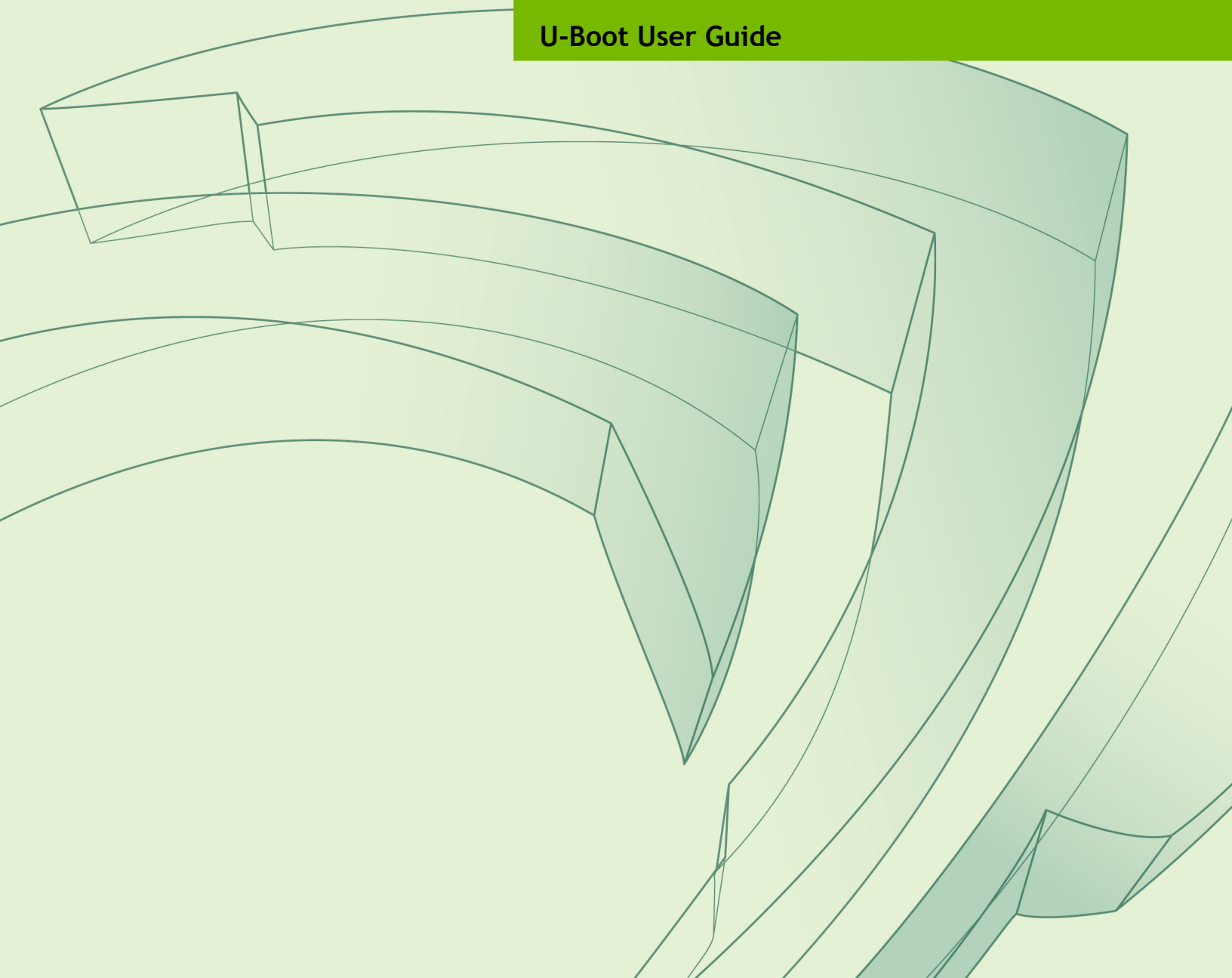
NVIDIA TEGRA LINUX DRIVER PACKAGE

DA_07298-001_01 | November 4, 2014
Advance Information | Subject to Change

U-Boot User Guide



DOCUMENT CHANGE HISTORY

DA_07298-001_01

Version	Date	Authors	Description of Change
v1.0	3 June 2014	hlang/mzensius	Initial release for R19.3.
v2.0	4 Nov 2014	ehuang/mzensius	Updated for the R21.1 release.

