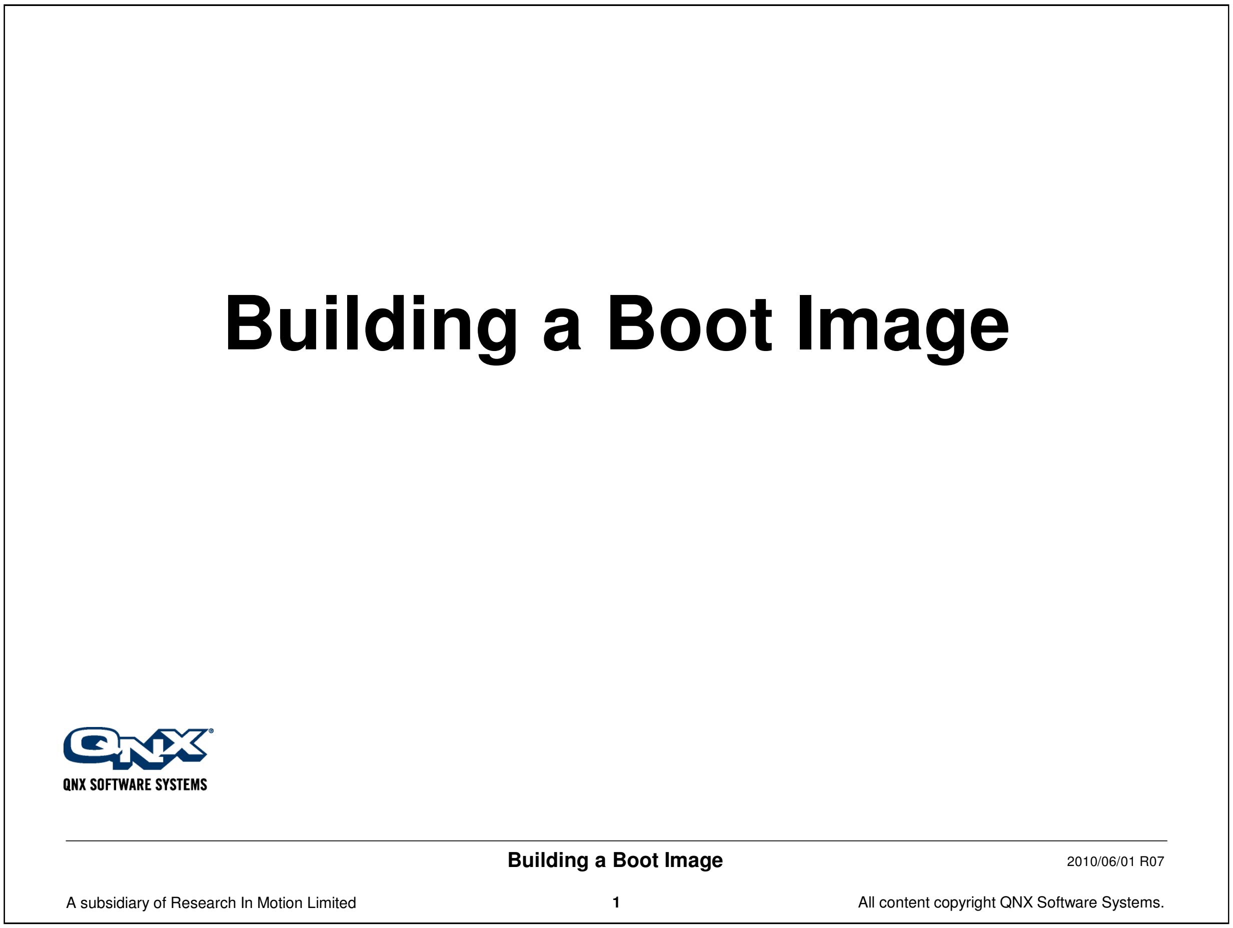
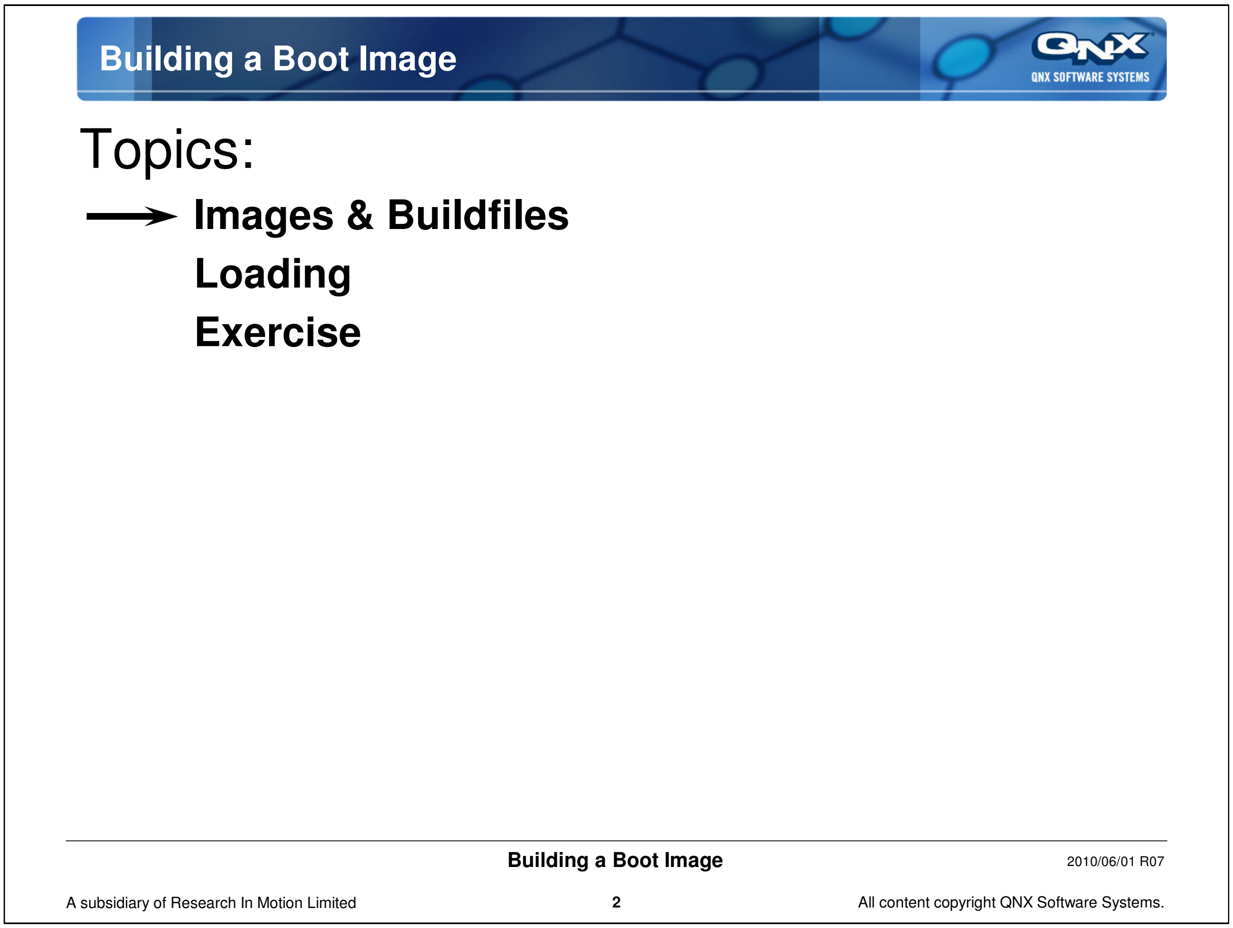


Building a Boot Image

NOTES:



Building a Boot Image

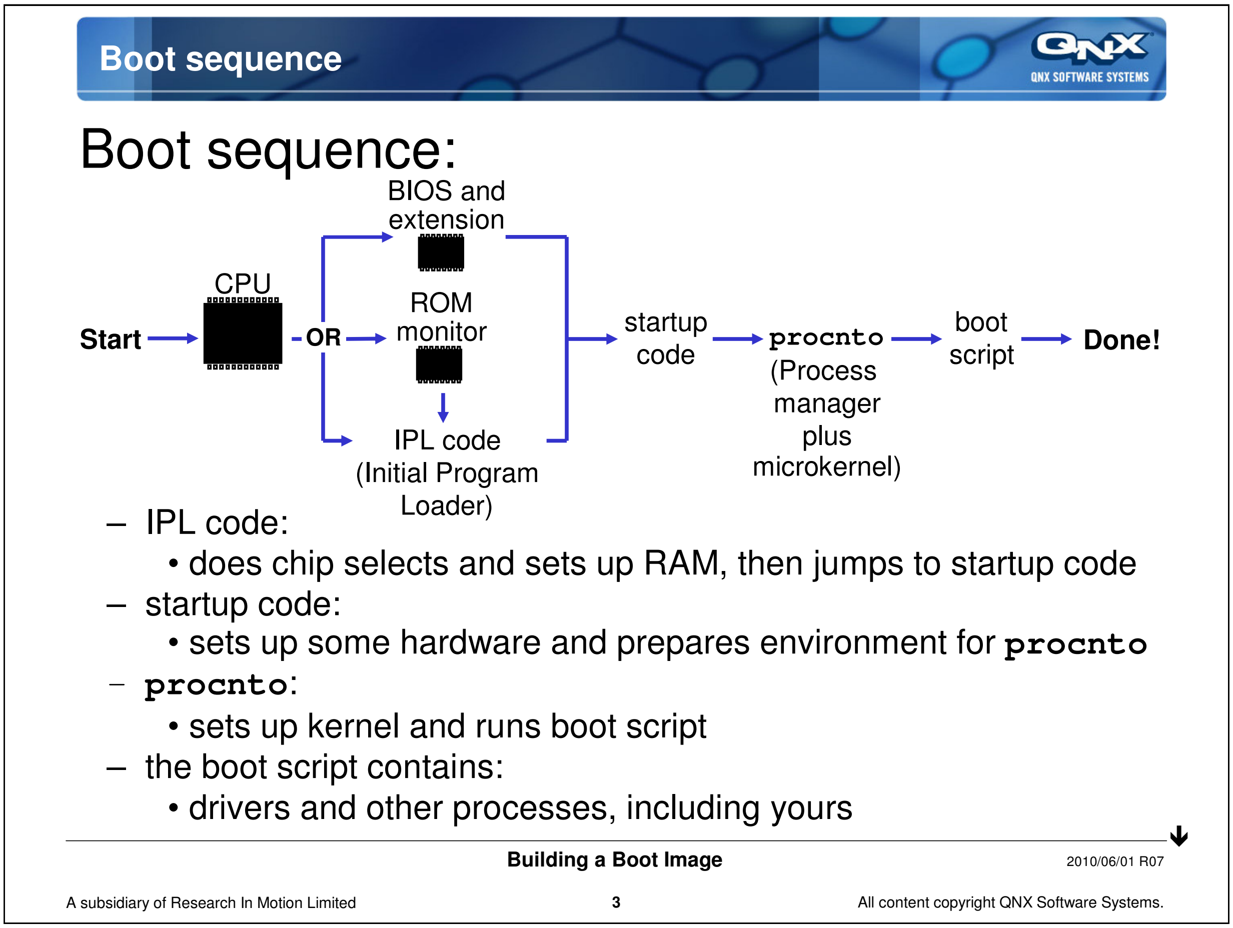
Topics:

→ **Images & Buildfiles**

Loading

Exercise

NOTES:



Boot sequence

Boot sequence:

Start

CPU

OR

BIOS and extension

ROM monitor

IPL code (Initial Program Loader)

startup code

procnto (Process manager plus microkernel)

boot script

Done!

- IPL code:
  - does chip selects and sets up RAM, then jumps to startup code
- startup code:
  - sets up some hardware and prepares environment for **`procnto`**
- **`procnto`**:
  - sets up kernel and runs boot script
- the boot script contains:
  - drivers and other processes, including yours

Building a Boot Image 2010/06/01 R07

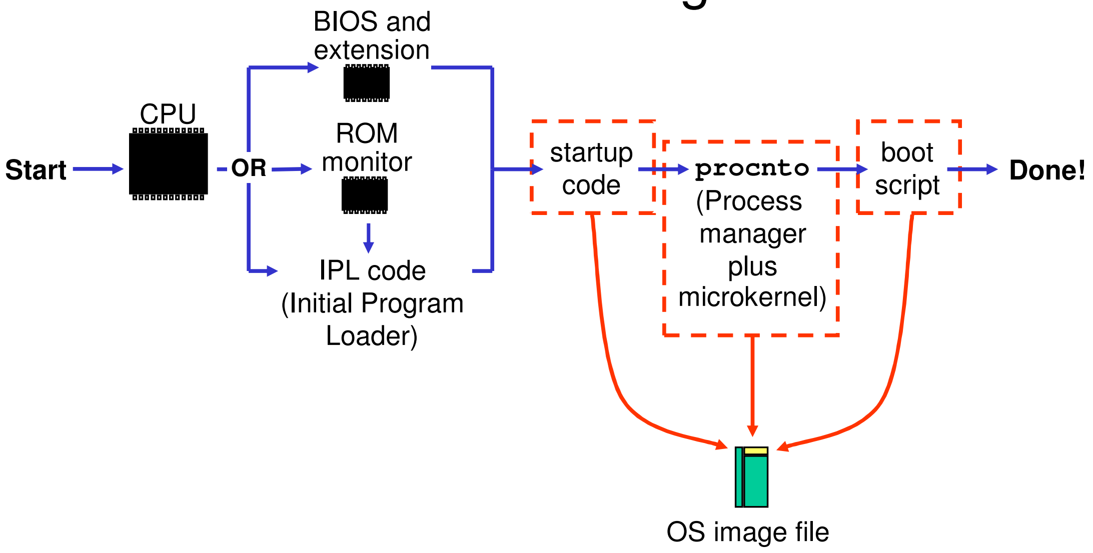
A subsidiary of Research In Motion Limited 3

NOTES:

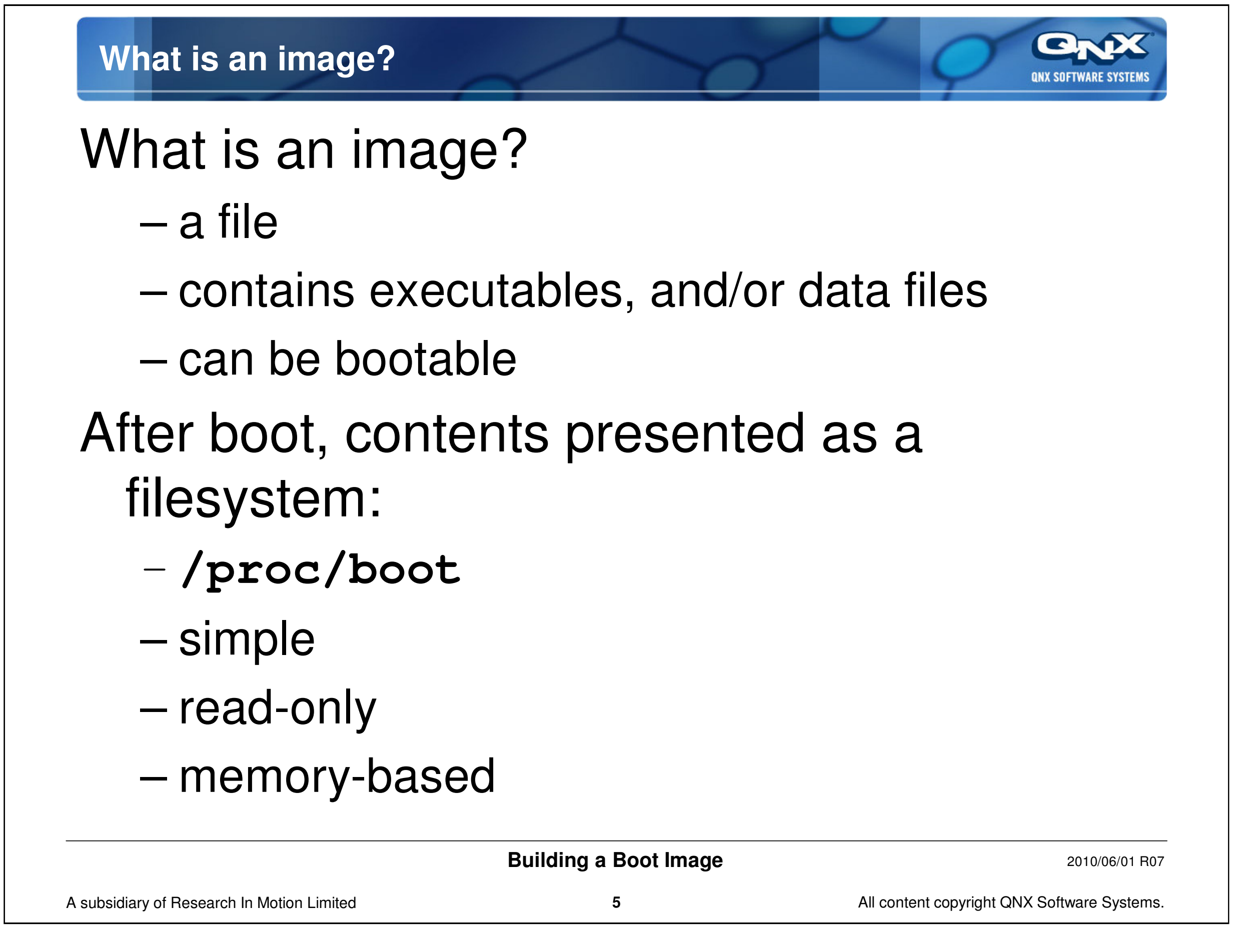
When you first boot up, the CPU executes some code at the reset vector. That could be a BIOS, ROM monitor, or an IPL. If it is a BIOS then the BIOS will find and jump to a BIOS extension (e.g.s network boot ROM, disk controller ROM) which will load and jump to the next step. If it is a ROM monitor (e.g. the RPX utility on the rpx-lite board) then the ROM monitor usually next jumps to the IPL code. In either case, the next thing that runs is some startup code which then runs **`procnto`**. **`procnto`** then runs the boot script which contains the commands for running everything else.

