header file which you can include in device tree description. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Blob (DTB)*: The system-readable device tree binary blob files which is burned in system for execution.

The following diagram shows the relationship (workflow) of the above forms.

Figure 1-1 Device Tree Workflow



1.3. Device Tree Source Code

The U-boot DTS file is in the following path:

```
freelight-u-sdk/u-boot/arch/riscv/dts/jh7110-u-boot.dtsi
```

For VisionFive 2, use the following DTS file: `starfive_visionfive-u-boot.dtsi`.

2. Configuration

2.1. Kernel Menu Deconfiguration

Follow the steps below to enable the kernel menu deconfiguration for U-Boot.

1. Use the following command to open the configuration file (with the ".configuration" extension).

- For VisionFive 2:

```
vim freelight-u-sdk/u-boot/configs/starfive_visionfive_defconfig
```

2. Locate your target macro definition item, and use the following approach to disable the function if needed.

- To disable a function, simply add a "#" ahead of the corresponding macro definition, or change the "y" inline with a macro definition to "n".



Note:

You are not allowed to enable a function in the deconfiguration file. If you need to re-enable a disabled function, perform the operation using `menuconfig` following [Kernel Menu Configuration \(on page 10\)](#). The deconfiguration file will be updated once you have successfully enabled your target function.

The following image shows an example of the configuration file.

Figure 2-1 Deconfiguration File

```
1 CONFIG_RISCV=y
2 CONFIG_SYS_MALLOC_F_LEN=0x8000
3 CONFIG_NR_DRAM_BANKS=1
4 CONFIG_SPL_DM_SPI=y
5 CONFIG_DEFAULT_DEVICE_TREE="starfive_visionfive"
6 CONFIG_SPL_MMC_SUPPORT=y
7 CONFIG_SPL=y
8 CONFIG_SPL_SPI_FLASH_SUPPORT=y
9 CONFIG_SPL_SPI_SUPPORT=y
10 CONFIG_BUILD_TARGET=""
11 CONFIG_TARGET_STARFIVE_VISIONFIVE=y
12 CONFIG_NR_CPUS=5
13 CONFIG_ARCH_RV64I=y
14 CONFIG_CMODEL_MEDANY=y
15 CONFIG_RISCV_SMODE=y
16 CONFIG_SHOW_REGS=y
17 CONFIG_FIT=y
18 CONFIG_SPL_FIT_SOURCE="jh7110-uboot-fit-image.its"
```

The following table provides descriptions for some typical macro definitions.

- **CONFIG_RISCV**: Whether to support the RISC-V ISA.
- **CONFIG_SYS_MALLOC_F_LEN**: Field length for the **malloc** function.
- **CONFIG_NR_DRAM_BANKS**: Total number of the DRAM banks.
- **CONFIG_SPL_DM_SPI**: Whether to enable device management via SPI.
- **CONFIG_DEFAULT_DEVICE_TREE**: Name of the default device tree file for DT control.
- **CONFIG_SPL_MMC_SUPPORT**: Whether to enable the **mmc** command support for *Secondary Program Loader (SPL)*.
- **CONFIG_SPL**: Whether to enable SPL.
- **CONFIG_SPL_SPI_FLASH_SUPPORT**: Whether to enable the SPI flash support for SPL.
- **CONFIG_SPL_SPI_SUPPORT**: Whether to enable the SPI support for SPL.
- **CONFIG_BUILD_TARGET**: Name of the build target file.
- **CONFIG_TARGET_STARFIVE_VISIONFIVE**: Whether to support VisionFive 2.
- **CONFIG_SPL_FIT_SOURCE**: The `"*.its"` source file for U-Boot FIT image.
- **CONFIG_SYS_PROMPT**: Shell prompt.
- And so on...