TABLE OF CONTENTS

OVERVIEW

This guide describes the U-Boot implementation for NVIDIA® Tegra® Linux Driver Package.

Note: In the R21.1 release, the default boot loader has changed to U-Boot from fastboot. Check that your environment is fully updated for this change in boot loader before compiling and flashing the boot loader and the kernel.

REQUIREMENTS

The software requirements and prerequisites, including Linux tools that are required, for Tegra Linux Driver Package (L4T) include:

- Linux-based Host System

  Functionality of the U-Boot build and flashing utilities was validated using an Ubuntu 12.04 host system; however, later versions or alternative Linux distributions may work with host-specific modifications.

- Tegra Linux Driver Package (L4T)

  Download the latest L4T package from the Tegra Developer Zone and follow the installation instructions in the user documentation. You can find L4T on the Tegra Developer Zone:

  ```
  http://developer.nvidia.com/linux-tegra
  ```

- Flex and Bison

  The U-Boot makefiles require flex and bison to parse various configuration files. If flex and bison are not already installed on your host machine, you can install them on an Ubuntu host machine with the following command:

```
$ sudo apt-get install flex bison
```

- Device Tree Compiler (dtc)

  Ensure that the full path to the dtc binary is available to the U-Boot make system by either passing the path as a variable or by making the dtc directory available in the local command path of the host machine. Most of the dtc packages available from standard Linux distribution package management systems (like apt) are not yet updated with a version of dtc with the features required by the U-Boot makefile. Therefore, an example of building dtc from source is included in this section. For the procedure, see Using the Device Tree Compiler.

  A pre-built DTC compiler is also included in the kernel directory of the release. This DTC compiler is built from the kernel sources in this release. These sources are located in the scripts/dtc directory and built by building the kernel dtbs target.

- ARM tool chain for cross compilation

  For more information, see the Toolchain section in the *Tegra Linux Driver Package Development Guide*.

- U-Boot source

  For more information, see Downloading and Building U-Boot.

- Kernel source

  For information, see the following topics in the *Linux Driver Package Development Guide* Getting Started chapter:

  - Setting Up Your Environment
  - Synchronizing the Kernel Sources
  - Building the NVIDIA Kernel

Also, see the Adding a Compiled Kernel to the Root File System.

FLASHING U-BOOT

This section describes flashing U-Boot in internal eMMC to fetch the rootfs from internal eMMC, an SD card, USB storage device, or IP network.

Note: The boot device where you flash U-Boot is the internal eMMC, even though root device (where the boot script and kernel are fetched and where the rootfs is present) can be eMMC, a SD card, USB device, or IP network.

To flash U-Boot and mount rootfs from internal eMMC

- To flash U-Boot to fetch boot script and kernel, and to mount rootfs from internal eMMC, execute the following command:

```
$ sudo ./flash.sh <target_board> mmcblk0p1
```

Where <target_board> is “jetson-tk1” for Jetson TK1.

Note: In the R21.1 release, the default boot loader has changed to U-Boot from fastboot. Check that your environment is fully updated for this change in boot loader before compiling and flashing the boot loader and the kernel.

To flash U-Boot and mount rootfs from an SD card

- To flash U-Boot to fetch boot script and kernel, and to mount rootfs from an SD card, execute the following command:

```
$ sudo ./flash.sh <target_board> mmcblk1p1
```

Where <target_board> is “jetson-tk1” for Jetson TK1.