OS image file

Much of this is in an OS image file:



NOTES:



What is an image?

What is an image?

- a file
- contains executables, and/or data files
- can be bootable

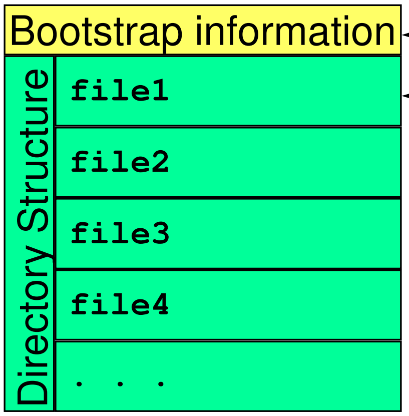
After boot, contents presented as a filesystem:

- **`/proc/boot`**
- simple
- read-only
- memory-based

NOTES:

What is an image?

Image:



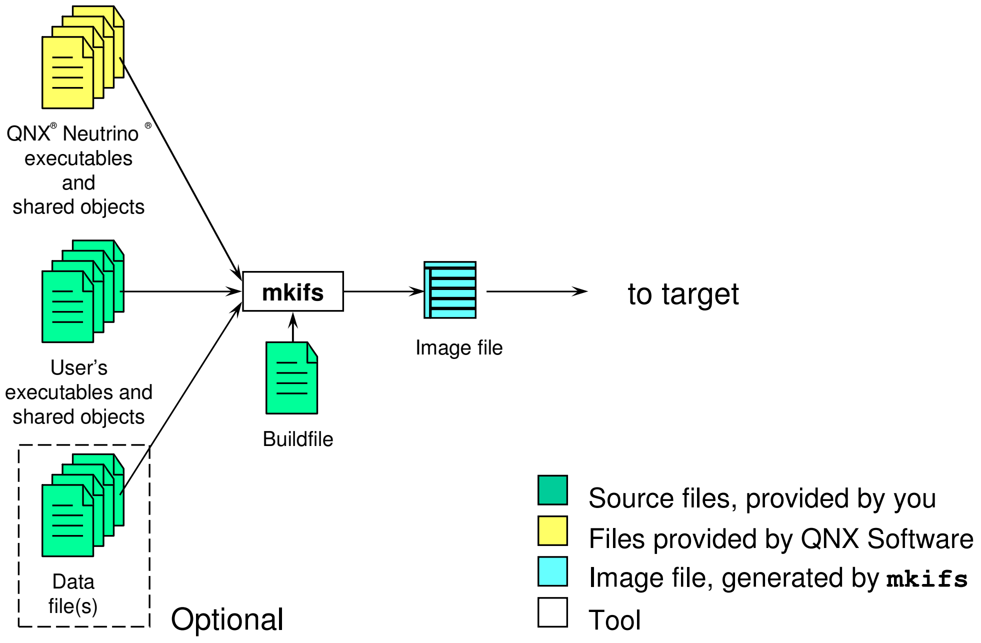
only on bootable images

files as specified by the **mkifs** build file

NOTES:

The image tool chain

Images are created by the following:



QNX® Neutrino® executables and shared objects

User's executables and shared objects

Data file(s)

Optional

mkifs

Buildfile

Image file

to target

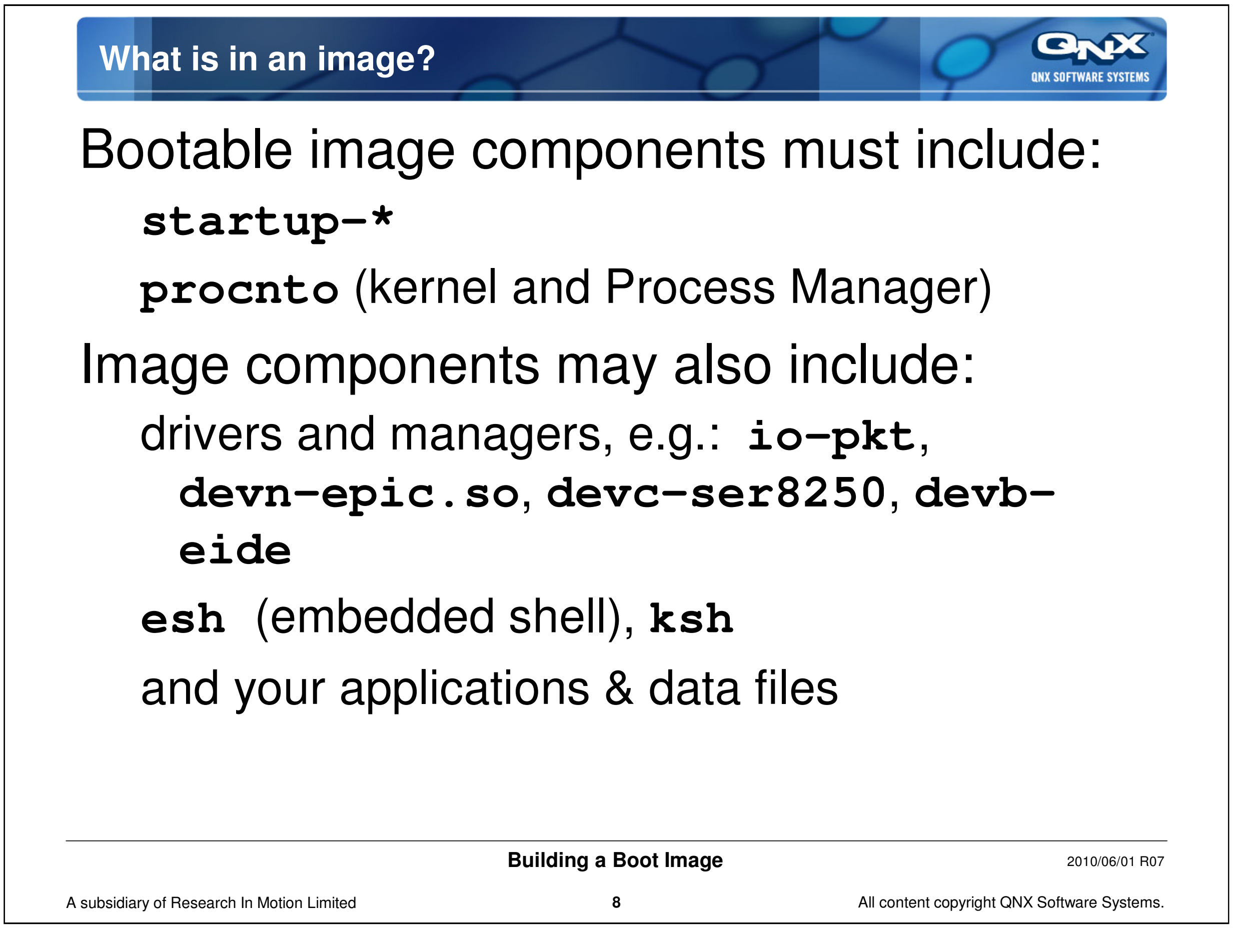
🟩 Source files, provided by you

🟨 Files provided by QNX Software

🟦 Image file, generated by **`mkifs`**

□ Tool

NOTES:



What is in an image?

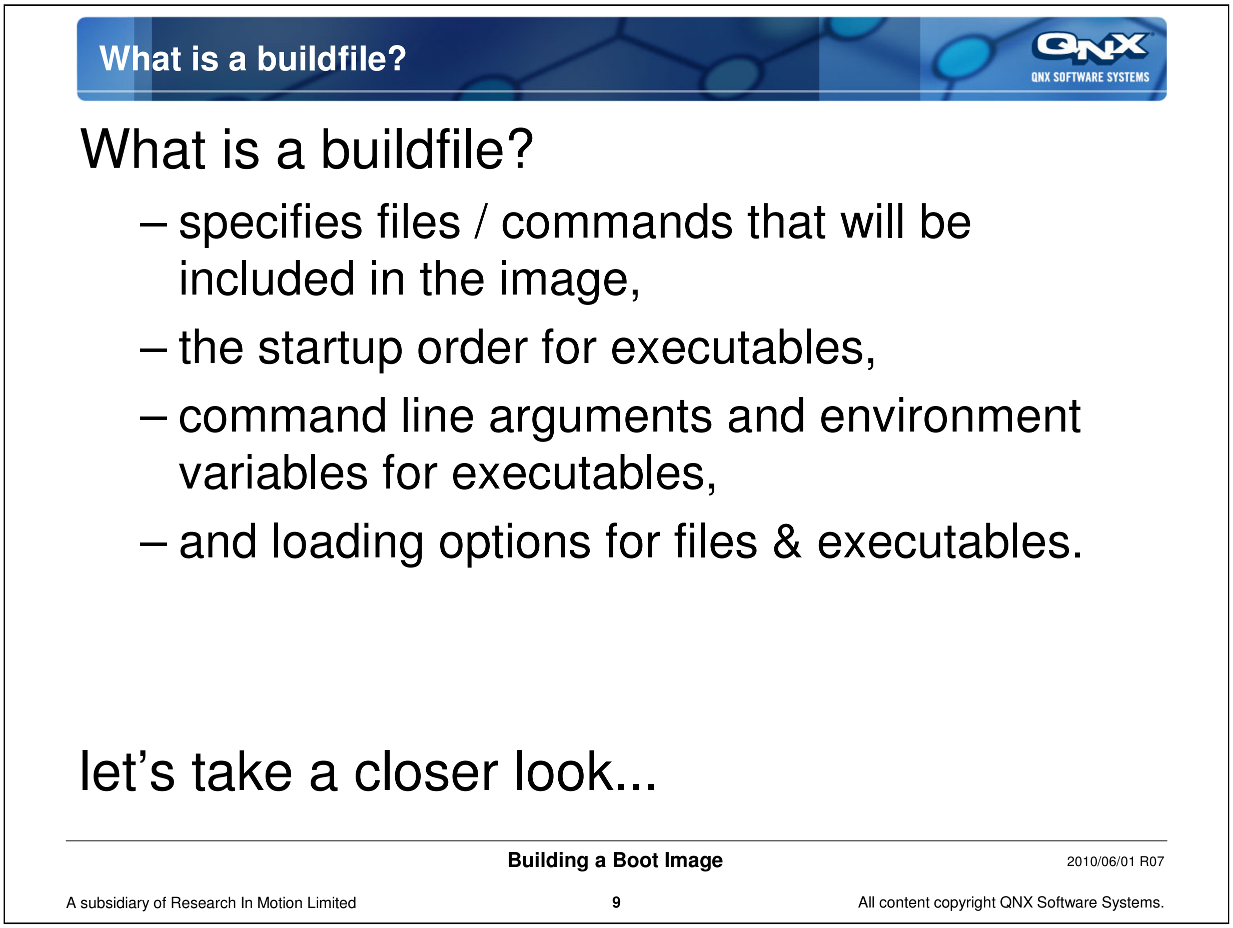
Bootable image components must include:

- **`startup-*`**
- **`procnto`** (kernel and Process Manager)

Image components may also include:

- drivers and managers, e.g.: **`io-pkt`**, **`devn-epic.so`**, **`devc-ser8250`**, **`devb-eide`**
- **`esh`** (embedded shell), **`ksh`**
- and your applications & data files

NOTES:



What is a buildfile?

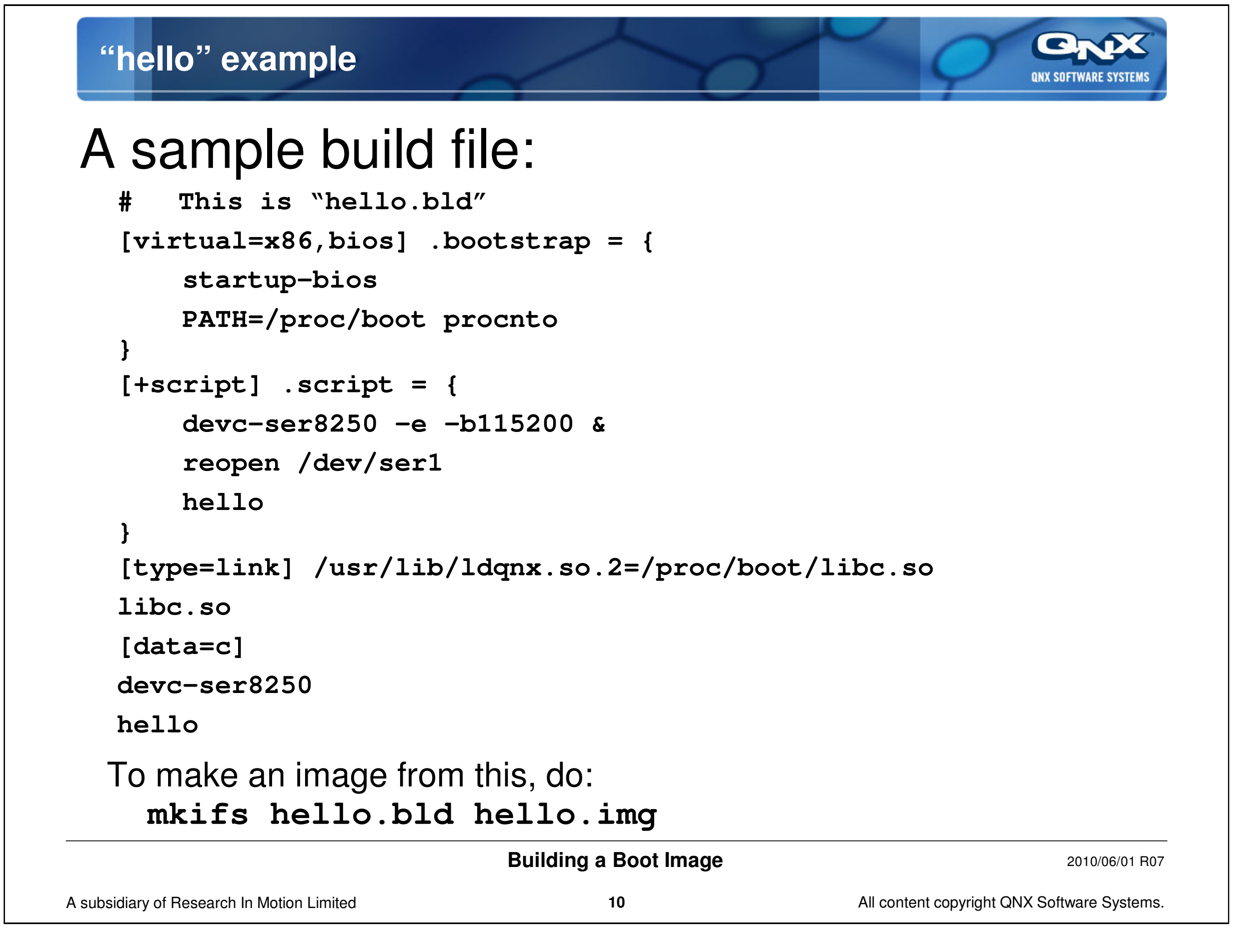
QNX SOFTWARE SYSTEMS

What is a buildfile?

