5.9.4. Execution Control Commands

5.9.4.1. source - run script from memory

```
=> help source
source - run script from memory


Usage:
source [addr]
   - run script starting at addr
   - A valid image header must be present
=>
```

With the `source` command you can run "shell" scripts under U-Boot: You create a U-Boot script image by simply writing the commands you want to run into a text file; then you will have to use the `mkimage` tool to convert this text file into a U-Boot image (using the image type `script`).

This image can be loaded like any other image file, and with `source` you can run the commands in such an image. For instance, the following text file:

```
echo
echo Network Configuration:
echo ----------------------
echo Target:
printenv ipaddr hostname
echo
echo Server:
printenv serverip rootpath
echo
```

can be converted into a U-Boot script image using the `mkimage` command like this:

```
bash$ mkimage -A ppc -O linux -T script -C none -a 0 -e 0 \
> -n "autoscr example script" \
> -d ./testsystems/dulg/testcases/example.script /tftpboot/duts/canyonlands/example.scr
Image Name:   autoscr example script
Created:      Mon Feb  8 16:36:04 2010
Image Type:   PowerPC Linux Script (uncompressed)
```

```
Data Size:    157 Bytes = 0.15 kB = 0.00 MB
Load Address: 0x00000000
Entry Point:  0x00000000
Contents:
   Image 0:    149 Bytes =   0 kB = 0 MB
```

Now you can load and execute this script image in U-Boot:

```
=> tftp 0x100000 /tftpboot/duts/canyonlands/example.scr
Using ppc_4xx_eth0 device
TFTP from server 192.168.1.1; our IP address is 192.168.100.6
Filename '/tftpboot/duts/canyonlands/example.scr'.
Load address: 0x100000
Loading: #
done
Bytes transferred = 221 (dd hex)
=> imi

## Checking Image at 00100000 ...
   Legacy image found
   Image Name:    autoscr example script
   Created:    2010-02-08  15:36:04 UTC
   Image Type:    PowerPC Linux Script (uncompressed)
   Data Size:    157 Bytes =  0.2 kB
   Load Address: 00000000
   Entry Point:    00000000
   Contents:
      Image 0: 149 Bytes =  0.1 kB
   Verifying Checksum ... OK
=> source 0x100000
## Executing script at 00100000

Network Configuration:
----------------------
Target:
ipaddr=192.168.100.6
hostname=canyonlands
```

```
Server:
serverip=192.168.1.1
rootpath=/opt/eldk/ppc_4xxFP

=>
```

5.9.4.2. bootm - boot application image from memory

```
=> help bootm
bootm - boot application image from memory

Usage:
bootm [addr [arg ...]]
    - boot application image stored in memory
   passing arguments 'arg ...'; when booting a Linux kernel,
   'arg' can be the address of an initrd image
   When booting a Linux kernel which requires a flat device-tree
   a third argument is required which is the address of the
   device-tree blob. To boot that kernel without an initrd image,
   use a '-' for the second argument. If you do not pass a third
   a bd_info struct will be passed instead

Sub-commands to do part of the bootm sequence.   The sub-commands must be
issued in the order below (it's ok to not issue all sub-commands):
   start [addr [arg ...]]
   loados   - load OS image
   ramdisk - relocate initrd, set env initrd_start/initrd_end
   fdt    - relocate flat device tree
   cmdline - OS specific command line processing/setup
   bdt    - OS specific bd_t processing
   prep    - OS specific prep before relocation or go
   go    - start OS
=>
```

The `bootm` command is used to start operating system images. From the image header it gets information about the type of the operating system, the file compression method used (if any), the load and entry point addresses, etc. The command will then load the image to the required memory address, uncompressing it on the fly if necessary. Depending on the OS it will pass the required boot arguments and start the OS at it's entry point.

The first argument to `bootm` is the memory address (in RAM, ROM or flash memory) where the image is stored, followed by optional arguments that depend on the OS.

`Linux` requires the flattened device tree blob to be passed at boot time, and `bootm` expects its third argument to be the address of the blob in memory. Second argument to `bootm` depends on whether an `initrd` initial ramdisk image is to be used. If the kernel should be booted without the initial ramdisk, the second argument should be given as "-", otherwise it is interpreted as the start address of `initrd` (in RAM, ROM or flash memory).

To boot a Linux kernel image without a `initrd` ramdisk image, the following command can be used:

```
=> bootm ${kernel_addr} - ${fdt_addr}
```

If a ramdisk image shall be used, you can type:

```
=> bootm ${kernel_addr} ${ramdisk_addr} ${fdt_addr}
```

Both examples of course imply that the variables used are set to correct addresses for a kernel, fdt blob and a `initrd` ramdisk image.

⚠ When booting images that have been loaded to RAM (for instance using TFTP download) you have to be careful that the locations where the (compressed) images were stored do not overlap with the memory needed to load the uncompressed kernel. For instance, if you load a ramdisk image at a location in low memory, it may be overwritten when the Linux kernel gets loaded. This will cause undefined system crashes.

5.9.4.3. go - start application at address 'addr'

```
=> help go

go - start application at address 'addr'



Usage:

go addr [arg ...]

    - start application at address 'addr'

      passing 'arg' as arguments

=>
```

U-Boot has support for so-called *standalone applications*. These are programs that do not require the complex environment of an operating system to run. Instead they can be loaded and executed by U-Boot directly, utilizing U-Boot's service functions like console I/O or *malloc()* and *free()*.

This can be used to dynamically load and run special extensions to U-Boot like special hardware test routines or bootstrap code to load an OS image from some filesystem.

The `go` command is used to start such standalone applications. The optional arguments are passed to the application without modification. For more information see5.12. U-Boot Standalone Applications.