To flash U-Boot and to mount rootfs from a USB storage device

- To flash U-Boot to fetch boot script and kernel, and to mount rootfs from an USB storage device such as a Pen Drive or other USB device, execute the following command:

```
$ sudo ./flash.sh <target_board> sda1
```

Where <target_board> is “jetson-tk1” for Jetson TK1.

Note: The U-Boot boot loader only detects USB external storage.The kernel detects both USB external storage and external SCSI_SATA storage.

Use only 1 external USB storage device at a time. If more than 1 external device is used, a random device may be chosen as root device.

To flash U-Boot and mount rootfs from an IP network

- To flash U-Boot to fetch boot script and kernel from internal eMMC, and to mount rootfs from an IP network, execute the following command:

```
$ sudo ./flash.sh –N <IPA>:/<nfs directory> [-n <target IPA>:<host
IPA>:<gateway IPA>:<netmask>] <target_board> <interface name>
```

Where:

- <target_board> is “jetson-tk1” for Jetson TK1.

- <interface name> is “eth0” for RJ45 connector and “eth1” for USB Ethernet dongle.
- <IPA> is the NFS server hosting the rootfs.
- <nfs_directory> is the full path name of exported rootfs.
- <target IPA> is the static IP address for the target device.
- <host IPA> is the static IP address for the NFS server.
- <gateway IPA> is the static IP address for the gateway.
- <netmask> is the static netmask for the local network.

Note: The -n option is only recommended on point-to-point network connections where no DHCP server is configured. In a normal lab/office network environment where a DHCP server is configured, do not use the -n option.

CHANGING EMMC PARTITION LAYOUT

Based on eMMC hardware and software layout information in the following files:

- <target_board>.conf
- <top>/Linux_for_Tegra/bootloader/<target_board>/cfg/gnu_linux_fastboot_emmc_full.cfg

Where <top> is your L4T root, flash.sh generates the internal eMMC partition layout. When you use the NvFlash utility and the fastboot.bin flash application, U-Boot shares the same internal eMMC partition layout as fastboot. The only difference is that L4T U-Boot does not use the kernel partition.

eMMC IC Parameter

The eMMC IC parameter is defined by 2 variables in the <target device>.conf file. These 2 parameters limit the size of the total usable data area and determines the location of GPT partitions.

- The BOOTPARTSIZE parameter specifies that the eMMC boot partition size (boot0 partition size + boot1 partition size.)
- The EMMCSIZE parameter specifies that the eMMC usable data size (BOOTPARTSIZE + user partition size.)

Note: boot0, boot1, and user partition size can be obtained from the eMMC device data sheet.

RootFS Size

Among many partitions, the rootfs partition is the largest and the size of the rootfs partition is one of the most important factors in partition layout determination. By default, flash.sh sets the rootfs size at 14 GB; you can change this by modifying the value of ROOTFSSIZE variable in the <target device>.conf file.

Note: The sum of space used by all partitions cannot exceed EMMCSIZE.

GPT Partitions

Based on the internal eMMC partition layout, the `flash.sh` script creates the primary and secondary GPT partitions automatically. The Protective MBR contains device information to prevent traditional boot loaders from performing destructive activities. The primary GPT partition contains the GUID Partition Table. The secondary GPT partition contains the same information as the primary GPT and is used as the backup. The Protective MBR is located at LBA 0, the primary GPT is located at LBA 1, and the secondary GPT is located at the last LBA of the boot device. The last Logical Block Address (LBA) varies from device to device. Both U-Boot and the kernel are able to obtain the last LBA.

LNX Partition

Normally, the LNX partition is not used by U-Boot; however, for compatibility, an empty LNX partition is allocated.

APP Partition

This partition is where rootfs is flashed for the internal eMMC rootfs use case. Since U-Boot expects boot script, kernel, and DTB files in the `<rootfs>/boot` directory, for the rootfs-on internal-eMMC configuration, `flash.sh` flashes the following kernel files in the APP partition:

- kernel (`zImage`)
- device_tree_blob (`tegra124-jetson_tk1-pm375-000-c00-00.dtb`)
- `sysboot_config` (`extlinux.conf`)

Note: flash.sh treats the rootfs-on-IP-network configuration as a special case and flashes kernel files listed above in the `<APP partition>:/boot` directory.