- specifies files / commands that will be included in the image,
- the startup order for executables,
- command line arguments and environment variables for executables,
- and loading options for files & executables.

let’s take a closer look...

NOTES:



“hello” example

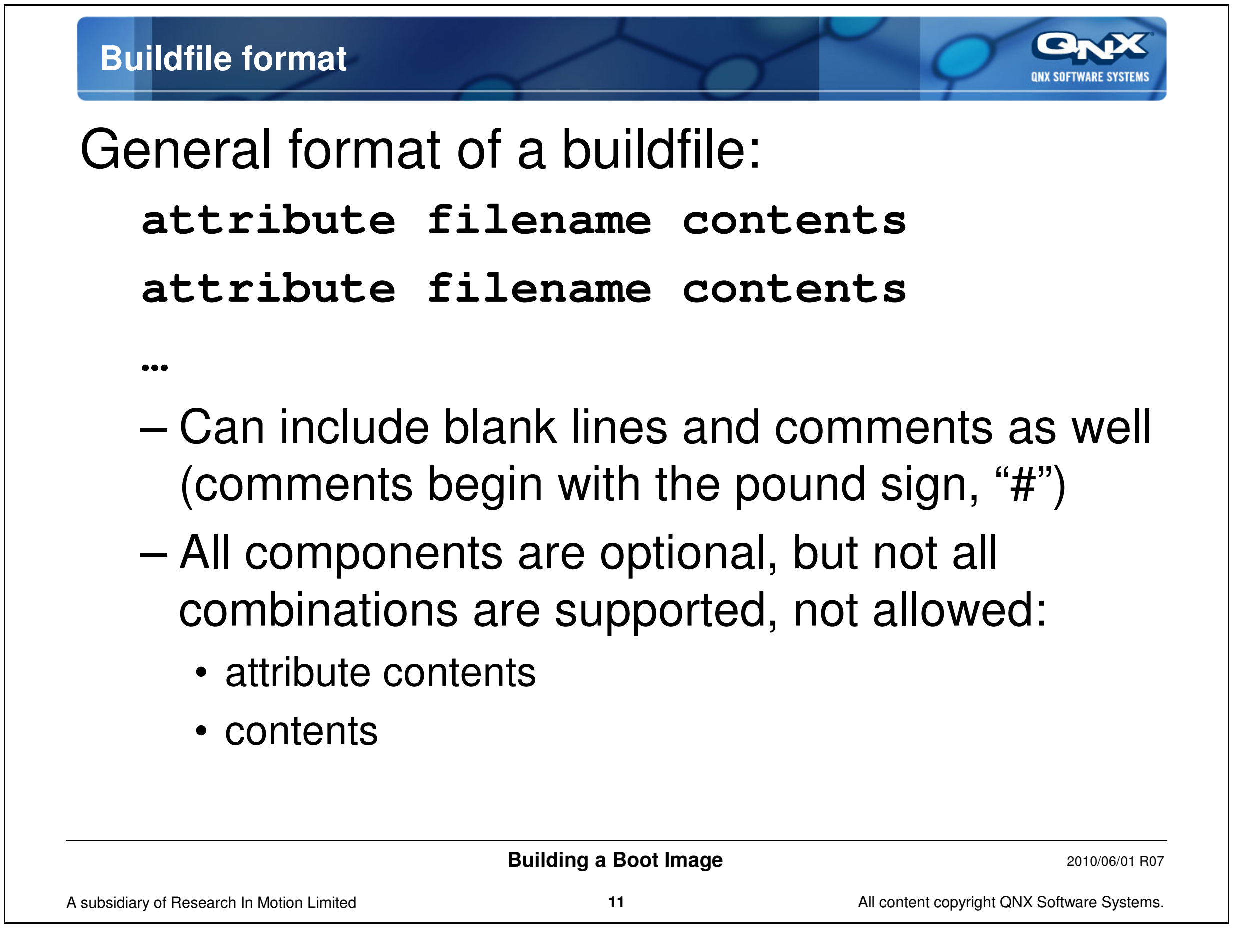
A sample build file:

```
#   This is “hello.bld”
[virtual=x86,bios] .bootstrap = {
    startup-bios
    PATH=/proc/boot procnto
}
[+script] .script = {
    devc-ser8250 -e -b115200 &
    reopen /dev/ser1
    hello
}
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
libc.so
[data=c]
devc-ser8250
hello
```

To make an image from this, do:

```
mkifs hello.bld hello.img
```

NOTES:



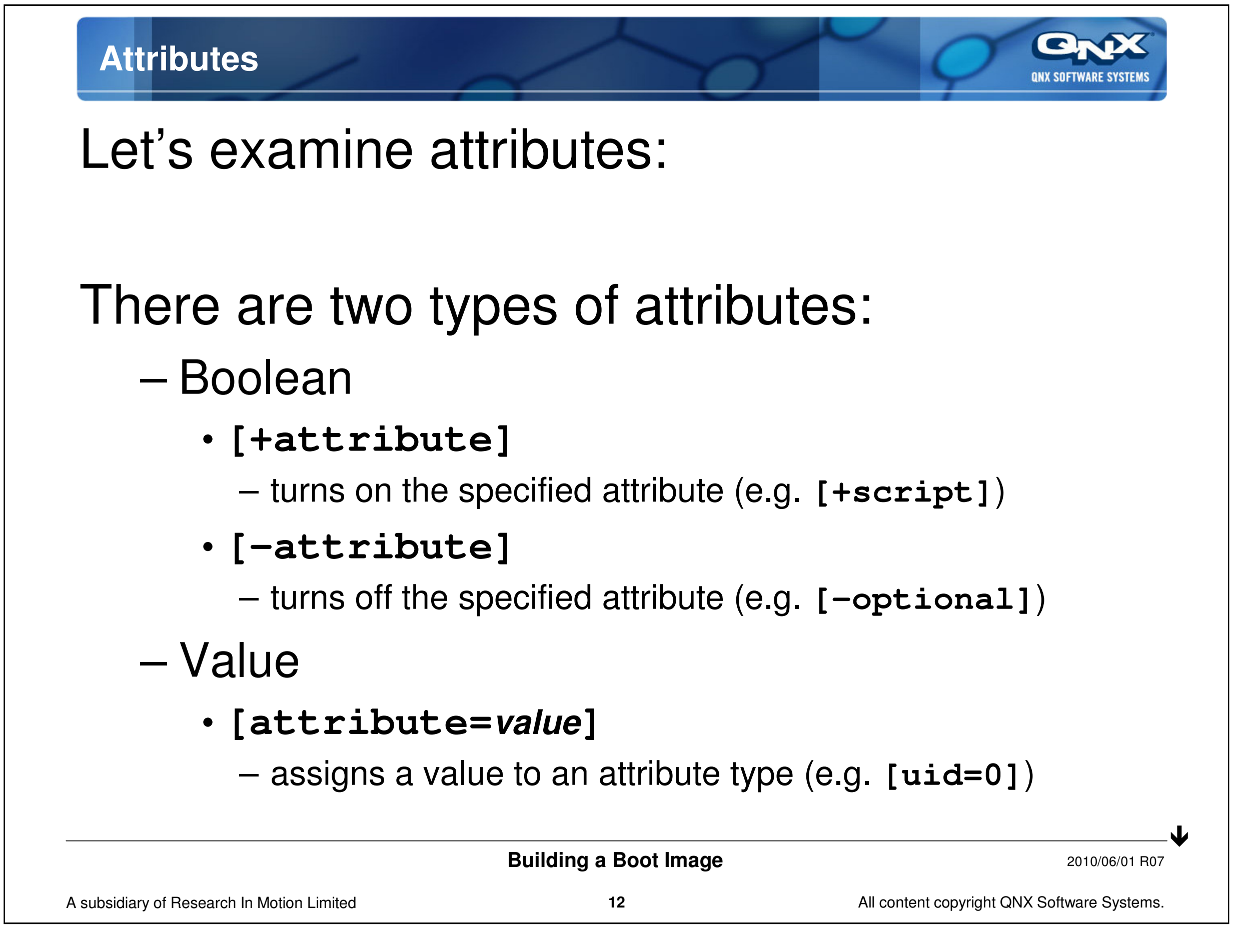
Buildfile format

General format of a buildfile:

```
attribute filename contents
attribute filename contents
…
```

- Can include blank lines and comments as well (comments begin with the pound sign, “#”)
- All components are optional, but not all combinations are supported, not allowed:
  - attribute contents
  - contents

NOTES:



Attributes

Let’s examine attributes:

There are two types of attributes:

- Boolean
  - **`[+attribute]`**
    - turns on the specified attribute (e.g. **`[+script]`**)
  - **`[-attribute]`**
    - turns off the specified attribute (e.g. **`[-optional]`**)
- Value
  - **`[attribute=`*`value`*`]`**
    - assigns a value to an attribute type (e.g. **`[uid=0]`**)

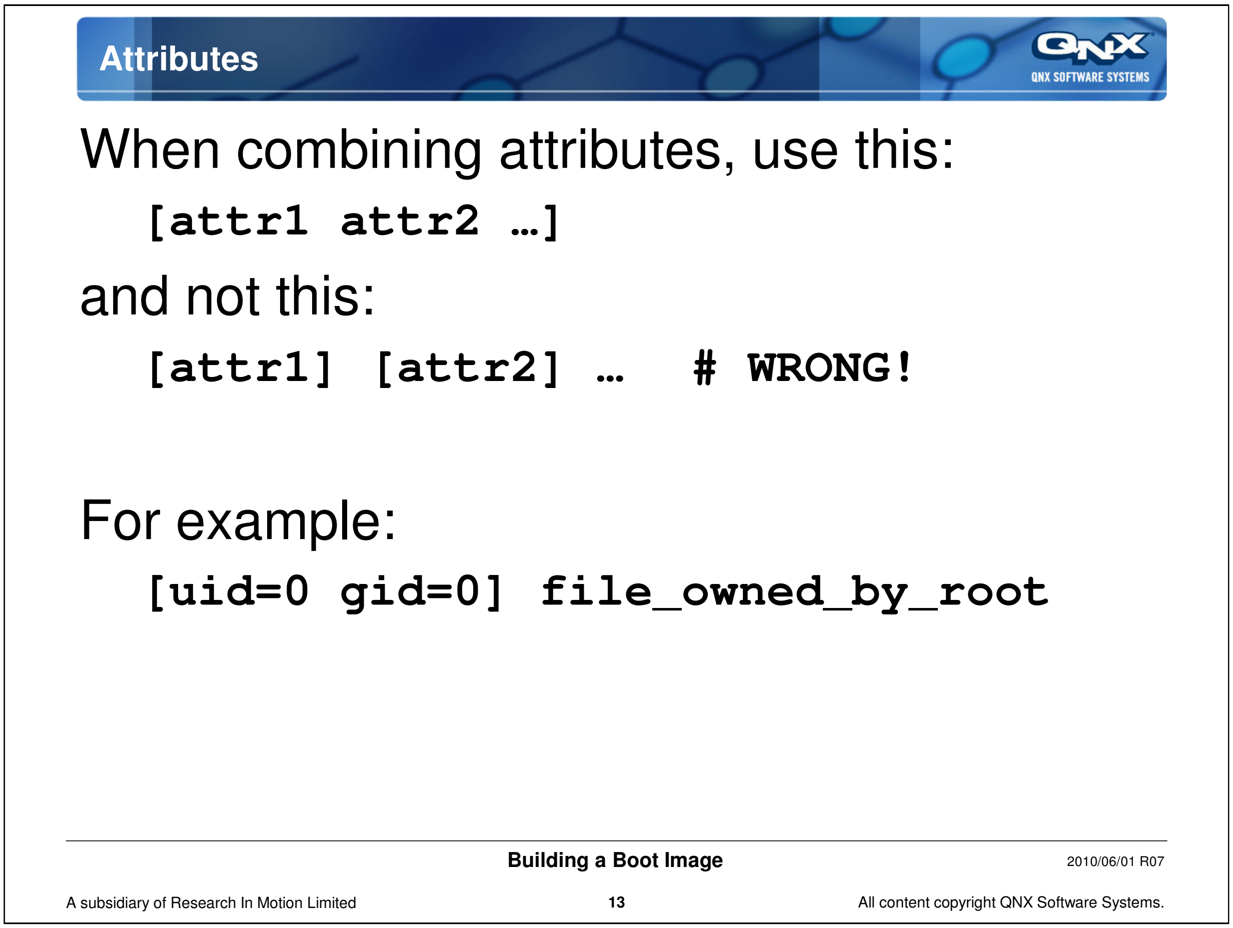
NOTES:

Note that some attributes default to “enabled” (on) or “disabled” (off). In some cases, therefore, specifying one boolean form or the other is redundant.

With the **`script`** attribute, if true, the given file is opened and processed as a script file after the process manager has initialized itself. More on this later.

The **`optional`** attribute, if false, means that if the given file can’t be found then **`mkifs`** should output an error and exit. By default, **`mkifs`** outputs an error and continues.

The **`uid=`** attribute sets the user ID for the file.



Attributes

When combining attributes, use this:

```
[attr1 attr2 …]
```

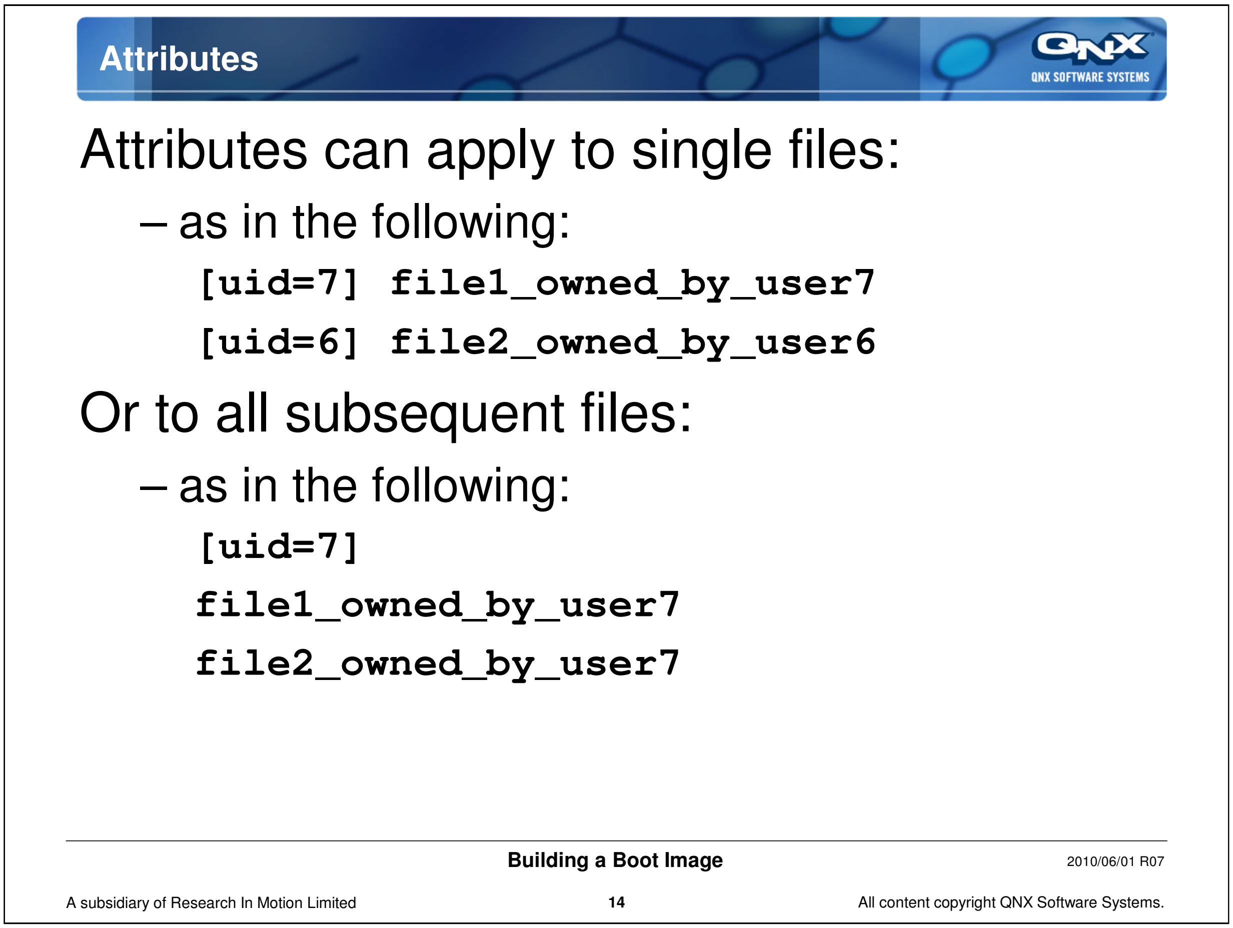
and not this:

```
[attr1] [attr2] …   # WRONG!
```