2.2. Kernel Menu Configuration

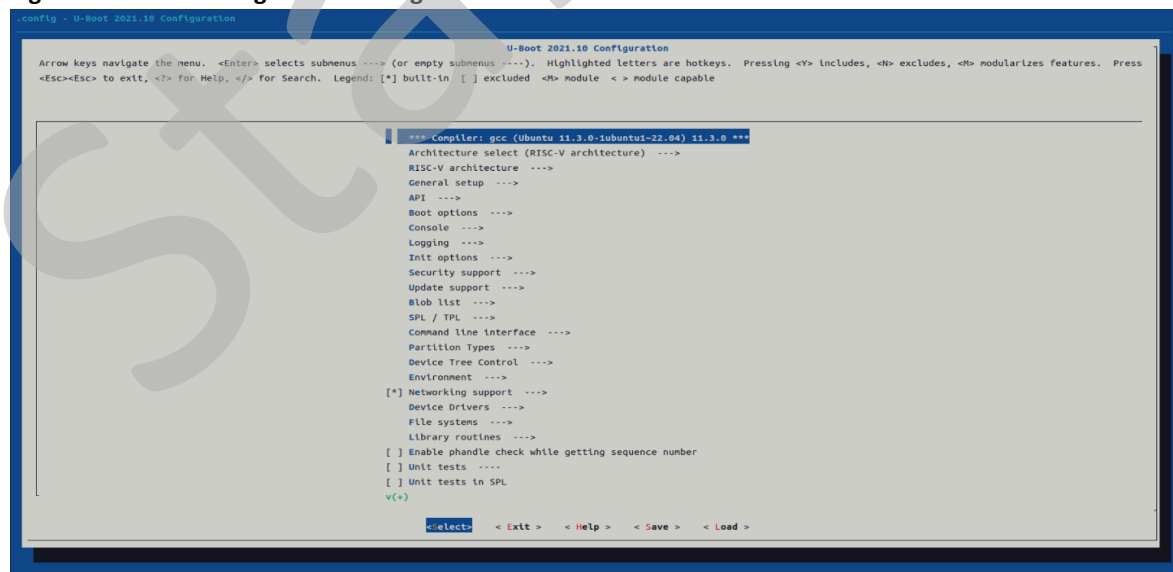
Follow the steps below to enable the kernel menu configuration for U-Boot.

1. Enter the root directory of the SDK: `freelight-u-sdk`.
2. Run the following command to build the kernel menu configuration dialog.

```
make uboot-menuconfig
```

Result: The following menu configuration dialog is generated.

Figure 2-2 Menu Configuration Dialog



3. If you need to enable a function, navigate the menu and select your target option in correspondence with the function.
4. Save your change before you exit the kernel configuration dialog.
5. Run the `make` command to generate the bin file.

**Note:**

You can use `menuconfig` to view and edit any a macro definition in the deconfiguration file as described in [Kernel Menu Deconfiguration \(on page 9\)](#).

1. Enter the kernel menu configuration dialog.
2. Type “/” to enter the search screen.
3. Input the keyword of the macro definition which you want to search.
4. Edit the macro definition following your need.

2.3. U-Boot DTS

Since Linux version 5.4, U-Boot no longer uses **sysconfig** and kernel DTS as configuration files, whereas it uses a standalone DTS file to contain the configuration parameters.

Location

The U-boot DTS file is in the following path:

```
freelight-u-sdk/u-boot/arch/riscv/dts/jh7110-u-boot.dtsi
```

**Note:**

Make sure you have included the following file in the U-Boot file path.

- For VisionFive 2: `starfive_visionfive-u-boot.dtsi`

Deconfiguration Options

Make sure you reference the correct file in the configuration settings.

The following table lists an example of the configuration options.

Table 2-1 Deconfiguration Options

Configuration Item	Option
CONFIG_DEFAULT_DEVICE_TREE	jh7110.dtsi
CONFTG_TARGET_STARFIVE_EVB	jh7110-evb.dts
CONFTG_TARGET_STARFIVE_VISIONFIVE	jh7110-visionfive-v2.dts

3. Interface Description

3.1. FDT Interfaces

The *Flattened Device Tree (FDT)* interfaces are used to include part of the device information structure in the device tree file.