Example Full Internal eMMC Partition Layout

This section provides example eMMC layout configuration (CFG) file contents. For the actual configuration used in the release, see the `gnu_linux_fastboot_emmc_full.cfg` file.

Note: The kernel partition (LNX) is not used by U-Boot by default.

```
 [device]
type=sdmmc
instance=3

[partition]
```

```
name=BCT
id=2
type=boot_config_table
allocation_policy=sequential
filesystem_type=basic
size=2097152            #BCTSIZE
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0

[partition]
name=PPT
id=3
type=data
allocation_policy=sequential
filesystem_type=basic
size=8388608            #PPTSIZE
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=ppt.img

[partition]
name=PT
id=4
type=partition_table
allocation_policy=sequential
filesystem_type=basic
size=2097152
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0

[partition]
name=EBT
id=5
type=bootloader
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
filename=fastboot.bin

[partition]
name=LNX
id=6
```

```
type=data
allocation_policy=sequential
filesystem_type=basic
size=16777216
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
filename=boot.img

[partition]
name=SOS
id=7
type=data
allocation_policy=sequential
filesystem_type=basic
size=6291456
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=recovery.img

[partition]
name=NVC
id=8
type=data          #TEGRABOOT
allocation_policy=sequential
filesystem_type=basic
size=2097152
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=nvtboot.bin

[partition]
name=MPB
id=9
type=data          #MTSPREBOOT
allocation_policy=sequential
filesystem_type=basic
size=6291456
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=mts_preboot_si

[partition]
name=MBP
id=10
```

```
type=data          #MTSBOOTPACK
allocation_policy=sequential
filesystem_type=basic
size=6291456
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=mts_si

[partition]
name=GP1
id=11
type=GP1
allocation_policy=sequential
filesystem_type=basic
size=2097152
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0

[partition]
name=APP
id=12
type=data
allocation_policy=sequential
filesystem_type=basic
size=1073741824
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
filename=system.img

[partition]
name=DTB
id=13
type=data
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=tegra.dtb

[partition]
name=EFI
id=14
type=data
```

```
allocation_policy=sequential
filesystem_type=basic
size=67108864           #EFISIZE
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=efi.img

[partition]
name=USP
id=15
type=data
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0

[partition]
name=TP1
id=16
type=data
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0

[partition]
name=TP2
id=17
type=data
allocation_policy=sequential
filesystem_type=basic
size=4194304
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0

[partition]
name=TP3
id=18
type=data
allocation_policy=sequential
filesystem_type=basic
size=4194304
```

```
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0

[partition]
name=WB0
id=19
type=data   #WB0BOOT
allocation_policy=sequential
filesystem_type=basic
size=2097152
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=nvtbootwb0.bin

[partition]
name=UDA
id=20
type=data
allocation_policy=sequential
filesystem_type=basic
size=2097152
file_system_attribute=0
partition_attribute=0
allocation_attribute=0x808
percent_reserved=0

[partition]
name=GPT
id=21
type=GPT
allocation_policy=sequential
filesystem_type=basic
size=0xFFFFFFFFFFFFFFFF
file_system_attribute=0
partition_attribute=0
allocation_attribute=8
percent_reserved=0
#filename=spt.img
```

ROOTFS TESTED BY DEVICE

This section provides expected results when testing the root file system location by device. Y indicates that correct U-Boot initialization and hand-off to the kernel occurred. However, this alone does not guarantee a fully-functional system.

RootFS Location	Jetson TK1

mmcblk0p1	Y
mmcblk1p1	Y
sda1	Y
eth0	Y
eth1	Y

USING THE DEVICE TREE COMPILER

Use this example to build the Device Tree Compiler (dtc) from the source to include the features required by the U-Boot makefile.