For example:

```
[uid=0 gid=0] file_owned_by_root
```

NOTES:



Attributes

Attributes can apply to single files:

- as in the following:

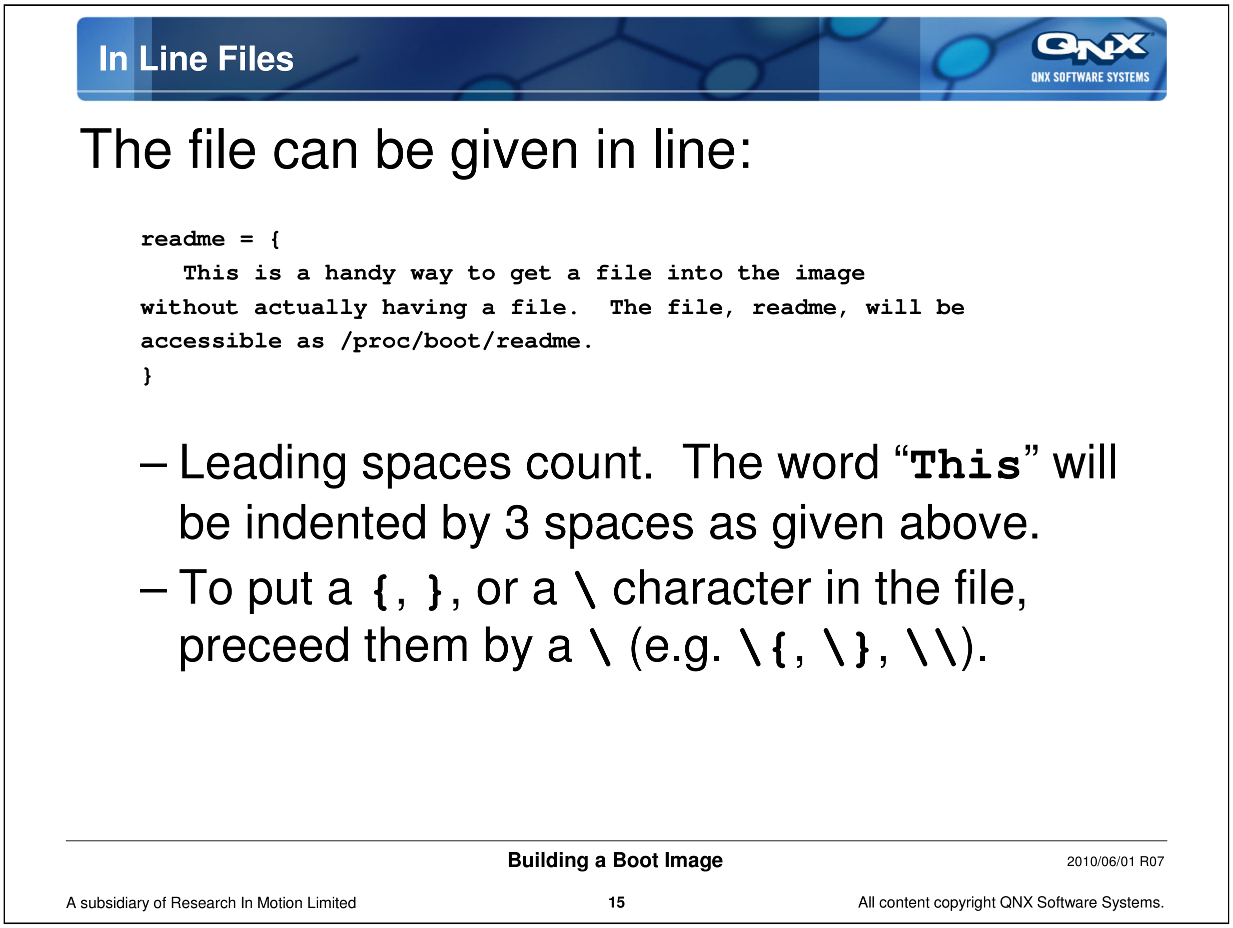
```
[uid=7] file1_owned_by_user7
[uid=6] file2_owned_by_user6
```

Or to all subsequent files:

- as in the following:

```
[uid=7]
file1_owned_by_user7
file2_owned_by_user7
```

NOTES:



In Line Files

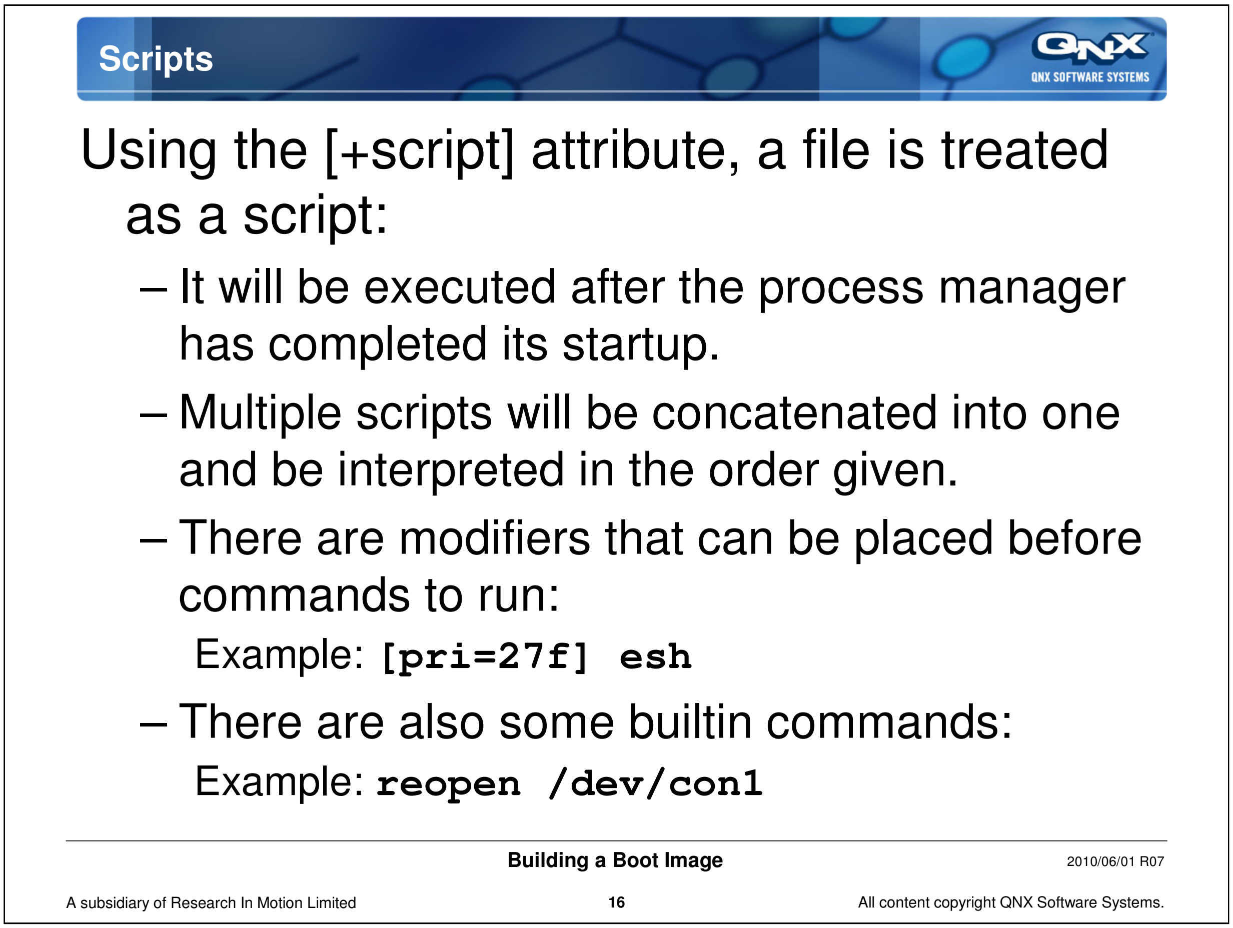
QNX SOFTWARE SYSTEMS

The file can be given in line:

```
readme = {
   This is a handy way to get a file into the image
without actually having a file.  The file, readme, will be
accessible as /proc/boot/readme.
}
```

- Leading spaces count. The word “**`This`**” will be indented by 3 spaces as given above.
- To put a **`{`**, **`}`**, or a **`\`** character in the file, preceed them by a **`\`** (e.g. **`\{`**, **`\}`**, **`\\`**).

NOTES:



Scripts

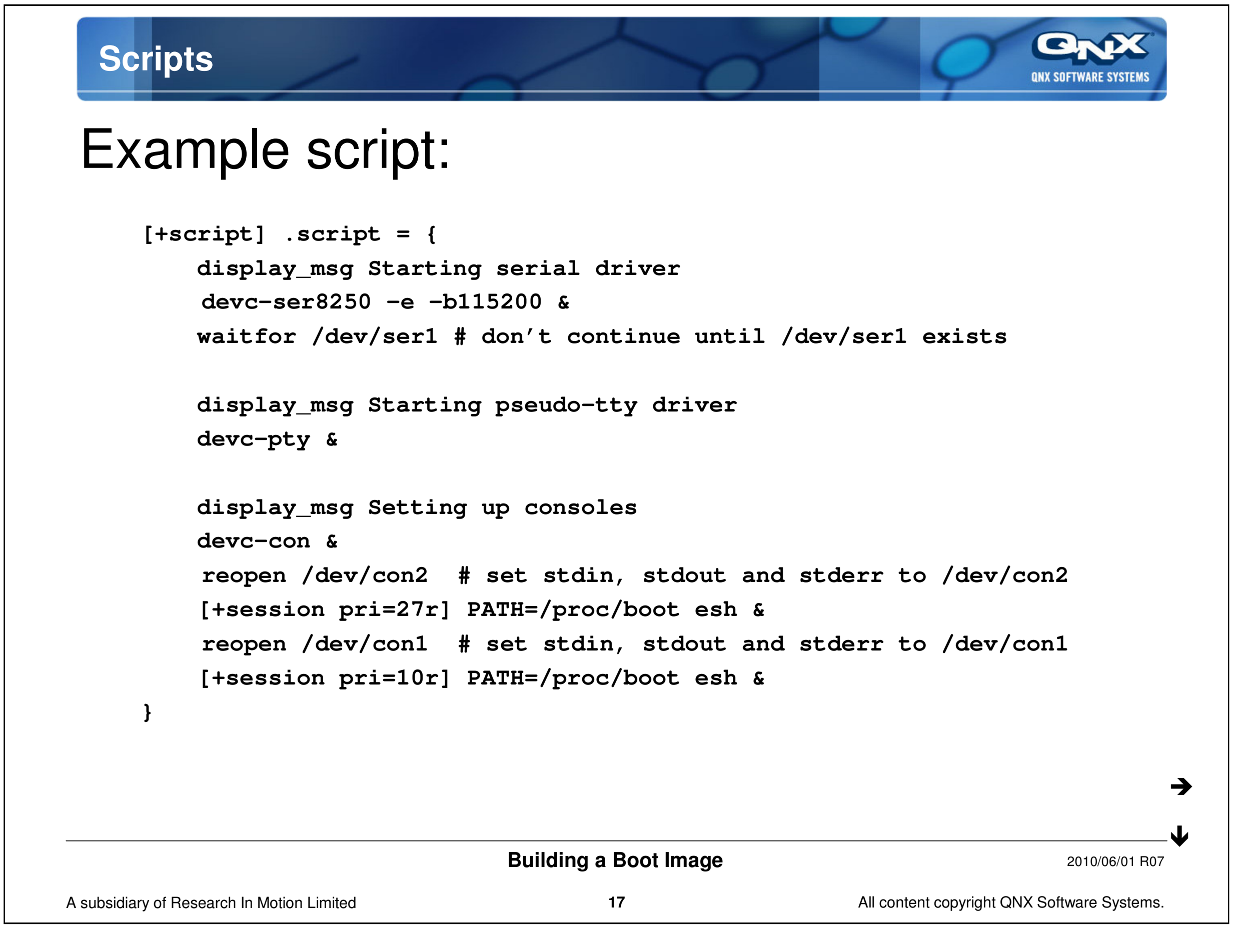
Using the [+script] attribute, a file is treated as a script:

- It will be executed after the process manager has completed its startup.
- Multiple scripts will be concatenated into one and be interpreted in the order given.
- There are modifiers that can be placed before commands to run:

  Example: **`[pri=27f] esh`**
- There are also some builtin commands:

  Example: **`reopen /dev/con1`**

NOTES:



Scripts

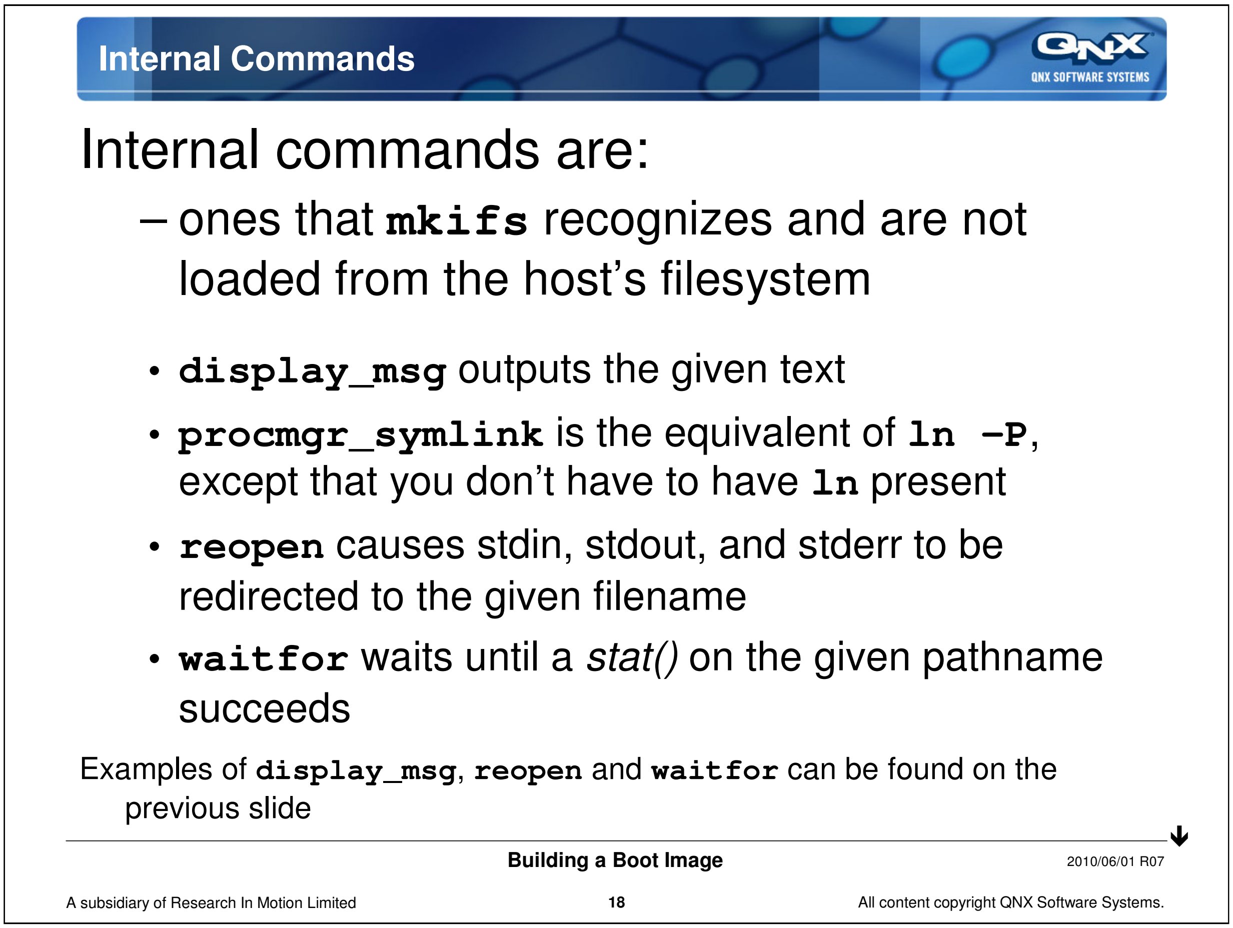
Example script:

```
[+script] .script = {
    display_msg Starting serial driver
    devc-ser8250 -e -b115200 &
    waitfor /dev/ser1 # don’t continue until /dev/ser1 exists

    display_msg Starting pseudo-tty driver
    devc-pty &

    display_msg Setting up consoles
    devc-con &
    reopen /dev/con2  # set stdin, stdout and stderr to /dev/con2
    [+session pri=27r] PATH=/proc/boot esh &
    reopen /dev/con1  # set stdin, stdout and stderr to /dev/con1
    [+session pri=10r] PATH=/proc/boot esh &
}
```

NOTES:

- **`display_msg`**, **`waitfor`** and **`reopen`** are all internal commands. They are discussed on the next slide.