3.1.1. fdt_getprop

The interface has the following parameters.

- **Synopsis:**

```
const void *fdt_getprop(const void *fdt, int nodeoffset, const char *name, int *lenp)
```

- **Description:** The interface is used to retrieve the value of a given property.

- **Parameter:**

- **fdt:** The pointer to the DTB file.
- **nodeoffset:** The offset of the node to get property from.
- **name:** The name of the property.
- **lenp:** The pointer to an integer variable (will be overwritten) or NULL.

- **Return:**

- **Success:** The pointer to the property's value.

If **lenp** is not NULL, ***lenp** will contain the length of the property value (≥ 0).

- **Failure:** NULL.

If **lenp** is not NULL, ***lenp** will contain an error code (< 0):

- **FDT_ERR_NOTFOUND:** The target property is not found.
- **FDT_ERR_BADOFFSET:** The node offset is invalid. The node offset does not point to the **FDT_BEGIN_NODE** tag.
- **FDT_ERR_BADMAGIC:** The magic field is invalid.
- **FDT_ERR_BADVERSION:** The version is invalid.
- **FDT_ERR_BADSTATE:** The state is invalid.
- **FDT_ERR_BADSTRUCTURE:** The file structure is invalid.
- **FDT_ERR_TRUNCATED:** standard meanings.

3.1.2. fdt_set_node_status

The interface has the following parameters.

- **Synopsis:**

```
int fdt_set_node_status(void *fdt, int nodeoffset, enum fdt_status status, unsigned int error_code)
```

- **Description:** The interface is used to set the status of a node.

- **Parameter:**

- **fdt:** The pointer to the DTB file.
- **status:** The following statuses are available.

- FDT_STATUS_OKAY -
- FDT_STATUS_DISABLED -
- FDT_STATUS_FAIL -
- FDT_STATUS_FAIL_ERROR_CODE -
- **error_code**: optional, only used if status is FDT_STATUS_FAIL_ERROR_CODE
- **Return:**
 - **Success**: 0.
 - **Failure**: Any value other than 0.

3.1.3. fdt_path_offset

The interface has the following parameters.

- **Synopsis:**

```
int fdt_path_offset(const void *fdt, const char *path);
```

- **Description:** The interface is used to find a tree node by its full path.

- **Parameter:**

- **fdt**: The pointer to the DTB file.
- **path**: The full path to the node.
- **error_code**: optional, only used if status is FDT_STATUS_FAIL_ERROR_CODE.

- **Return:**

- **Success**: The structure block offset of the node with the requested path (≥ 0).
- **Failure**: Any value < 0 .

3.2. ENV Interfaces

The environment (ENV) interfaces are used to provide U-Boot with the capabilities of dynamic configuration during program operation. U-Boot defines a set of default environment variables in file `include/env_default.h`.

3.2.1. int env_set

The interface has the following parameters.

- **Synopsis:**

```
int env_set(const char *varname, const char *value);
```

- **Description:** The interface is used to set an environment variable.

- **Parameter:**

- **varname**: The environment variable you want to change.
- **varvalue**: The new value of the environment variable.

- **Return:**

- **Success**: 0.
- **Failure**: 1.

3.2.2. env_get

The interface has the following parameters.

- **Synopsis:**

```
char *env_get(const char *varname);
```

- **Description:** The interface is used to get the value of an environment variable.

- **Parameter:**

- **varname:** The environment variable you want to get value from.

- **Return:**

- **Success:** Value of the target environment variable, or "NULL" if the environment variable is not found.
 - **Failure:** Any other values.

3.2.3. env_save

The interface has the following parameters.

- **Synopsis:**

```
int env_save(void);
```

- **Description:** The interface is used to save the environment variables to storage.

- **Parameter:** None.

- **Return:**

- **Success:** 0.
 - **Failure:** Any value other than 0.

3.3. U-Boot Commands

The U-Boot command line interfaces are used to create a minimum valid command line interface. So, after the users have started up U-Boot, they can have access to a lot of command lines, including `tftp`, `bootm`, and `booti`, etc.

3.3.1. run_command_list

The interface has the following parameters.

- **Synopsis:**

```
int run_command_list(const char *cmd, int len, int flag)
```

- **Description:** The interface is used to execute the U-Boot command line.

- **Parameter:**

- **cmd:** The pointer to the command.
 - **len:** The length of the command line. To load the length automatically, set this value to "-1".
 - **flag:** Not used.

- **Return:**

- **Success:** 0.
 - **Failure:** Any value other than 0.

3.3.2. do_bootm

The interface has the following parameters.

- **Synopsis:**

```
int do_bootm(struct cmd_tbl *cmdtp, int flag, int argc, char *const argv[]);
```

- **Description:** The interface is used to run the `bootm` command, and boot the application from the image in memory.

- **Parameter:**

- **cmdtp:** The command information for the `bootm` command.
- **flag:** The command flags.
- **argc:** The number of arguments.
- **argv:** The list of arguments.

- **Return:**

- **Success:** 0.
- **Failure:** Any value other than 0.

3.3.3. do_booti

The interface has the following parameters.

- **Synopsis:**

```
int do_booti(struct cmd_tbl *cmdtp, int flag, int argc, char *const argv[]);
```

- **Description:** The interface is used to run the `booti` command, and boot the Linux kernel "Image" format from memory.

- **Parameter:**

- **cmdtp:** The command information for the `booti` command.
- **flag:** The command flags.
- **argc:** The number of arguments.
- **argv:** The list of arguments.

- **Return:**

- **Success:** 0.
- **Failure:** Any value other than 0.

3.3.4. do_tftpboot

The interface has the following parameters.

- **Synopsis:**

```
int do_tftpboot(struct cmd_tbl *cmdtp, int flag, int argc, char *const argv[]);
```

- **Description:** The interface is used to run the `tftpboot` command, and boot the image via network using TFTP protocol.

- **Parameter:**

- **cmdtp:** The command information for the `tftpboot` command.
- **flag:** The command flags.
- **argc:** The number of arguments.
- **argv:** The list of arguments.

- **Return:**

- **Success:** 0.
- **Failure:** 1.

3.3.5. do_mmcinfo

The interface has the following parameters.

- **Synopsis:**

```
static int do_mmcinfo(struct cmd_tbl *cmdtp, int flag, int argc, char *const argv[])
```

- **Description:** The interface is used to run the `mmcinfo` command, and display the information of the current MMC device.

- **Parameter:**

- **cmdtp:** The command information for the `mmcinfo` command.
- **flag:** The command flags.
- **argc:** The number of arguments.
- **argv:** The list of arguments.

- **Return:**

- **Success:** 0.
- **Failure:** 1.

3.4. Flash Read and Write

The flash read and write interfaces are used to read data from and write data to the SPI flash. This can be used to flash images to the SPI flash via serial interfaces under U-Boot mode.

3.4.1. spi_flash_read

The interface has the following parameters.

- **Synopsis:**

```
static inline int spi_flash_read(struct spi_flash *flash, u32 offset, size_t len, void *buf)
```

- **Description:** The interface is used to read data from the SPI flash.

- **Parameter:**

- **flash:** The SPI flash device.
- **offset:** The address offset of the above device in bytes.
- **len:** The length of bytes to read.
- **buf:** The buffer to store the read data.

- **Return:**

- **Success:** 0.
- **Failure:** Any value other than 0.

3.4.2. spi_flash_write

The interface has the following parameters.

- **Synopsis:**

```
static inline int spi_flash_write(struct spi_flash *flash, u32 offset, size_t len, const void *buf)
```


- **Description:** The interface is used to write data into the SPI flash.
- **Parameter:**
 - **flash:** The SPI flash device.
 - **offset:** The address offset of the above device in bytes.
 - **len:** The length of bytes to write.
 - **buf:** The buffer to contain the data to write.
- **Return:**
 - **Success:** 0.
 - **Failure:** Any value other than 0.

3.5. Partition Information

The partition information interface is used to find a partition by partition name in all registered partitions and get its partition information.

3.5.1. part_get_info_by_name

The interface has the following parameters.

- **Synopsis:**

```
int part_get_info_by_name(struct blk_desc *dev_desc, const char *name, struct disk_partition *info);
```

- **Description:** The interface is used to find a partition by partition name in all registered partitions and get its partition information.
- **Parameter:**
 - **dev_desc:** The device descriptor.
 - **name:** The name of the specific partition table entry.
 - **info:** The partition information to query.
- **Return:**
 - **Success:** The level of the target GPIO interface: 0 as low level, 1 as high level.
 - **Failure:** -1.

3.6. GPIO Operations

The GPIO operation interfaces are used to get and set GPIO levels to read the input and output status and alter the GPIO from an output to an output.

3.6.1. gpio_get_value

The interface has the following parameters.

- **Synopsis:**

```
int gpio_get_value(unsigned gpio)
```

- **Description:** The interface is used to get the level from a target GPIO interface. The level is used to define whether the GPIO is an input or an output.
- **Parameter:**
 - **gpio:** The GPIO index number.

- **Return:**

- **Success:** The level of the target GPIO interface: 0 as low level, 1 as high level.
- **Failure:** -1.

3.6.2. gpio_set_value

The interface has the following parameters.

- **Synopsis:**

```
void gpio_set_value(unsigned gpio, int value)
```

- **Description:** The function is used to set the level to a target GPIO interface. The level can be used to alter the GPIO from an output to an output.



Note:

The GPIO has to be an output, otherwise, this function may have no effect.

- **Parameter:**

- **gpio:** The GPIO index number.
- **value:** The expected level: 0 as low level, 1 as high level.

- **Return:** None.

4. Debug Methods

U-Boot stores its debug settings in the following file: `freelight-u-sdk/u-boot/driver/core/Kconfig`.

In this file, you can find the settings for the following debug methods:

- [DM_DEBUG \(on page 19\)](#)
- [DEBUG_DEVRES \(on page 19\)](#)
- [TOOLS_DEBUG \(on page 19\)](#)



Tip:

All the debug settings can be found in the deconfiguration dialog. See [Kernel Menu Deconfiguration \(on page 9\)](#) for more information.



Tip:

All the debug settings can be enabled in the kernel configuration menu. See [Kernel Menu Configuration \(on page 10\)](#) for more information.

4.1. DM_DEBUG

Driver model (DM) is a set of unified methods for driver definition and message interfacing. DM provides a set of standard driver-device development models which are very similar to the driver-device models of the Linux kernel.

You can configure the **DM_DEBUG** option to enable debug messages in the driver model core.

4.2. DEBUG_DEVRES

You can configure the **DEBUG_DEVRES** option to manage the debug functions for device resources. If this option is enabled, the system will print all the DEVRES debug messages. Also, you can use the `dm devres` command to dump the list of device resources for each devices.

Make sure to select this option if you are having a problem with DEVRES or want to debug the resource management for a managed device.

4.3. TOOLS_DEBUG

You can configure the **TOOLS_DEBUG** option to enable debug information on tools. For example, you can enable the generation of debug information on tools including `mkimage`.

The generated debug information are used for debug purposes only. With the help of these debug information, it is possible to set breakpoints on a particular line, perform single-step debug through the source code, etc.