To build dtc from source

1. If you do not want to pass in dtc as a parameter to the U-Boot environment, ensure a local command path (such as `./usr/local/bin` or another choice) is at the beginning of the shell command path.

   ```
   $ export PATH=<local_command_path>:${PATH}
   ```

   If you execute:

   ```
   $ make install
   ```

   The dtc makefile installs the binary into the first entry of shell PATH variable. Therefore, it is important that the local command path is at the beginning of the shell PATH variable.

2. Create a directory to contain the dtc source code and change directories into it:

   ```
   $ mkdir -p <dtc_src_dir>
   $ cd <dtc_src_dir>
   ```

3. Download dtc source code by executing the following git clone command:

   ```
   $ git clone git://git.kernel.org/pub/scm/utils/dtc/dtc.git
   ```

4. Build and optionally install dtc by executing:

```
$ cd <dtc_src_dir>/dtc
$ make
```

Or, alternatively, if you want it installed on your local host file system execute:

```
$ make install
```

5. **I**f you specified just make be sure to pass in the following to the U-Boot make system:

```
DTC=${PATH_TO_DTC_TOOL_BINARY}
```

Where PATH_TO_DTC_TOOL_BINARY is the location of the dtc binary, such as:

```
<dtc_src_dir>/dtc/dtc
```

DOWNLOADING AND BUILDING U-BOOT

Follow these procedures to download and build U-Boot to use as the boot loader for your Tegra device.

PREREQUISITE

- Before copying U-Boot, backup the original `u-boot.bin` file in:

```
<your_L4T_root>/Linux_for_Tegra/bootloader/<platform>/u-boot.bin
```

  Where `<platform>` is the Tegra hardware platform, such as ardbeg.

To download and build U-Boot

1. Download the L4T U-Boot source code by executing the following commands:

```
$ mkdir -p <uboot_src_dir>
$ cd <uboot_src_dir>
$ git clone -n git://nv-tegra.nvidia.com/3rdparty/u-boot.git
```

   Alternatively, you can use the source_sync.sh script that is provided in the L4T release and skip the next step.

   When running source_sync.sh -u, if no parameters are provided, the script prompts for the <TAG_NAME>, which is provided in the release notes. Also, you can run the script by passing the <TAG_NAME> in as follows:

```
$ cd <your_L4T_root>/Linux_for_Tegra
$ ./source_sync.sh -u <TAG_NAME>
```

   This will sync the source to `<source_sync.sh_location>/sources/u-boot_source`. The `<uboot_src_dir>` directory becomes

<your_L4T_root>/Linux_for_Tegra/sources/u-boot_source. Use that path to build U-Boot in the steps below.

2. Check out the git tag name:

```
$ cd u-boot
$ git checkout –b <branch_name> <tag_name>
```

Where <branch_name> is the name of your local branch and <tag_name> is the release tag name provided in the Release Notes.

3. Set the build environment:

```
$ export ARCH=arm
$ export CROSS_COMPILE=<your_toolchain_location>
$ export DTC=<dtc_binary_location>
```

4. Build U-Boot by executing:

```
$ cd <uboot_src_dir>/u-boot
$ make distclean
$ make <target_board>_defconfig
$ make
```

Where <target_board> is the device, such as code-name jetson-tk1 for Jetson TK1.

UPDATING AND RUNNING NEWLY BUILT U-BOOT

This section provides procedures for flashing and running the newly built U-Boot as a boot loader for the Tegra device.

To copy U-Boot for flashing to the device

- To copy U-Boot for flashing to the device, execute the following:

```
$ cp <uboot_src_dir>/u-boot/u-boot-dtb-tegra.bin
<your_L4T_root>/Linux_for_Tegra/bootloader/<target_board>/u-boot.bin
```

Once the newly built U-Boot is copied, you can flash the Tegra device with either the new U-Boot only or the full L4T image.

To flash new U-Boot only

- To flash the new U-Boot only, execute the following:

```
$  sudo ./flash.sh -k EBT <target_board> mmcblk0p1
```

  Where <target_board> is “jetson-tk1” for Jetson TK1.

To flash the full L4T image

To flash the entire device again, see Flashing U-Boot in this document.

ADDING A COMPILED KERNEL TO THE ROOT FILE SYSTEM

Follow these procedures to create and install the kernel image required by U-Boot into the sample file system.

PREREQUISITE

- You have compiled the kernel as described in the *Linux Driver Package Development Guide Getting Started* chapter.

To configure a file system for U-Boot

1. Use the `apply_binaries` script to copy the `zImage` in the kernel directory into the `rootfs` directory in the `/boot` folder.
2. Install the `rootfs` directory onto your device.

   For U-Boot to function properly, there must be a `zImage` and dtb files in the `/boot` directory of the target file system.

   For more information on installing the `rootfs` directory onto your device, see the *Setting Up the Root File System* topic in the *Linux Driver Package Development Guide Getting Started* chapter.

3. If you have already installed your `rootfs` onto a device, manually copy the `zImage` and dtb files to the previously installed root file system.

To configure a file system installed in the internal eMMC

1. Optionally, backup the existing release kernel and dtb files to avoid overwriting.

2. Copy the compiled `zImage` and dtb files over the current L4T release kernel directory by executing the following commands:

```
$ cp arch/arm/boot/zImage <L4T_path>/Linux_for_Tegra/kernel
$ cp arch/arm/boot/dts/tegra124-jetson_tk1-pm375-000-c00-00.dtb
<L4T_path>/Linux_for_Tegra/kernel
```

`flash.sh` automatically copies `zImage` and dtb files to the internal eMMC rootfs.

Adding a new Kernel

This topic provides the commands to use to replace the kernel after U-Boot has been flashed as the default boot loader.

To replace the kernel in systems that boot from internal eMMC

1. Boot the Jetson TK1 system and log in.
2. Copy the new kernel files (using scp) into the /boot directory.
3. Re-boot the Jetson TK1 system.

To replace the kernel in systems that boot from an SD Card or USB Pen Drive

1. Connect the SD Card or USB Pen Drive to your host system.
2. Copy the new kernel files to a /boot directory on the SD Card or USB Pen Drive.
3. Disconnect the SD Card or USB Pen Drive from the host system.
4. Connect the SD Card or USB Pen Drive to the Jetson TK1 system.
5. Re-boot the Jetson TK1 system.

To replace the kernel in systems that boot from an IP network

1. Boot the Jetson TK1 system and log in.
2. On the target system enter the following command:

```
$ sudo mount /dev/mmcblk0p1 /mnt
```

3. Copy the new kernel files (using scp) to the mnt/boot directory.
4. Re-boot the Jetson TK1 system.

EXAMPLE SYSBOOT CONFIGURATION FILES

The U-Boot functionality includes a default booting scan sequence. It scans bootable devices in the following order:

- External SD Card
- Internal eMMC
- USB Device
- NFS Device

It looks for an `extlinux.conf` configuration file located in the following directory of the bootable device:

```
<rootfs>/boot/extlinux
```

If U-Boot finds the `extlinux.conf` file, it:

- Uses the `sysboot` command to read out boot configuration from `extlinux.conf`
- Loads kernel `zImage` file and device tree file
- Boots the kernel.

The `zImage` and device tree files are all user accessible in the `<rootfs>/boot` location after booting. The `extlinux.conf` file is user accessible in the `<rootfs>/boot/extlinux` location. Users can easily change these files to test their own kernel without flashing.

The file `extlinux.conf` is a standard text-format sysboot configuration file that contains all boot information. It indicates the U-Boot kernel image filename, the device tree blob filename, and the kernel boot command line. There are four example `extlinux.conf` files provided in the L4T release:

```
<target_board>_extlinux.conf.emmc
<target_board>_extlinux.conf.sdcard
```

```
<target_board>_extlinux.conf.usb
<target_board>_extlinux.conf.nfs
```

During flashing, `flash.sh` copies the appropriate variant to the following location:

```
<rootfs>/boot/extlinux/extlinux.conf
```

The `extlinux.conf` files are very similar except for different kernel boot command lines. The `extlinux.conf` files are in the following location:

```
bootloader/<platform>/
```

Where `<platform>` is `ardbeg` for Jetson TK1.

EMMC SYSBOOT EXTLINUX.CONF FILE

The following shows the contents of the `extlinux.conf` file.

```
TIMEOUT 30
DEFAULT primary

MENU TITLE Jetson-TK1 eMMC boot option

LABEL primary
      MENU LABEL primary kernel
      LINUX zImage
      FDT tegra124-pm375.dtb
      APPEND console=ttyS0,115200n8 console=tty1 no_console_suspend=1
lp0_vec=2064@0xf46ff000 video=tegrafb mem=1862M@2048M memtype=255
ddr_die=2048M@2048M section=256M pmuboard=0x0177:0x0000:0x02:0x43:0x00
vpr=151M@3945M tsec=32M@3913M otf_key=c75e5bb91eb3bd947560357b64422f85
usbcore.old_scheme_first=1 core_edp_mv=1150 core_edp_ma=4000
tegraid=40.1.1.0.0 debug_uartport=lsport,3 power_supply=Adapter
audio_codec=rt5640 modem_id=0 android.kerneltype=normal
usb_port_owner_info=0 fbcon=map:1 commchip_id=0 usb_port_owner_info=0
lane_owner_info=6 emc_max_dvfs=0 touch_id=0@0
tegra_fbmem=32899072@0xad012000
board_info=0x0177:0x0000:0x02:0x43:0x00 root=/dev/mmcblk0p1 rw
rootwait tegraboot=sdmmc gpt
```

Different boot methods have different APPEND strings in the `extlinux.conf` file. Check each file for details.

Note: NFS root uses eMMC as boot device. The <rootfs>/boot directory is flashed into eMMC, but the kernel mounts the NFS device as rootfs.

DEBUGGING THE U-BOOT ENVIRONMENT

Use these debugging tips to help you debug your U-Boot environment. These examples do not represent a comprehensive listing of U-Boot functionality. For a full list of supported commands and their usage by U-Boot, consult U-Boot documentation and source.

When creating your own kernel, U-Boot sometimes has trouble finding it. To eliminate this issue, use the commands in these examples to verify that U-Boot can read the device and can see the files in the system. If a boot device is not found, or the device has trouble booting with a kernel other than the reference kernel provided in the L4T release, review these examples for debugging purposes.

INTERRUPTING U-BOOT

You can interrupt U-Boot during boot.

To interrupt U-Boot

- Press any key during boot.

GETTING HELP

On the U-Boot terminal screen, type help at any time for the list of supported commands from the U-Boot terminal.

To see the U-Boot Help text

- To see the U-Boot help text enter the following command:

```
# help
```

The following example Help information is printed when executing help on a Jetson TK1 device.

```
?       - alias for 'help'
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
bootz   - boot Linux zImage image from memory
cmp     - memory compare
coninfo - print console devices and information
cp      - memory copy
crc32   - checksum calculation
dfu     - Device Firmware Upgrade
dhcp    - boot image via network using DHCP/TFTP protocol
dm      - Driver model low level access
echo    - echo args to console
editenv - edit environment variable
enterrcm- reset Tegra and enter USB Recovery Mode
env     - environment handling commands
exit    - exit script
ext2load- load binary file from a Ext2 filesystem
ext2ls  - list files in a directory (default /)
ext4load- load binary file from a Ext4 filesystem
ext4ls  - list files in a directory (default /)
false   - do nothing, unsuccessfully
fatinfo - print information about filesystem
fatload - load binary file from a dos filesystem
fatls   - list files in a directory (default /)
fdt     - flattened device tree utility commands
go      - start application at address 'addr'
gpio    - query and control gpio pins
help    - print command description/usage
i2c     - I2C sub-system
imxtract- extract a part of a multi-image
itest   - return true/false on integer compare
load    - load binary file from a filesystem
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loadx   - load binary file over serial line (xmodem mode)
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
ls      - list files in a directory (default /)
md      - memory display
mii     - MII utility commands
mm      - memory modify (auto-incrementing address)
mmc     - MMC sub system
```

```
mmcinfo - display MMC info
mw      - memory write (fill)
nm      - memory modify (constant address)
part    - disk partition related commands
pci     - list and access PCI Configuration Space
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
pxe     - commands to get and boot from pxe files
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sf      - SPI flash sub-system
showvar - print local hushshell variables
size    - determine a file’s size
sleep   - delay execution for some time
source  - run script from memory
sspi    - SPI utility command
sysboot - command to get and boot from syslinux files
test    - minimal test like /bin/sh
tftpboot- boot image via network using TFTP protocol
true    - do nothing, successfully
ums     - Use the UMS [User Mass Storage]
usb     - USB sub-system
usbboot - boot from USB device
version - print monitor, compiler and linker version
```

LISTING A DIRECTORY STRUCTURE

You can list the directory structure of a particular device. For example, to list the directory structure of `sda1` in U-Boot by type: `mmc 0:1` (for eMMC device 0 partition 1).

To list the directory structure

- To list the directory structure enter the following command:

```
# ext2ls mmc 0:1
```

This also functions correctly on EXT3/EXT4 file systems.

Example output follows:

```
<DIR>       4096 .
<DIR>       4096 ..
<DIR>       4096 bin
<DIR>       4096 boot
<DIR>       4096 dev
<DIR>       4096 etc
```

```
<DIR>       4096 home
<DIR>       4096 lib
<DIR>       4096 lost+found
<DIR>       4096 media
<DIR>       4096 mnt
<DIR>       4096 opt
<DIR>       4096 proc
<DIR>       4096 root
<DIR>       4096 sbin
<DIR>       4096 selinux
<DIR>       4096 srv
<DIR>       4096 sys
<DIR>       4096 tmp
<DIR>       4096 usr
<DIR>       4096 var
```

LISTING THE CONTENTS OF A DIRECTORY

You can list the contents of any directory.

To list the contents of a directory

- List directory contents with the following command:

```
# ext2ls mmc 0:1 <directory>
```

Where `<directory>` is an expected path on the device.

For example, to list contents of the `/boot` directory where the `zImage` file should be (as shown in the example output below), use the following command:

```
# ext2ls mmc 0:1 /boot
<DIR>       1024 .
<DIR>       1024 ..
           34642 tegra124-pm375.dtb
             908 extlinux.conf
         5910248 zImage
```

PRINTING THE U-BOOT ENVIRONMENT

You can print the entire U-Boot environment.

To print the U-Boot environment

- Execute the following command:

```
# printenv
```

PRINTING/SETTING ENVIRONMENT VARIABLES

You can print and set environment variables.

To print an environment variable

- Execute the following command:

```
# printenv <environtment_variable>
```

Where <environtment_variable> refers to an environment variable in U-Boot.

For example, to print the boot device partition number, execute:

```
# printenv pn
```

Output can be as follows:

```
pn=1
```

To set an environment variable

- Execute the following command:

```
# setenv <environtment_variable> <new_value>
```

Where <environtment_variable> refers to an environment variable in U-Boot and <new_value> is the new value for that variable.

For example, to set the partition number variable, enter the following command:

```
# setenv pn 1
```

To save the modified environment

- Execute the following command:

```
# saveenv
```

The saved modified environment is preserved in case of resets and reboots.

Notice

Trademarks

Copyright