- The **`session`** modifier causes a new session to be created for the command (e.g. for **`esh`**).
- The **`pri`** modifier specifies what priority and optionally, what scheduling algorithm, to run at.



Internal Commands

Internal commands are:

– ones that **mkifs** recognizes and are not loaded from the host's filesystem

- **display_msg** outputs the given text
- **procmgr_symlink** is the equivalent of **ln -P**, except that you don't have to have **ln** present
- **reopen** causes stdin, stdout, and stderr to be redirected to the given filename
- **waitfor** waits until a *stat()* on the given pathname succeeds

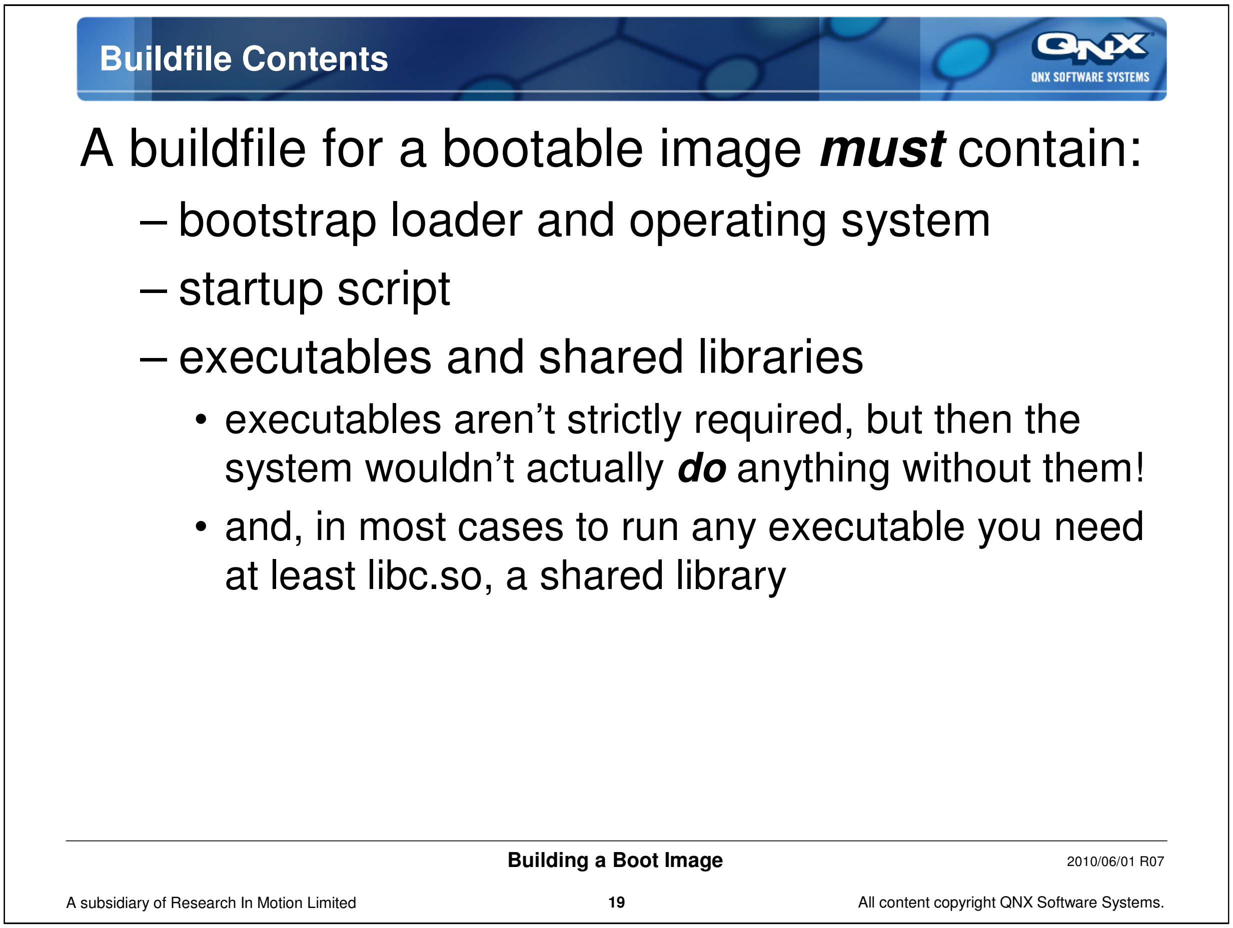
Examples of **display_msg**, **reopen** and **waitfor** can be found on the previous slide

NOTES:

If you have a command that has the same name as an internal command then you can give the **[+external]** modifier for that command.

Example (pretending that you have your own version of **display_msg**):

```
[+external] display_msg Hello
```

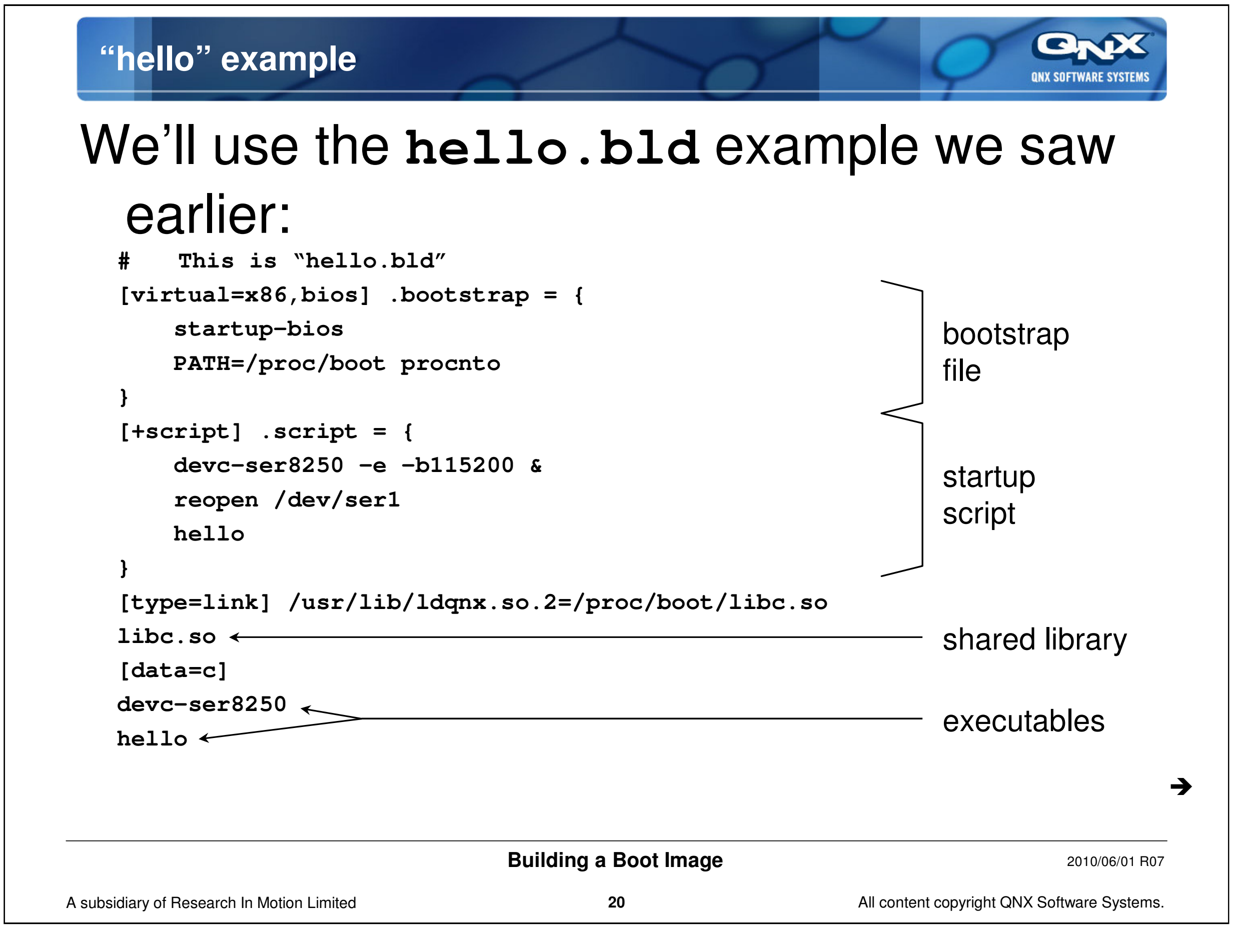


Buildfile Contents

A buildfile for a bootable image ***must*** contain:

- bootstrap loader and operating system
- startup script
- executables and shared libraries
  - executables aren’t strictly required, but then the system wouldn’t actually ***do*** anything without them!
  - and, in most cases to run any executable you need at least libc.so, a shared library

NOTES:



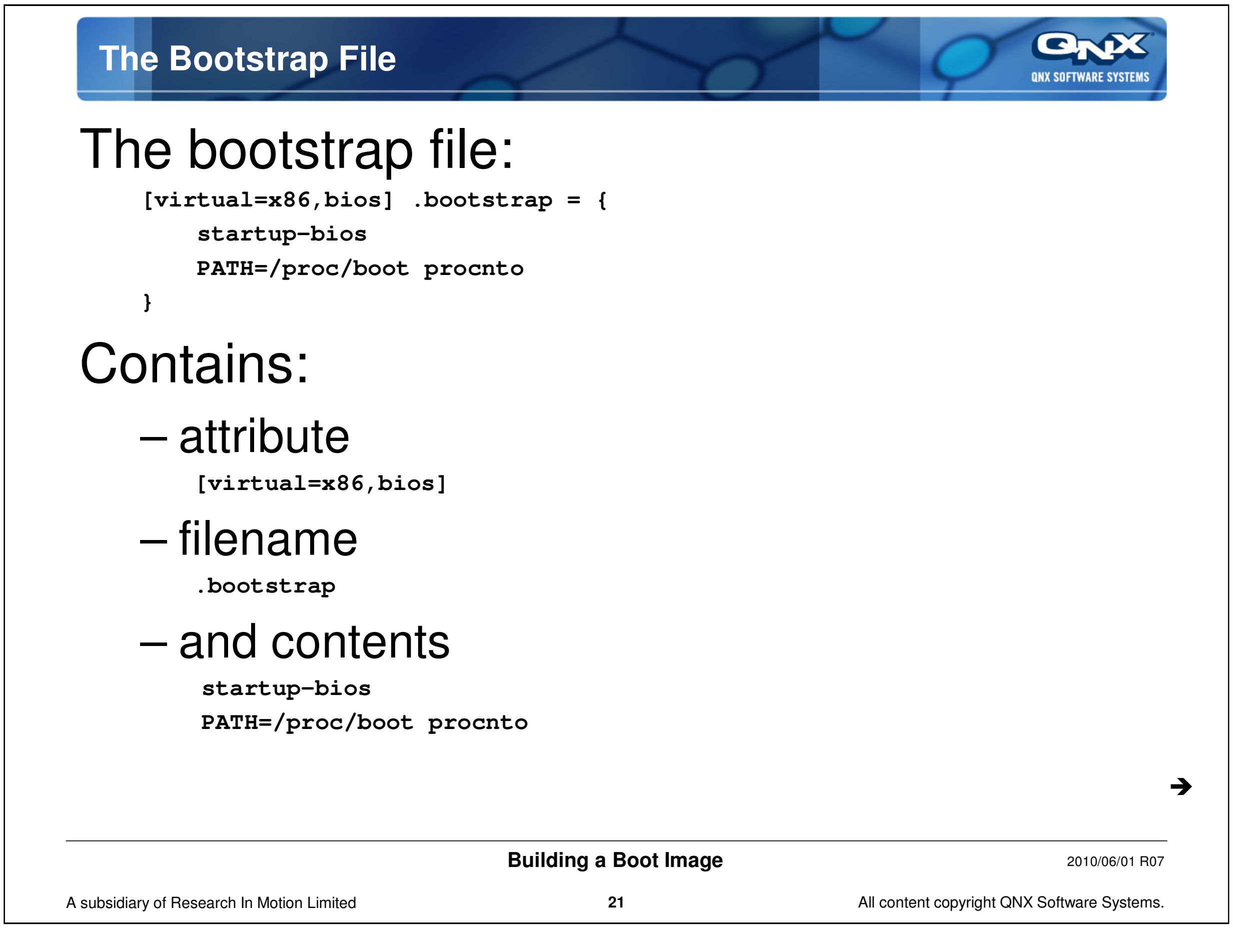
“hello” example

We’ll use the **`hello.bld`** example we saw earlier:

```
#   This is “hello.bld”
[virtual=x86,bios] .bootstrap = {
    startup-bios
    PATH=/proc/boot procnto
}
[+script] .script = {
    devc-ser8250 -e -b115200 &
    reopen /dev/ser1
    hello
}
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
libc.so
[data=c]
devc-ser8250
hello
```

bootstrap file
startup script
shared library
executables
➔

NOTES:



The Bootstrap File

The bootstrap file:

```
[virtual=x86,bios] .bootstrap = {
    startup-bios
    PATH=/proc/boot procnto
}
```

Contains:

- attribute

  `[virtual=x86,bios]`

- filename

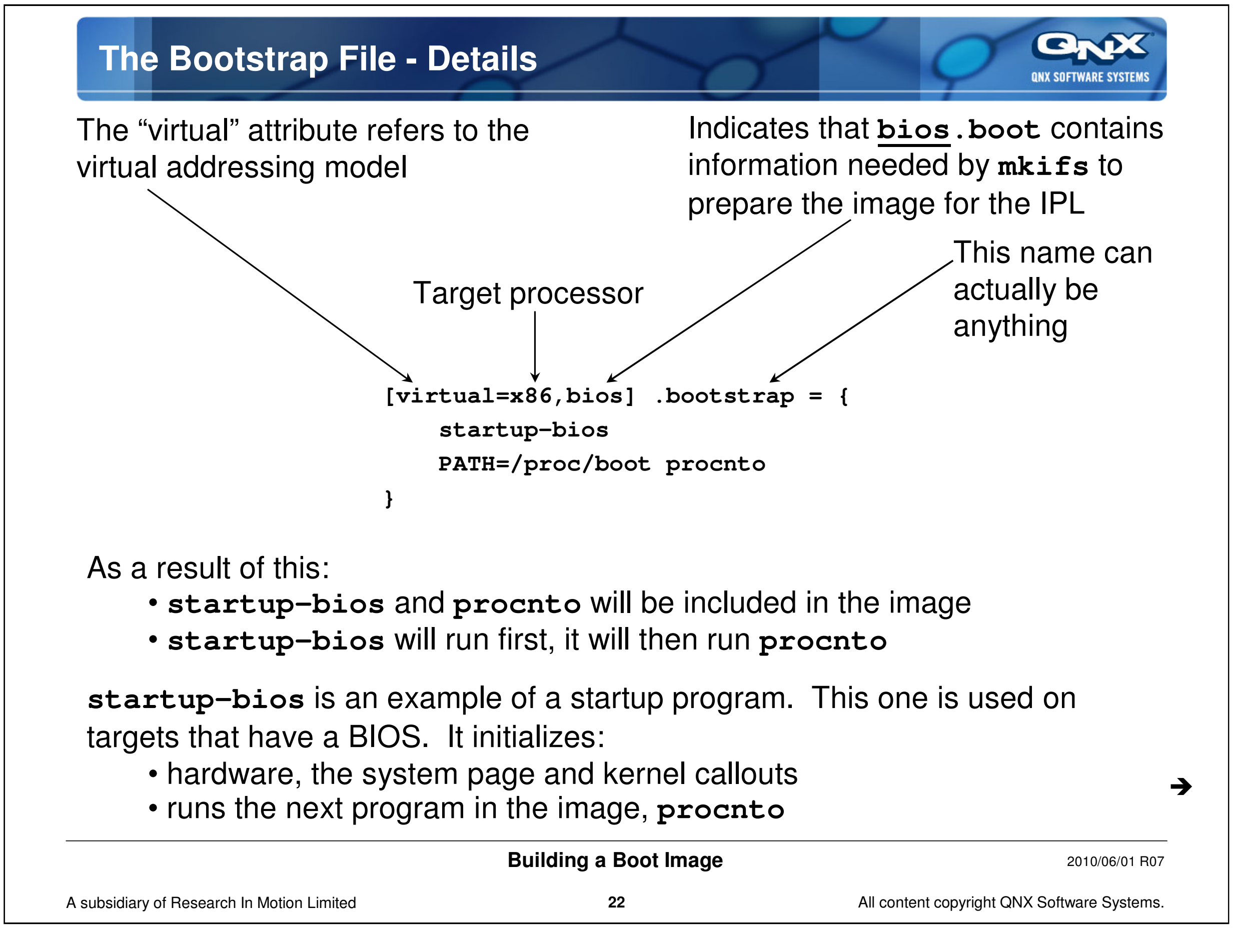
  `.bootstrap`

- and contents

  ```
  startup-bios
  PATH=/proc/boot procnto
  ```

➔

NOTES:



The Bootstrap File - Details

The “virtual” attribute refers to the virtual addressing model

Indicates that **bios.boot** contains information needed by **mkifs** to prepare the image for the IPL

Target processor

This name can actually be anything

```
[virtual=x86,bios] .bootstrap = {
    startup-bios
    PATH=/proc/boot procnto
}
```

As a result of this:

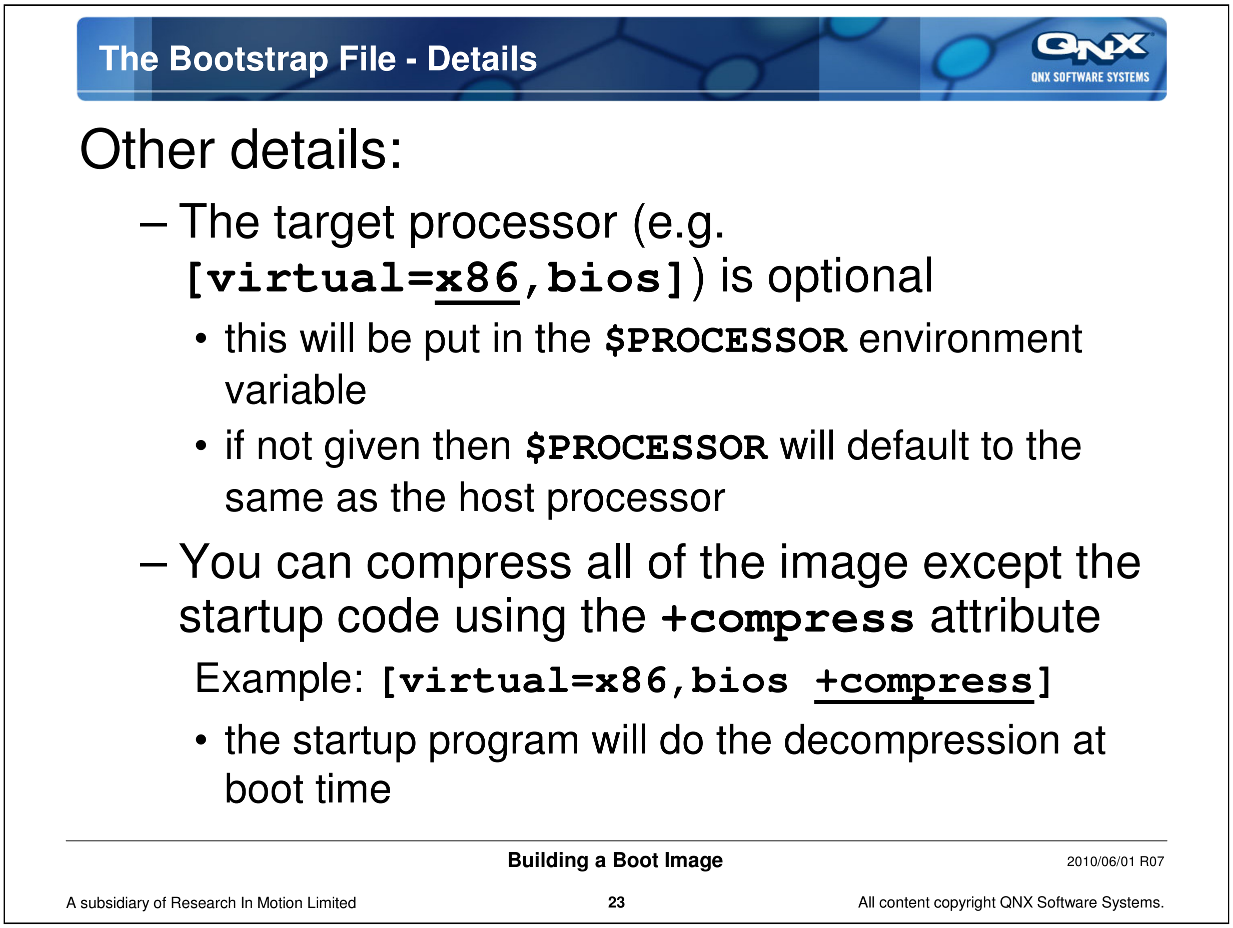
- **startup-bios** and **procnto** will be included in the image
- **startup-bios** will run first, it will then run **procnto**

startup-bios is an example of a startup program. This one is used on targets that have a BIOS. It initializes:

- hardware, the system page and kernel callouts
- runs the next program in the image, **procnto**

➔

NOTES:



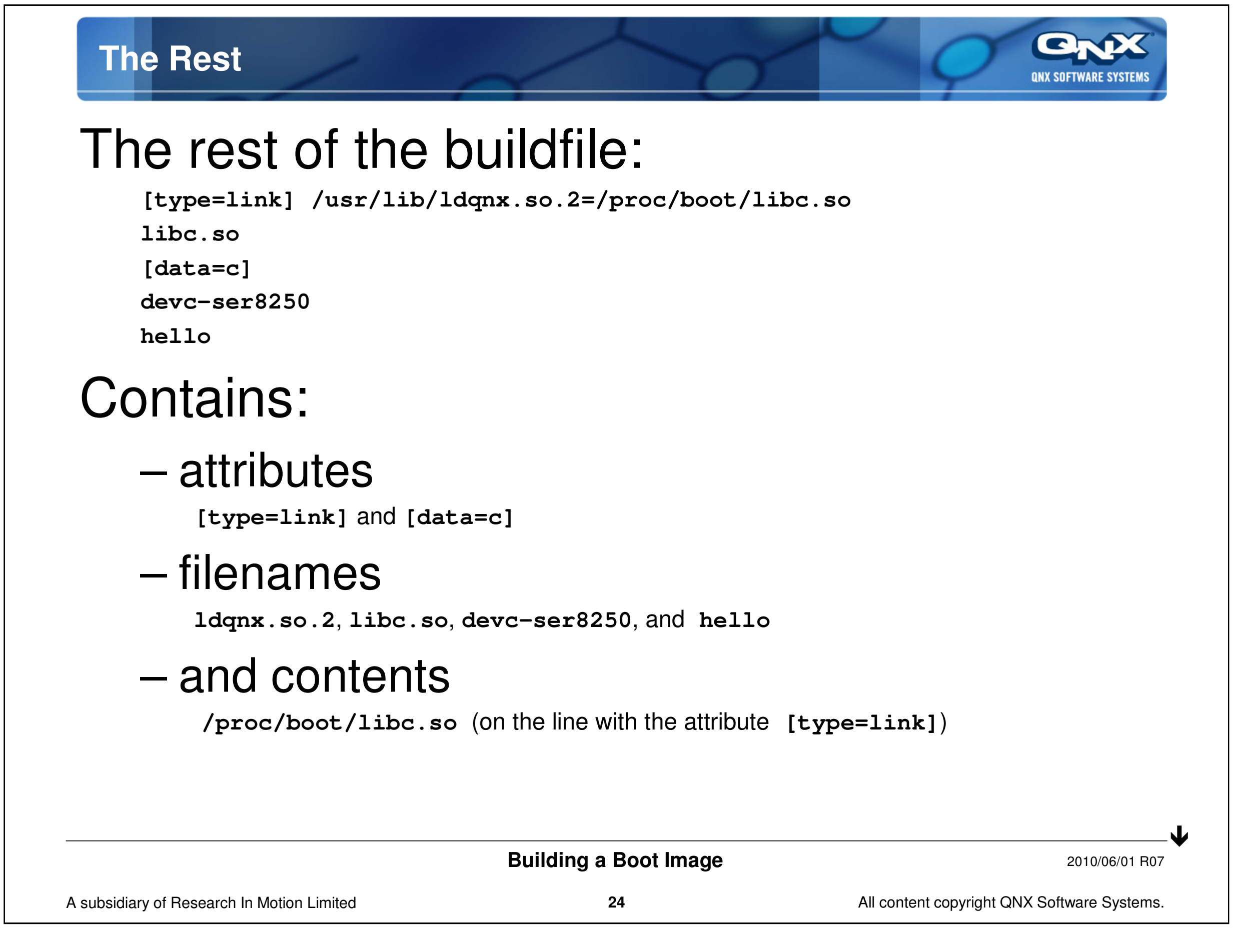
The Bootstrap File - Details

Other details:

- The target processor (e.g. `[virtual=x86,bios]`) is optional
  - this will be put in the `$PROCESSOR` environment variable
  - if not given then `$PROCESSOR` will default to the same as the host processor
- You can compress all of the image except the startup code using the `+compress` attribute

  Example: `[virtual=x86,bios +compress]`
  - the startup program will do the decompression at boot time

NOTES:



The Rest

The rest of the buildfile:

```
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
libc.so
[data=c]
devc-ser8250
hello
```

Contains:

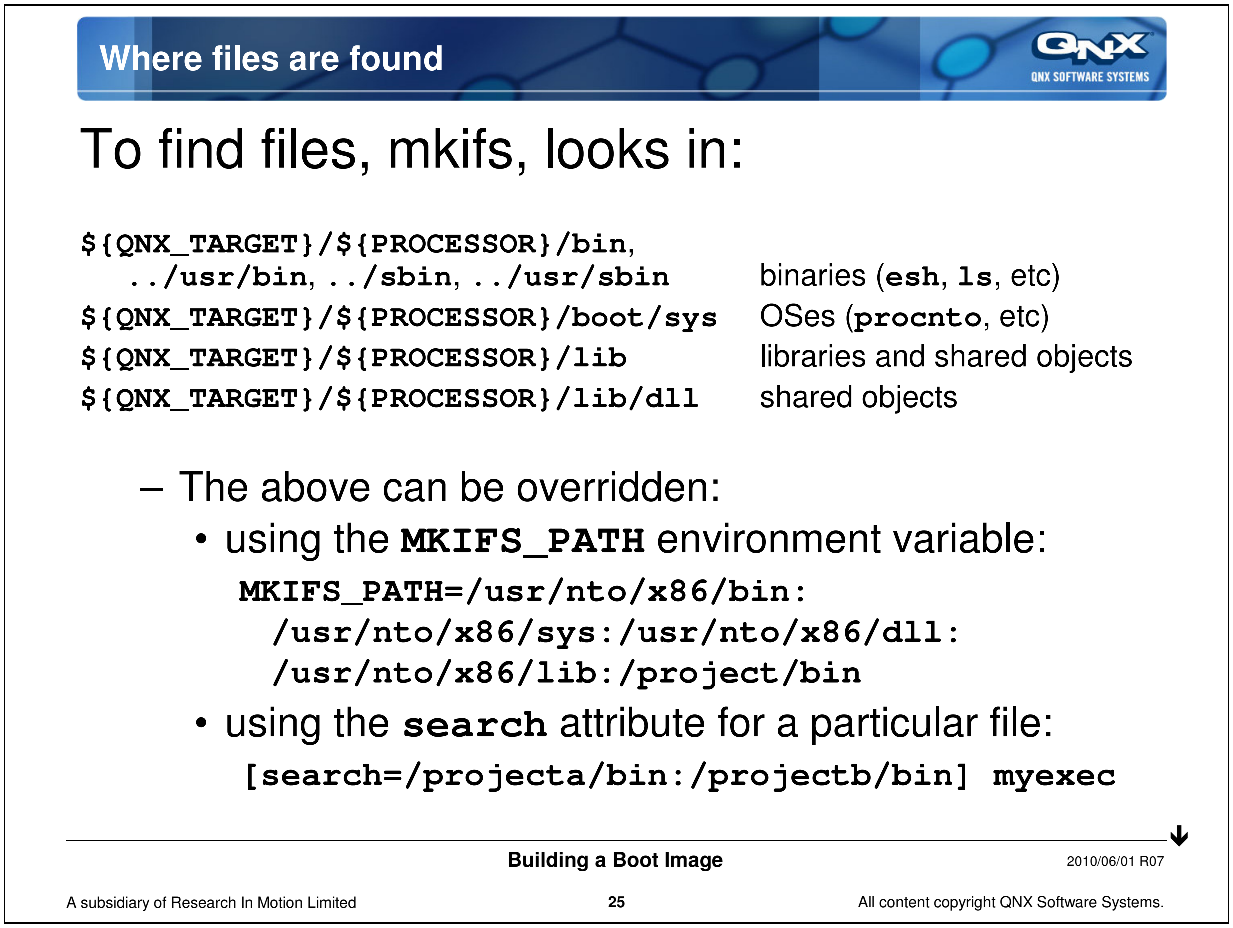
- attributes
  `[type=link]` and `[data=c]`
- filenames
  `ldqnx.so.2`, `libc.so`, `devc-ser8250`, and `hello`
- and contents
  `/proc/boot/libc.so` (on the line with the attribute `[type=link]`)

NOTES:

It is important to understand that the above executables are not run. The image is basically a filesystem so here we are just telling mkifs what files should show up in the filesystem. The bootstrap file and the startup script which we've just seen are what are run. Of course, if there is a command in a script that is not listed above then when the script is run, that command will not be found (unless something in the image provides access to another filesystem).

The "**`[type=link]`**" attribute creates a symbolic link. In this case, what we see is that the **`[type=link]`** line says that the file "**`/usr/lib/ldqnx.so.2`**" has the same contents as "**`/proc/boot/libc.so`**".

The "**`[data=c]`**" attribute says that when any of the programs that follow it in the build file are run, give them their own copy of the data. This may seem puzzling at first until you remember that the things in the build file end up in a RAM based image filesystem and are run from there. This attribute says that when they are loaded from this RAM filsystem, don't use the data that is in the RAM filesystem. Instead, copy it into a whole new data area for use by this instance of the running process. The alternative would be to use the data area in the RAM filesystem, but that will only work well once.



Where files are found

To find files, mkifs, looks in:

${QNX_TARGET}/${PROCESSOR}/bin, **../usr/bin**, **../sbin**, **../usr/sbin**	binaries (**esh**, **ls**, etc)
${QNX_TARGET}/${PROCESSOR}/boot/sys	OSes (**procnto**, etc)
${QNX_TARGET}/${PROCESSOR}/lib	libraries and shared objects
${QNX_TARGET}/${PROCESSOR}/lib/dll	shared objects

– The above can be overridden:

- using the **MKIFS_PATH** environment variable:

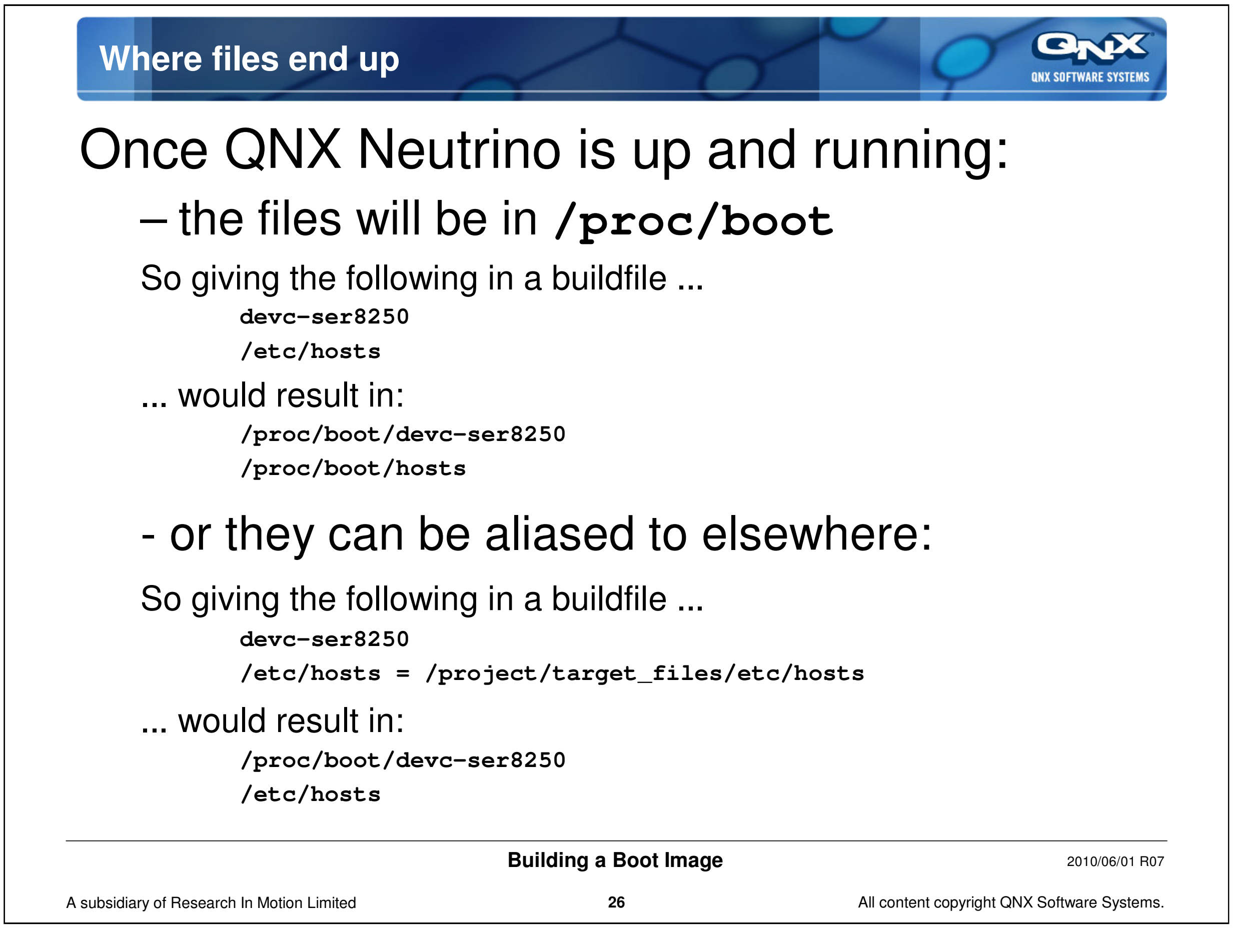
```
MKIFS_PATH=/usr/nto/x86/bin:
  /usr/nto/x86/sys:/usr/nto/x86/dll:
  /usr/nto/x86/lib:/project/bin
```

- using the **search** attribute for a particular file:

```
[search=/projecta/bin:/projectb/bin] myexec
```

NOTES:

On Windows hosted development systems, the path would use semicolons to separate the path components as a colon is a valid path character.



Where files end up

Once QNX Neutrino is up and running:

– the files will be in **`/proc/boot`**

So giving the following in a buildfile ...

```
devc-ser8250
/etc/hosts
```

... would result in:

```
/proc/boot/devc-ser8250
/proc/boot/hosts
```

- or they can be aliased to elsewhere:

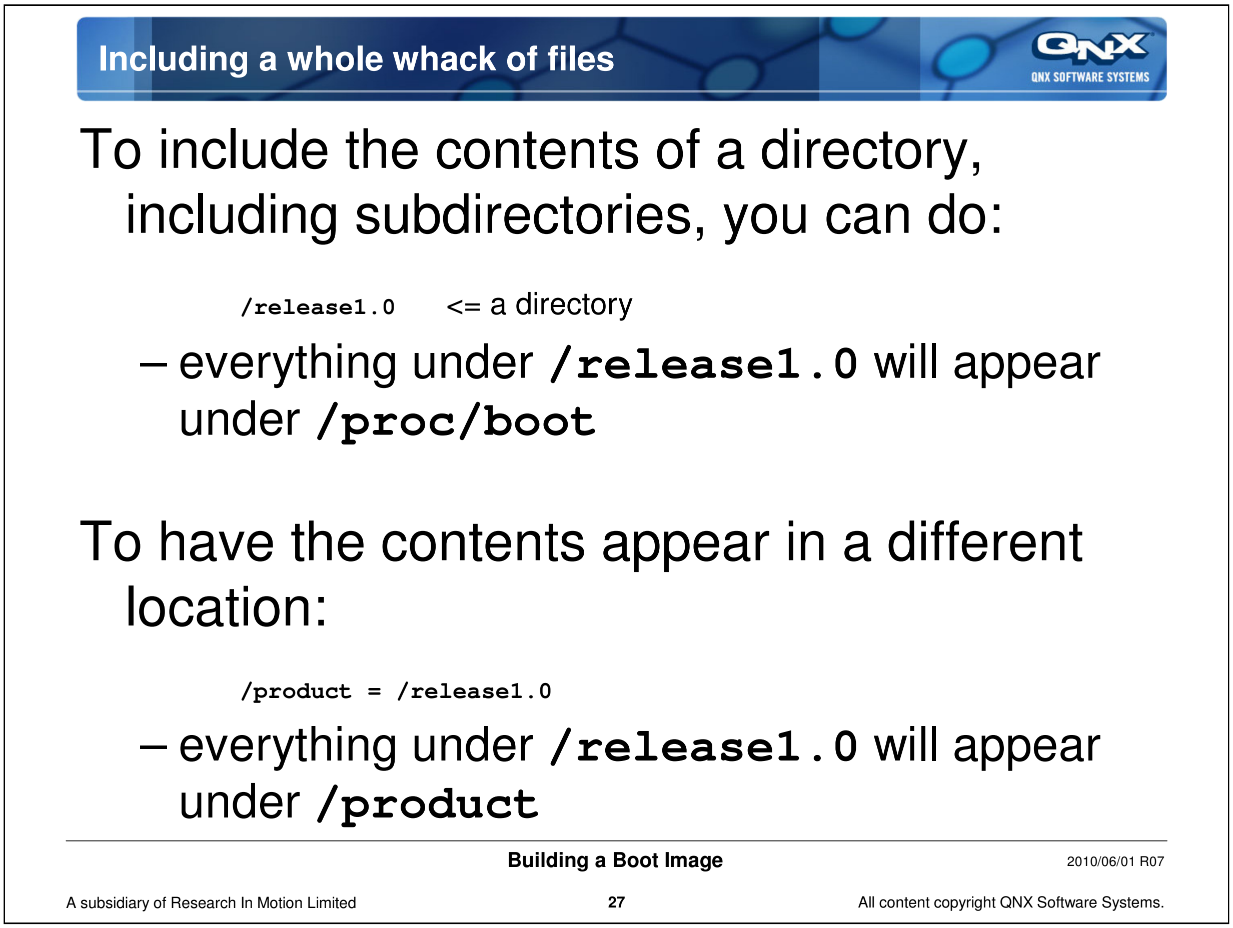
So giving the following in a buildfile ...

```
devc-ser8250
/etc/hosts = /project/target_files/etc/hosts
```

... would result in:

```
/proc/boot/devc-ser8250
/etc/hosts
```

NOTES:



Including a whole whack of files

QNX SOFTWARE SYSTEMS

To include the contents of a directory, including subdirectories, you can do:

```
/release1.0     <= a directory
```

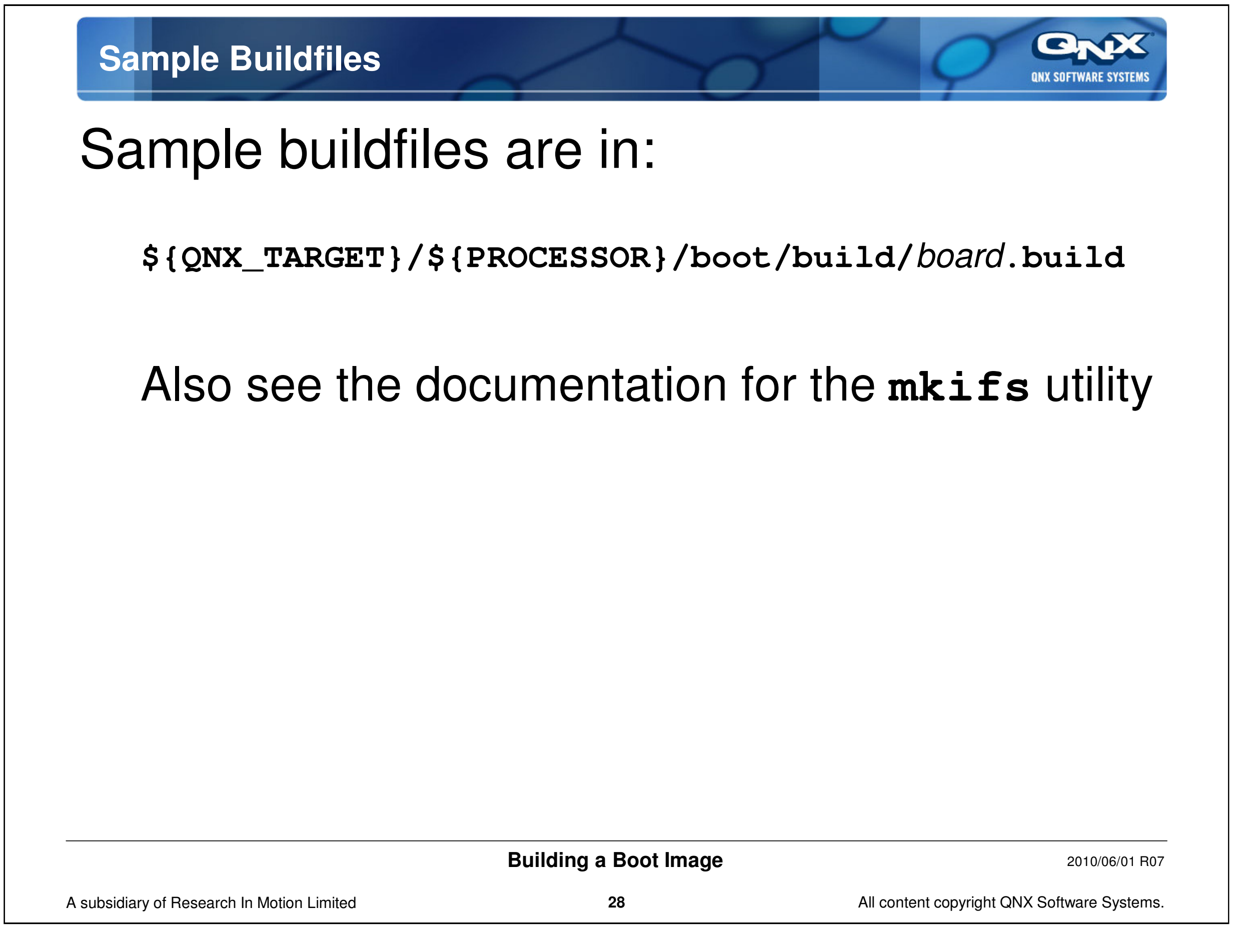
– everything under **`/release1.0`** will appear under **`/proc/boot`**

To have the contents appear in a different location:

```
/product = /release1.0
```

– everything under **`/release1.0`** will appear under **`/product`**

NOTES:



Sample Buildfiles

Sample buildfiles are in:

`${QNX_TARGET}/${PROCESSOR}/boot/build/`*board***`.build`**

Also see the documentation for the **`mkifs`** utility

NOTES:

Building a Boot Image

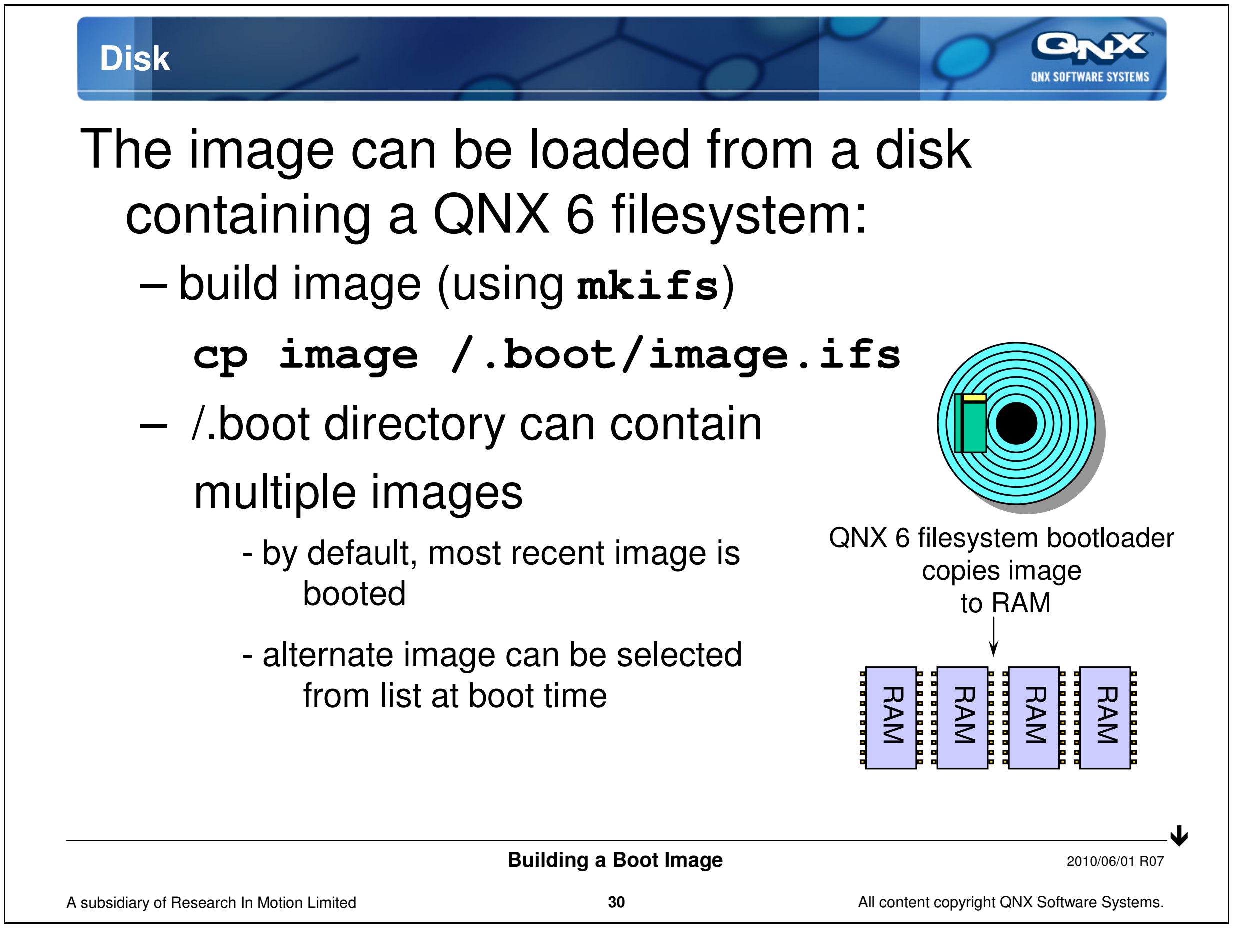
Topics:

Images & Buildfiles

→ **Loading**

Exercise

NOTES:



Disk

The image can be loaded from a disk containing a QNX 6 filesystem:

- build image (using **`mkifs`**)

  `cp image /.boot/image.ifs`

- /.boot directory can contain multiple images
  - by default, most recent image is booted
  - alternate image can be selected from list at boot time

QNX 6 filesystem bootloader copies image to RAM

RAM
RAM
RAM
RAM

NOTES:

For QNX 4 filesystems:

The image could also be loaded from a disk containing a QNX 4 filesystem:

–build image (using **`mkifs`**)

```
cp image /.boot
```

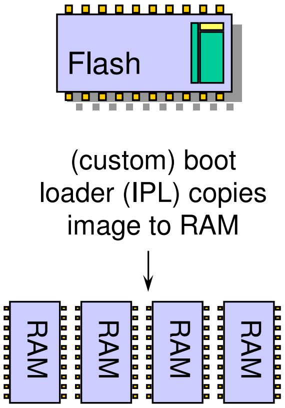
–can also be alternate image

```
cp image /.altboot
```

Pseudo-disk

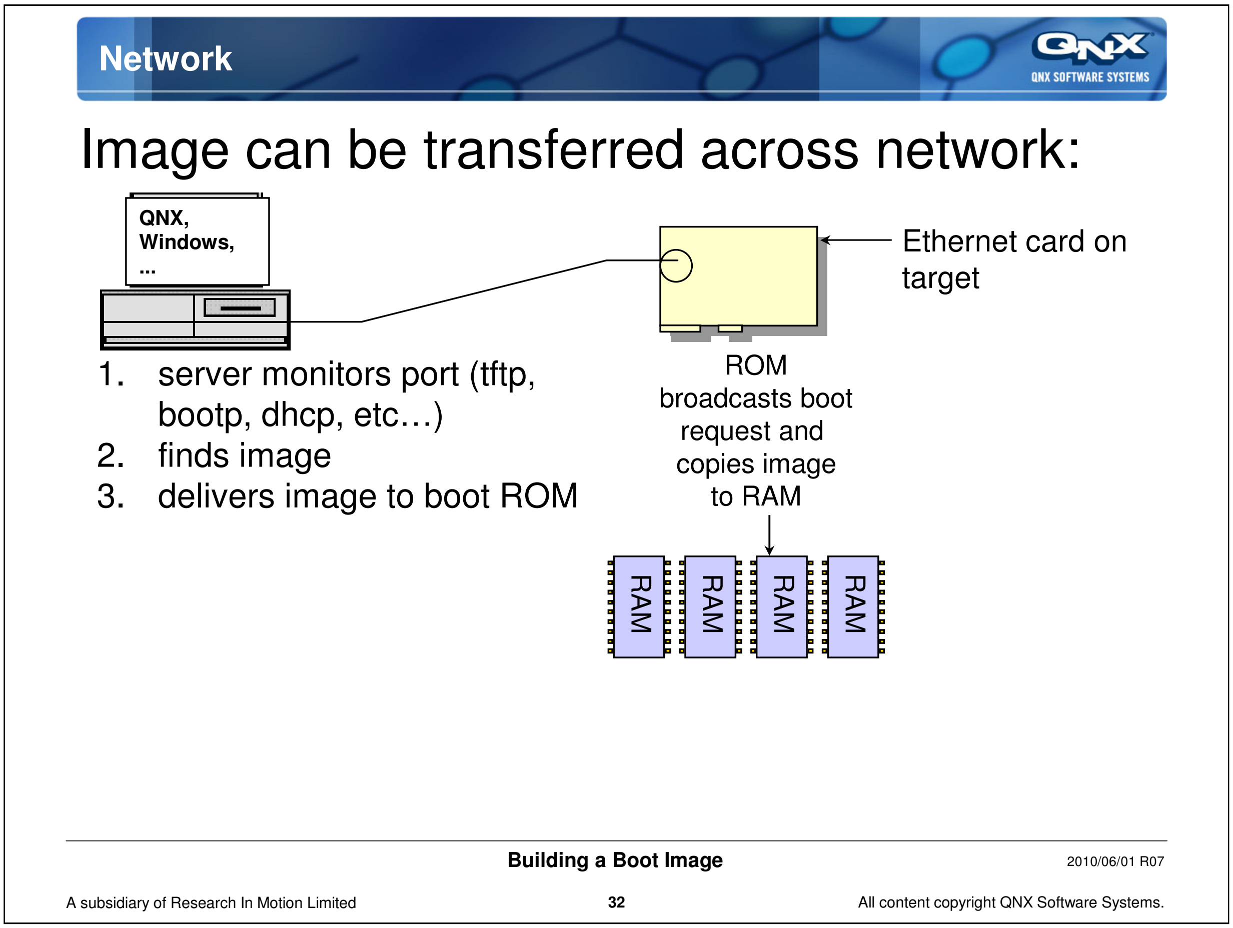
Image stored on pseudo-disk (Flash):

- custom (or manufacturer supplied) tools required



(custom) boot loader (IPL) copies image to RAM

NOTES:



Network

Image can be transferred across network:

QNX,
Windows,
…
RAM
RAM
RAM
RAM

Ethernet card on target

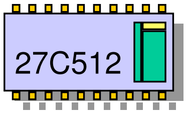
1. server monitors port (tftp, bootp, dhcp, etc…)
2. finds image
3. delivers image to boot ROM

ROM broadcasts boot request and copies image to RAM

NOTES:

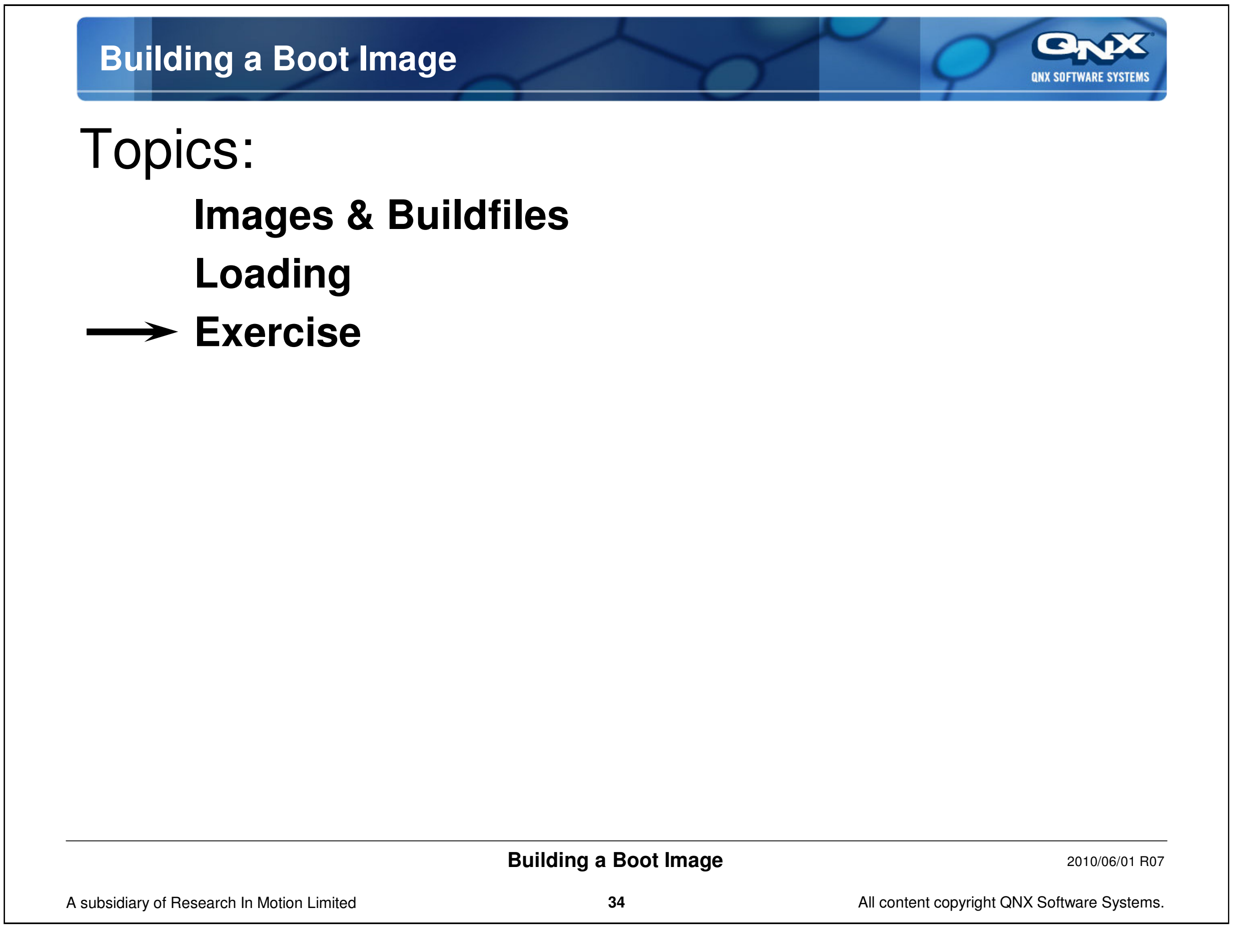
Linearly Mapped

Image can be in linearly mapped memory



no need to
copy image,
image is
directly accessible
(XIP)

NOTES:

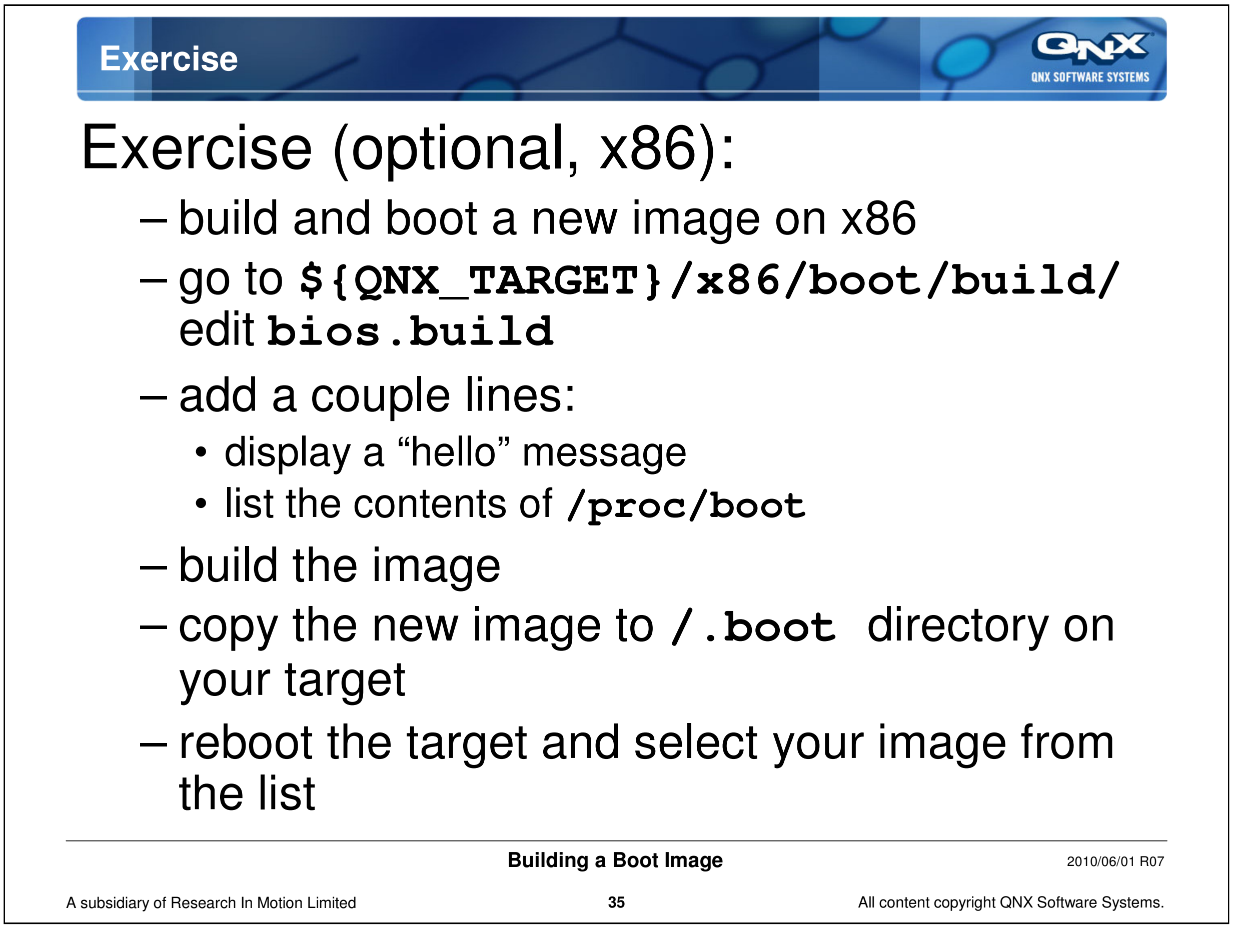


Building a Boot Image

Topics:

- **Images & Buildfiles**
- **Loading**
- → **Exercise**

NOTES:

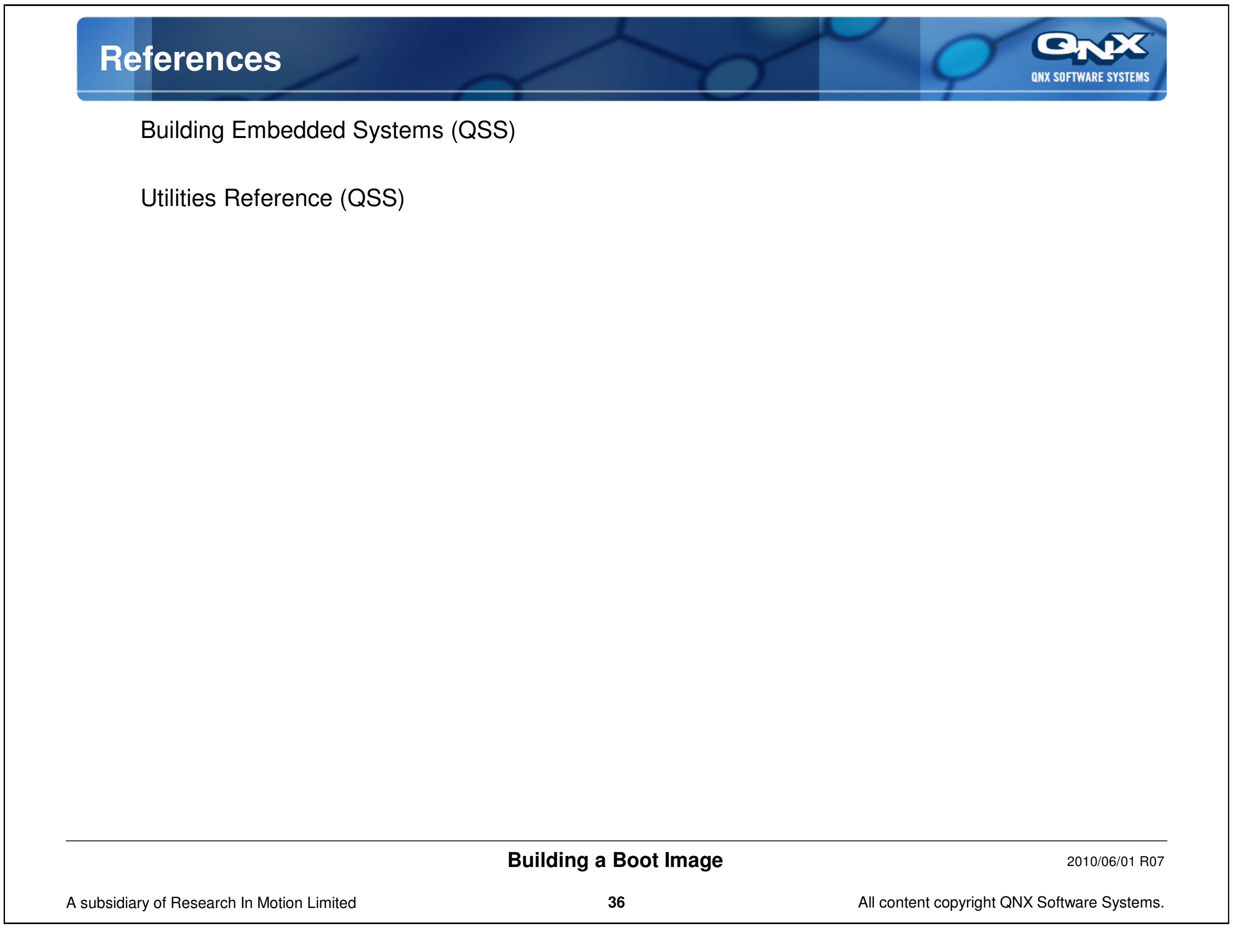


Exercise

Exercise (optional, x86):

- build and boot a new image on x86
- go to **`${QNX_TARGET}/x86/boot/build/`** edit **`bios.build`**
- add a couple lines:
  - display a “hello” message
  - list the contents of **`/proc/boot`**
- build the image
- copy the new image to **`/.boot`** directory on your target
- reboot the the target and select your image from the list

NOTES:



References

Building Embedded Systems (QSS)

Utilities Reference (QSS)

NOTES: