

Building and Customizing Target Images

©2014–2016, QNX Software Systems Limited, a subsidiary of BlackBerry Limited. All rights reserved.

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: <http://www.qnx.com/>

QNX, QNX CAR, Momentics, Neutrino, and Aviage are trademarks of BlackBerry Limited, which are registered and/or used in certain jurisdictions, and used under license by QNX Software Systems Limited. All other trademarks belong to their respective owners.

Electronic edition published: March 30, 2016

Contents

About This Guide	5
Typographical conventions.....	6
Technical support.....	8
Chapter 1: Target Images	9
Overview of an image, boot process, and filesystem layout.....	11
Chapter 2: Tools at a Glance	13
Chapter 3: Process to Generate an Image	15
Chapter 4: Build a Target Image	19
Sample image directory structure.....	22
Chapter 5: Image Configuration	25
Configuration file for <code>mksysimage.py</code>	26
Configuration file for <code>mkimage.py</code>	28
Calculating the size of target images and partitions.....	30
Target startup scripts.....	32
Chapter 6: Board Support Packages (BSPs)	33
Building a BSP.....	34
Chapter 7: Transferring an Image to a microSD card	35
Index	37

About This Guide

The *Building and Customizing Target Images* provides detailed information about how to modify OS images for the QNX CAR platform. The guide describes the detailed steps, the utilities used, and the resulting filesystem layout.

To find out about:	See:
Images, the boot process, and filesystems	Target Images
Tools for building images	Tools at a Glance
Familiarizing yourself with the process to create an image	Process to Generate an Image
How to build your own target image for the QNX CAR platform	Build a Target Image
Customizing a target image	Image Configuration
Calculating the image or partition size	Calculating the size of target images and partitions
BSPs	Board Support Packages (BSPs)
Transferring an image to your target	Transferring an Image to a microSD card

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	<code>if(stream == NULL)</code>
Command options	<code>-lR</code>
Commands	<code>make</code>
Constants	<code>NULL</code>
Data types	unsigned short
Environment variables	<i>PATH</i>
File and pathnames	/dev/null
Function names	<i>exit()</i>
Keyboard chords	Ctrl–Alt–Delete
Keyboard input	<code>Username</code>
Keyboard keys	Enter
Program output	<code>login:</code>
Variable names	<i>stdin</i>
Parameters	<i>parm1</i>
User-interface components	Navigator
Window title	Options

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** → **Show View**.

We use notes, cautions, and warnings to highlight important messages:



Notes point out something important or useful.



CAUTION: Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.



WARNING: Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (www.qnx.com).

You'll find a wide range of support options, including community forums.

Chapter 1

Target Images

When you complete a project, you must build a target image that your users can install on their target platform. You do this by packaging all the artifacts that make up the complete target image:

- Board Support Packages (BSPs)
- the core OS and utilities
- HTML5 apps
- Qt runtime
- HMI
- Browser/WebKit
- other binaries

Typically, you wouldn't build all of these components yourself. Instead, you have these prebuilt binaries available in your development environment when you install the QNX CAR platform.

Artifact Organization

When designing a system, you choose how to organize artifacts into one or more IFS files and partition images. Partition images are then combined to produce target images:

- *Image filesystems (IFS)* (or **.ifs** files) are created by `mkifs`. An IFS is a single binary that is loaded into RAM, generally on bootup by a bootloader or IPL. IFS files are usually quite minimal and contain only the drivers and libraries needed to start the kernel and mount additional partitions. Because an IFS is loaded into RAM and the files in it are more difficult to upgrade than files on a regular filesystem, an IFS is usually used only for startup code and a few key libraries, especially libraries that need to load early in the boot process to speed up boot time or improve performance. In the case of the QNX CAR platform, for example, everything required to start a minimal system and display the backup camera is stored in an IFS, but the HMI and apps are loaded from a storage device. The primary IFS is automatically mounted to **/proc/boot**.
- *Partition images* (or **.image** files) are created by `mkxfs`. These files contain the contents of a partition that are written to a storage device. A system might consist of a number of partitions. For example, an SD image for the QNX CAR platform contains 3 partitions: a FAT16 partition for booting and 2 QNX6 partitions (one for system data and another for user data). Partition images can contain a variety of file types including IFS files. For the QNX CAR platform, the primary IFS is stored in a FAT16 partition because most targets are capable of reading FAT16 with their default bootloader.
- Partition images are combined to produce *target images* (or **.img** files). A target image (also referred to as a disk image or system image) contains an entire target system—a partition table and the partition contents—and so is convenient to install. You can load a target image directly onto a storage medium such as an SD card without having to modify partition information.

System Generation

In general terms, to generate a system image for the QNX CAR platform, you perform the following steps:

1. Develop the **.build** files that contain the scripts and files used by `mkifs`.
2. Run `mkifs` to generate IFS files.
3. Determine all the files to include in each partition and develop **.build** files to be used by `mkxfs`.
4. Run `mkxfs` to generate all the partition images.
5. Run `diskimage` to generate the partition table and write all the partition images to a single **.img** file.

Because generating the various files is time-consuming and error-prone, this isn't the process we recommend. Instead, the QNX CAR platform includes an `mksysimage.py` script that handles the entire process of generating a system image. In fact, the `mksysimage.py` script supports multiple platforms and a number of configurations per platform. For example, in the case of OMAP5432, the `mksysimage.py` script produces images for SD card, EMMC, and EMMC + SATA SSD. For more information on `mksysimage.py`, see the following sections:

- [Process to Generate an Image](#)
- [Configuration file for `mksysimage.py`](#)
- [Build a Target Image](#)

Overview of an image, boot process, and filesystem layout

The steps that occur when the system starts up are as follows:

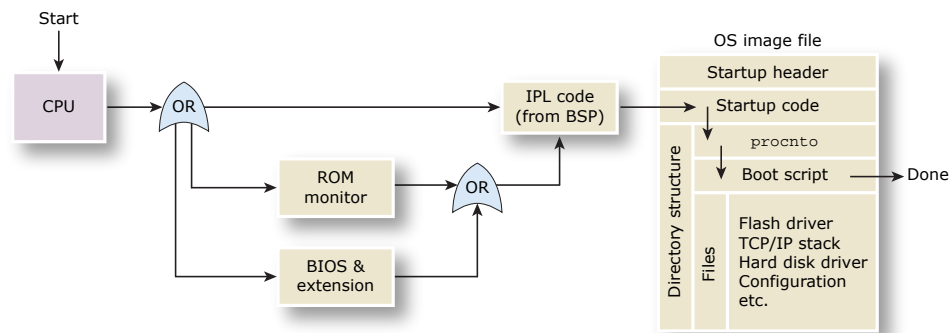
1. The processor begins executing at the **reset vector**.
2. The Initial Program Loader (**IPL**) locates the image filesystem (**IFS**) and transfers control to the startup program in the image.
3. The startup program configures the system and transfers control to the `procnto` module (combined microkernel and process manager).
4. The `procnto` module loads additional drivers and any application programs.

The **reset vector** is the address at which the processor begins executing instructions after the processor's reset line has been activated. On the x86, for example, this is the address `0xFFFFFFF0`.

The **IPL** minimally configures the hardware to create an environment that allows the startup program microkernel to run.

The IFS is a file that contains the OS, your executables, and any data files that might be related to your programs. The IFS contains a directory structure and some files.

To begin to create an image for your platform, you'll first need to understand the components of an image and the boot process. The following illustration shows the boot sequence.



You can also use the QNX Momentics IDE to create a System Builder Project to generate an image from scratch or to import an existing buildfile. For more information see “Build Images” in the *IDE User's Guide*.

When the bootup process starts, the CPU executes code at the reset vector, which could be a BIOS, ROM monitor, or an IPL. If it's a BIOS, then it'll find and jump to a BIOS extension (for example, a network boot ROM or disk controller ROM), which will load and jump to the next step. If it's a ROM monitor, typically `uboot`, then the ROM monitor jumps to the IPL code.

The IPL code does chip selects and sets up RAM, then jumps to the startup code. In either case, the next thing that runs is some startup code that sets up some hardware and prepares the environment for `procnto` to run.

The `procnto` module sets up the kernel and runs a boot script that contains drivers and other processes (which may include those you specify), and any additional commands for running anything else. The files included will be those as specified by the `mkifs` buildfile.

A buildfile specifies any files and commands to include in the image, the startup order for the executables, the loading options for the files and executables, as well as the command-line arguments and environment variables for the executables.

Chapter 2

Tools at a Glance

To you generate an image, you'll need to familiarize yourself with the following tools used in the image generation process.

More information about these tools is available in either the QNX CAR *System Services Reference* or the QNX Neutrino RTOS *Utilities Reference*.

Utility	Description
<code>gen-ifs</code>	Generates the various IFS files used by the QNX CAR platform. This tool concatenates multiple build file segments to generate the appropriate build file for the platform and variant being built. In addition, this tool sets the <code>MKIFS_PATH</code> variable as appropriate for the platform and variant.
<code>gen-osversion</code>	Generates a file (<code>/etc/os.version</code>) that contains relevant build and version details. This file is used by the software update process.
<code>mkifs</code>	Generates an IFS (Image FileSystem) from a <code>.build</code> file. The IFS is loaded on system startup.
<code>mkimage.py</code>	Processes <code>.tar</code> files and handles the generation of partition images (<code>.image</code> files) and target images (<code>.img</code> files) by calling <code>mkxfs</code> and <code>diskimage</code> .
<code>mksysimage.py</code>	A control script used to complete the imaging process, which includes the generation of the IFS, <code>mtar</code> archive, and the resulting target image file.
<code>mtar</code>	Creates a <code>.tar</code> file that contains all the files, directories, symbolic links (along with their permissions, user IDs and group IDs) used in a target image. The <code>mtar</code> utility uses a number of XML files (<code>filesset</code> files) to group files into sets that can be easily included or excluded.
<code>diskimage</code>	A binary file that combines all of the filesystem files generated by <code>mkxfs</code> (the <code>.ifs</code> files) into one file, the resulting image file.

Chapter 3

Process to Generate an Image

The process for generating a target image for the QNX CAR platform is described below.

Overall image generation process

The following illustration shows the process used to generate a QNX CAR platform target image:

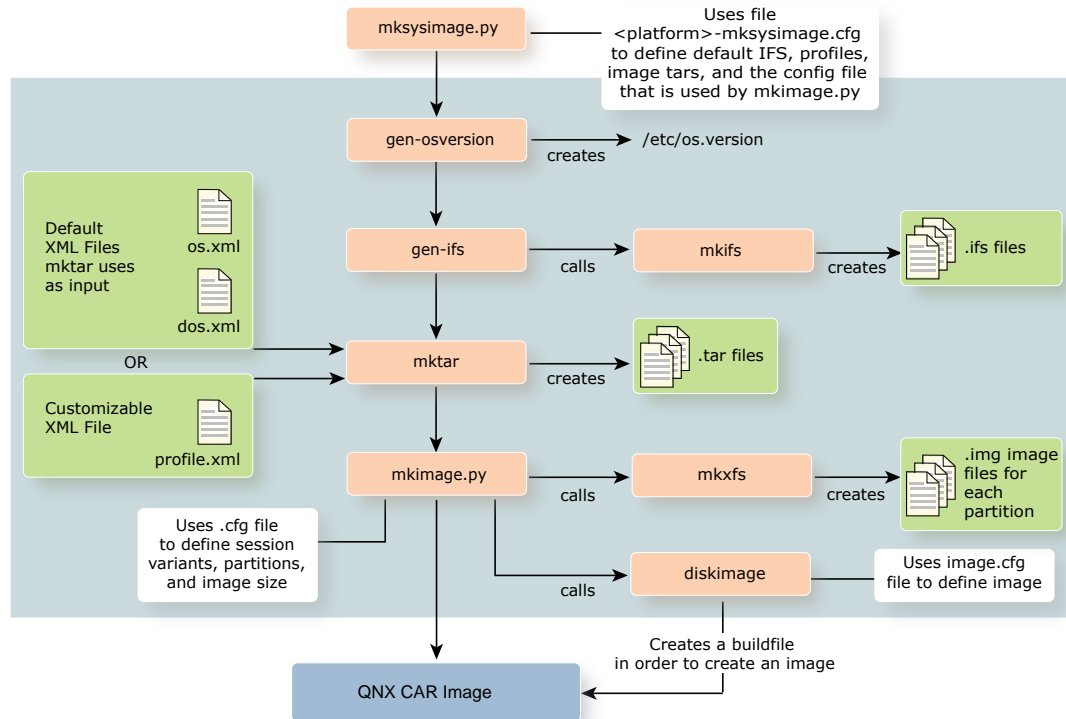


Figure 1: Image generation process for the QNX CAR platform

mksysimage.py

The first utility to run in the image-generation process is the `mksysimage.py` utility script. This Python script invokes other utilities to generate **tar** files and images for each platform. The script is located at:

- For Linux:
`$QNX_CAR_DEPLOYMENT/scripts/mksysimage.py`
- For Windows:
`%QNX_CAR_DEPLOYMENT%/scripts/mksysimage.py`

where `%QNX_CAR_DEPLOYMENT%` is `<install_location>/qnx660/deployment/qnx-car`.

By default, `mksysimage.py` reads a platform-specific configuration file from:

- For Linux:
`$ QNX_TARGET/<platform>/sd-boot/config/<platform>-mksysimage.cfg`
- For Windows:
`% QNX_TARGET%/<platform>/sd-boot/config/<platform>-mksysimage.cfg`

where **QNX_TARGET** is `<install_location>/qnx660/deployment`.

The configuration files for the supported platforms are:

- `imx61sabre-mksysimage.cfg`
- `jacinto5eco-mksysimage.cfg`
- `omap5uevm-mksysimage.cfg`

For detailed information about changing the values in a config file for `mksysimage.py`, see “[Configuration file for `mksysimage.py`](#)”.

These configuration files define which **tar** files and images are generated. The image variants for each platform are defined within this configuration file. By default, for each image variant, `mksysimage.py` generates two tar files and one image:

Filename	Description
<code><platform>-os.tar</code>	This tar file contains two QNX CAR2 filesystems that include all files except MLO and IFS files.
<code><platform>-dos-<image_variant>.tar</code>	This tar file contains a FAT16 filesystem that includes all bootup files, such as MLO and IFS files.

The generated image includes the two **tar** files indicated above. You can change the default configuration file, or specify your own by using the `-c` option in `mksysimage.py` to customize your **tar** files and images. For more information about this utility, see `mksysimage.py` in the *System Services Reference*. To see an example of the basic directory structure of an image generated for this release of the QNX CAR platform, see “[Sample image directory structure](#)”.

gen-osversion

The `gen-osversion` utility generates the `/etc/os.version` file based on the specified build environments. For more information about this utility, see `gen-osversion` in the *System Services Reference*.

gen-ifs

The `gen-ifs` utility calls the `mkifs` utility to create the `.ifs` file(s) that are included in the final target image. An IFS is a bootable image filesystem that contains the `procnto` module, your boot script, and possibly other components such as drivers and shared objects. For more information about this utility, see `gen-ifs` in the *System Services Reference*.

mk`tar`

The `mktar` utility creates a **tar** file containing the filesystem for a specified variant. The result from running this utility is a file called ***platform.tar*** that contains the QNX CAR filesystem for the specified platform variant. This resulting file will be included in the QNX CAR2 image. In order to run, the `mktar` utility will require binary content from the installed QNX CAR2 SDP. As input, the `mktar` utility uses the ***dos-variant.xml*** and ***os.xml*** files; otherwise, it uses the default ***profile.xml*** file.

mk`image.py`

The `mkimage` utility builds an image from each partition called ***<partition_name>.image***. The QNX CAR-specific Python script ***mk`image.py`*** uses a configuration file (*<platform>-<variant>.cfg*) to define session variants, partitions, and image size:

- The `mkimage` utility progresses and parses the command line, places the bootable image file(s) first in the resulting output file, followed by embedded filesystem files, and any other files that were on the command line.
- The `mkimage.py` script uses `mkxfs` (see `mkkifs` and `mkefs`) to create the image files (***.image*** files) for each partition specified in the `mkimage` configuration file. The `diskimage` utility creates the final image that combines all of the partition image files (***<partition_name>.image***) into a single image.

Final image

The final step in the image-generation process for `mksysimage.py` is the creation of the OS image (***.img***) for the platform. The resulting **tar** file will also be located in the same output directory as the image.

Chapter 4

Build a Target Image

To build your own target image:

1. Set up the environment variables for the QNX development environment:

- For Linux, enter the following command:

```
# source install_location/qnx660-env.sh
```

where by default the *install_location* is **\$HOME/qnx660/**.

- For Windows, enter the following command:

```
install_location\qnx660-env.bat
```

where by default the *install_location* is **C:\qnx660**.

As part of the installation process for the QNX CAR platform, a workspace was created for you that contains the scripts and configuration files you'll be using. These files are located in the following locations:

- Scripts:
 - For Linux: ***\$QNX_CAR_DEPLOYMENT*/deployment/scripts/**
 - For Windows: ***%QNX_CAR_DEPLOYMENT%*\deployment\scripts**
where *QNX_CAR_DEPLOYMENT* is ***install_location*/qnx660/deployment/qnx-car/**.
- Configuration files:
 - For Linux: ***\$QNX_CAR_DEPLOYMENT*/boards/<platform>/etc/**
 - For Windows: ***%QNX_CAR_DEPLOYMENT%*\boards\<platform>\etc**

2. Extract a BSP. For detailed instructions, see “[Building a BSP](#)”.

3. Create an output directory where you want to have the image generated.



You must specify a valid directory name; the directory must exist prior to running the `mksysimage.py` script, otherwise the image won't be generated.

4. To generate a target system image, run the appropriate `mksysimage.py` command.

- For Linux, enter the following command:

```
mksysimage.sh -vvvvvvv -o outputPath platform.external
```

- For Windows, enter the following command:

```
mksysimage.bat -vvvvvvv -o outputPath platform.external
```

where *outputPath* is the location for the new system image. If this directory isn't empty, run `mksysimage.py` with the `-f` option (`mksysimage.py` won't overwrite existing system images unless `-f` is specified). Specify a filename according to your *platform* as shown in the following table:

Platform	Filename
Texas Instruments OMAP5432 EVM	omap5uevm.external
Texas Instruments J5 ECO EVM811x EVM	jacinto5eco.external
Freescale i.MX6Q SABRE Lite	imx61sabre.external

The `mksysimage.py` utility generates images for various configurations. For example, for SABRE Lite, image files are created for SD and SD/SATA:

- `imx61sabre-dos-sd-sata.tar`
- `imx61sabre-dos-sd.tar`
- `imx61sabre-os.tar`
- `imx61sabre-sd-sata.img`
- `imx61sabre-sd.img`

The following output shows the results from creating an image for the OMAP5432 board:

```
[info]: Generating os.version file...
[info] gen-osversion.py: Generated os.version file: [C:\Users\Public\repos\trunk\deployment\qnx-car\
scripts\...\..\target\etc\os.version]
date: Thu Oct 10 12:06:07 2013
project: Local Build
buildHost: WIN-M9DICV29QL6
buildID: Local Build
buildNum: Local Build

platform: omap5uevm.external
car2Branch: trunk
car2Rev: 7244
externalBranch: mainline
externalRev: 3171

[info]: Generating qnxcar/system/info PPS file...
[info]: Generating IFS [omap5-sd.ifs]...
** Building omap5-sd.ifs...
** Building omap5-emmc.ifs...
** Building omap5-emmc-sata.ifs...
** Building etc/secondary.ifs...
[info]: Copying Boot IFS: [omap5-sd.ifs] to qnx-ifs...
[info]: Generating Archive...
Locating files...
Writing tar file...
Locating files...
Writing tar file...
[info]: Generating Image...
Open tar: C:\myWork\myimage\omapExample\omap5uevm-os.tar
Open tar: C:\myWork\myimage\omapExample\omap5uevm-dos-sd.tar
Created temporary directory c:\users\admini~1\appdata\local\temp\tmpoklciy
data <-- accounts [dir]
.
.
.

base <-- /etc/system/hmi-notification/policy.cfg [file]
.
.
.

data <-- appinfo/menuentry/ApiDemos.bar [file]
data <-- var/pps/system/installer/upd/current/job.testEM1163ejTBOmqnEnAHLA6AA [file]
boot <-- /MLO [file]
boot <-- /qnx-ifs [file]
invoking: mkxfs -t qnx6fsimg c:\users\admini~1\appdata\local\temp\tmpo0ydt1\data.build c:\users\admini~1\appdata\local\temp\tmpo0ydt1\data.image
invoking: mkxfs -t qnx6fsimg c:\users\admini~1\appdata\local\temp\tmpo0ydt1/base.build c:\users\admini~1\appdata\local\temp\tmpo0ydt1/base.image
invoking: mkxfs -t fatfsimg c:\users\admini~1\appdata\local\temp\tmpo0ydt1/boot.build c:\users\admini~1\appdata\local\temp\tmpo0ydt1/boot.image
```

```
invoking: diskimage -c c:\users\admini~1\appdata\local\temp\tmpo0ydt1\image.cfg -o C:\myWork\myimage\
omapExample\omap5uevm-emmc.img
Removed temporary directory c:\users\admini~1\appdata\local\temp\tmpo0ydt1

Disk image created at C:\myWork\myimage\omapExample\omap5uevm-emmc.img
[info]: Generating os.version file...
[info] gen-osversion.py: Generated os.version file: [C:\Users\Public\repos\trunk\deployment\qnx-car\
deployment\scripts\...\target\etc\os.version]
date: Thu Oct 10 12:56:47 2013
project: Local Build
buildHost: WIN-M9DICV29QL6
buildID: Local Build
buildNum: Local Build

platform: omap5uevm.external
car2Branch: trunk
car2Rev: 7244
externalBranch: mainline
externalRev: 3171

[info]: Generating qnxcar/system/info PPS file...
[info]: Generating IFS [omap5-emmc-sata.ifs]...
** Building omap5-sd.ifs...
** Building omap5-emmc.ifs...
** Building omap5-emmc-sata.ifs...
** Building etc/secondary.ifs...
[info]: Copying Boot IFS: [omap5-emmc-sata.ifs] to qnx-ifs...
[info]: Generating Archive...
[warning]: Tar file already exists. (Use --force to force overwriting): [C:\myWork\myimage\
omapExample\omap5uevm-os.tar].
Locating files...
Writing tar file...
[info]: Generating Image...
DONE
WARNING: DISKIMAGE ONLY SUPPORT LITTLE ENDIAN.
Open tar: C:\myWork\myimage\omapExample\omap5uevm-dos-emmc-sata.tar
Created temporary directory c:\users\admini~1\appdata\local\temp\tmprozsm_
boot <-- /MLO [file]
boot <-- /qnx-ifs [file]
invoking: mkxfs -t fatfsimg c:\users\admini~1\appdata\local\temp\tmprozsm_/boot.build c:\users\admini~1\appdata\local\temp\tmprozsm_/boot.image
invoking: diskimage -c c:\users\admini~1\appdata\local\temp\tmprozsm_/image.cfg -o C:\myWork\
myimage\omapExample\omap5uevm-emmc-sata.img
Removed temporary directory c:\users\admini~1\appdata\local\temp\tmprozsm_

Disk image created at C:\myWork\myimage\omapExample\omap5uevm-emmc-sata.img
```

Sample image directory structure

The following output shows the first three levels of a directory structure for a basic image:

```
.
./html5
./html5/common
./html5/common/fonts
./html5/common/img
./html5/common/js
./html5/common/themes
./html5/common/ui-framework
./html5/tools
./html5/tools/arm
./html5/tools/SenchaCMD
./html5/webworks
./html5/webworks/apps
./html5/webworks/apps-legacy
./html5/webworks/apps-legacy/CarControl/modules
./html5/webworks/tools
./qt
./qt/src
./qt/src/hmi
./target
./target/accounts
./target/accounts/1000
./target/appinstall
./target/appinstall/bars
./target/boards
./target/boards/armle-v7
./target/boards/common
./target/boards/imx61sabre
./target/boards/imx61sabre.external
./target/boards/imx61sabre/
./target/boards/jacinto5eco
./target/boards/jacinto5eco.external
./target/boards/jacinto5eco.external/
./target/boards/jacinto5eco/armle-v7
./target/boards/jacinto5eco/armle-v7/bin
./target/boards/jacinto5eco/armle-v7/boot
./target/boards/jacinto5eco/armle-v7/boot/build
./target/boards/jacinto5eco/armle-v7/boot/sys
./target/boards/jacinto5eco/armle-v7/lib
./target/boards/jacinto5eco/armle-v7/lib/dll
./target/boards/jacinto5eco/armle-v7/sbin
./target/boards/jacinto5eco/armle-v7/usr
./target/boards/jacinto5eco/armle-v7/usr/bin
./target/boards/jacinto5eco/armle-v7/usr/lib
./target/boards/jacinto5eco/armle-v7/usr/lib/graphics
./target/boards/jacinto5eco/armle-v7/usr/lib/graphics/jacinto5
./target/boards/jacinto5eco/
./target/boards/omap5uevm
```

```
./target/boards/omap5uevm.external
./target/boards/omap5uevm.external/
./target/boards/omap5uevm/
./target/deployment
./target/etc
./target/opt
./target/opt/asr
./target/opt/webkit
./target/opt/webkit/config
./target/opt/webkit/config/webkit
./target/root
./target/runtime-external
./target/scripts
./target/scripts/startup-support
./target/scripts/vncdiscovery
./target/usr
./target/usr/hmi
./target/usr/mlink
./target/usr/sbin
./target/usr/share
./target/var
./target/var/certificates
./target/var/certmgr
./target/var/db
./target/var/etc
./target/var/pps
./target/var/tmp
./target/var/twonky
./target/var/usb
./target/var/usb/usbd/services
./target/var/webplatform
```


Chapter 5

Image Configuration

When you create your own OS image for your platform, you can modify various customizable options in the configuration files associated with the [mksysimage.py](#) and [mkimage](#) utilities. These files enable you to define files for the system image for a specific platform type and provide size and partition information.

Configuration file for `mksysimage.py`

A configuration file for `mksysimage.py` defines the components for a specific platform type.



For information about running the `mksysimage.py` Python script utility, see `mksysimage.py` in the *System Services Reference*.

The components defined by a `mksysimage.py` configuration file are:

- Specific IFS file that's renamed to **qnx-ifs** and is used as the default boot file
- The **tar** files to generate
- The **tar** files to include in the image
- Configuration file used to define the size of partitions for the image

You can find the default configuration file at

`%QNX_CAR_DEPLOYMENT%/<board>/<platform>/sd-boot/config/platform-mksysimage.cfg` where `%QNX_CAR_DEPLOYMENT%` is `<install_location>/qnx660/deployment/qnx-car`. For example, the default configuration file for the OMAP5432 board is as follows:

```
[sd]
default-ifs=omap5-sd.ifs
profiles=os.xml,dos-sd.xml
image-tars=omap5uevm-os.tar,omap5uevm-dos-sd.tar
image-config=omap5uevm-sd.cfg

[emmc]
default-ifs=omap5-emmc.ifs
profiles=os.xml,dos-emmc.xml
image-tars=omap5uevm-os.tar,omap5uevm-dos-emmc.tar
image-config=omap5uevm-emmc.cfg

[emmc-sata]
default-ifs=omap5-emmc-sata.ifs
profiles=os.xml,dos-emmc-sata.xml
image-tars=omap5uevm-dos-emmc-sata.tar
image-config=omap5uevm-emmc-sata.cfg
```

The contents of this particular configuration file reveal that this OMAP5432 board has three image variants called `sd`, `emmc`, and `emmc-sata`, and each of these image variants defines the following:

Item	Description
<code>default-ifs</code>	The <code>ifs</code> image name used as the default bootup IFS (<code>qnx-ifs</code>).
<code>profiles</code>	The <code>mktar</code> profiles used to generate tar files. The <code>image-tars</code> elements generated tar files are <code><platform>-<profile_name>.tar</code> .
<code>image-tars</code>	The tar files included in the image.

Item	Description
<code>image-config</code>	The configuration file used for specifying the size of each partition in the res in the <code>%QNX_CAR_DEPLOYMENT%/<board>/<platform>/sd-boot/config/</code> dire

Configuration file for `mkimage.py`

The `mkimage.py` Python script utility takes as input a configuration file that provides image information.



For information about running the `mkimage.py` Python script utility, see `mkimage.py` in the *System Services Reference*.

The configuration file used by `mkimage.py` provides the following information:

- maximum size of the image
- size and number of partitions, to a maximum of four
- order of partitions
- type of partition
- path to the partition

For example, the following file shows the contents for the `jacinto5eco-sd.cfg` configuration file:


```
[disk]
heads=64
sectors_per_track=32
cylinders=3724
sector_size=512

[boot]
path=/dos
type=12
num_sectors=1048576
order=1

[base]
path=/base
type=179
num_sectors=1048576
order=3

[data]
path=/
type=178
num_sectors=4194304
order=4
```

Type	Description
[disk]	This section doesn't specify a partition, but rather partitions. This section is required, must not be empty, and must be the first section in the [disk].

Type	Description
<code>heads</code>	The number of heads for the data medium used.
<code>sectors_per_track</code>	The number of sectors for each track for the data medium.
<code>cylinders</code>	The number of cylinders for the data medium.
<code>sector_size</code>	The size of the sectors used to store the data.
<code>[partition_name]</code>	A partition in the image. In the example above, however, these partition names can be any name.
<code>path</code>	Identifies the path for the partition.
<code>type</code>	Represents the identifier for the type of partition. See the <i>System Architecture</i> guide for QNX Neutrino for more information.
<code>num_sectors</code>	The number of sectors for the partition.
<code>order</code>	The order for the specified partition in the image.  If the order is 1, then it's the bootable partition.

The example above shows that there are three partitions:

- `[boot]` is of type 12 (FAT), has a partition order of 1 (meaning the first partition in the image) located at `/dos`. The configuration file used with `mksysimage.py` will indicate that this first partition is the boot **ifs** and that the **ifs** file will be renamed to **qnx-ifs**.
- `[base]` is of type 179 (QNX 6.x), has a partition order of 3, and is located at `/base`.
- `[data]` is of type 178 (QNX 6.x), has a partition order of 4, and is located at the root `/`.

Calculating the size of target images and partitions

To customize the size of an image or partition, you need to modify the following variant-specific configuration file:

```
%QNX_CAR_DEPLOYMENT%/<board>/<platform>/  
sd-boot/config/<platform>-<image_variant>.cfg
```

Example

The configuration file for `omap5uevm-emmc.cfg` is as follows:

```
[disk]  
heads=64  
sectors_per_track=32  
cylinders=3724  
sector_size=512  
  
[boot]  
path=/dos  
type=12  
num_sectors=1048576  
order=1  
  
[base]  
path=/base  
type=179  
num_sectors=1048576  
order=3  
  
[data]  
path=/  
type=178  
num_sectors=4194304  
order=4
```

The `[boot]` section in the configuration file specifies the first partition. A target image can support at most four partitions.

Calculating the maximum size of a target image

To calculate the total size of the image, you must multiply the values given in the `[disk]` section of the configuration file.



The `disk` section doesn't specify a partition; it provides important size information and must appear at the top of the configuration file, before any partitions are specified.

```
heads  
x sectors_per_track
```

```

x cylinders
x sector_size
-----
total maximum size of image

```

Therefore, for the OMAP5432 example for the `emmc` variant above, the maximum size of the image would be 3.9 GB (3.63 GB actual) and would be calculated as follows:

```

64 heads
x 32 sectors_per_track
x 3724 cylinders
x 512 sector_size
-----
3904897024 bytes for a total of 3.63 GB for the total maximum size
of the image

```



Limitations:

- The total size of all partitions can't exceed the total size of the image.
- The maximum number of `heads` is 255.
- The maximum number of `sectors_per_track` is 63.

Calculating the size of a partition

To calculate the size of a partition in the example above:

```

heads x sectors_per_track x cylinders = number_of_sectors
number_of_sectors x sector_size = partition size

64 x 32 x 3724 = 7626752
7626752 x 512 = 3904897024 bytes

```

Therefore, the size of this specific partition is 3724 MB.

Target startup scripts

Buildfiles let you incorporate scripts to be run on your target. The `[+script]` attribute in the buildfile tells `mkifs` that the specified file is a script file, which is a sequence of commands that you want `procnto` to execute when it's completed its own startup. Script files look like regular shell scripts, except that:

- you can position special modifiers before the actual commands you want to run
- `mkifs` parses the script file's contents before placing them into the image

To run a command, its executable must be available when the script is executed. You can add the executable to the image or get it from a filesystem that's started before the executable is required. The latter approach results in a smaller image.

For more information about script files, see “The script file” in the *Building Embedded Systems User's Guide* and `mkifs` in the *Utilities Reference*.

System Launch and Monitor (SLM)

The SLM service automates process management. The SLM process starts early in the boot sequence to launch complex applications consisting of many processes that must start in a certain order. The configuration file lists all the processes for SLM to manage, any dependencies between the processes, the commands for launching the processes, and other properties.

For more information about configuring SLM, see “System Launch and Monitor (SLM)” in the *System Services Reference*.

Chapter 6

Board Support Packages (BSPs)

After you install the QNX OS, you can download any processor-specific Board Support Package (BSP) from our website, <http://community.qnx.com/sf/sfmain/do/viewProject/projects.bsp>. The BSPs are designed to help you get the QNX OS running on various supported platforms. To use a BSP, you must either unzip the archive and build it on the command line or import it into the IDE.

To become more familiar with BSPs, see “Working with a BSP” in the *Building Embedded Systems* guide.

A BSP typically includes an Initial Program Loader (IPL), a startup program, a default buildfile, networking support, board-specific device drivers, system managers, utilities, and so on.

Building a BSP

The QNX CAR platform includes BSPs for these reference boards:

- Texas Instruments OMAP5432 EVM
- Texas Instruments J5 ECO EVM811x EVM
- Freescale i.MX6Q SABRE Lite

To get these BSPs, go to the following location on the QNX download site:

<http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/BSPAndDrivers>

Then, follow the included instructions for unzipping the BSP onto your host system.

To build the BSP for your board:

1. In the directory where you unzipped the BSP, enter this command:

```
# make
```

2. Copy the resulting BSP binaries to your QNX CAR workspace.



In the reference board BSPs, the **prebuilt** directory contains all that's needed for the image, so you can copy the binaries from there. But if you changed any BSP content and are rebuilding the image, the binaries need to come from the **install** directory, meaning the copy commands shown below should be modified to refer to this other directory.

- For Linux:

Board	Copy command
OMAP5432	<code>cp -r prebuilt/armle-v7 \$QNX_CAR_DEPLOYMENT/boards/omap5uevm</code>
J5 ECO	<code>cp -r prebuilt/armle-v7 \$QNX_CAR_DEPLOYMENT/boards/jacinto5eco</code>
SABRE Lite	<code>cp -r prebuilt/armle-v7 \$QNX_CAR_DEPLOYMENT/boards/imx6lsabre</code>

where `$QNX_CAR_DEPLOYMENT` is `<install_location>/qnx660/deployment/qnx-car`.

- For Windows:

Board	Copy command
OMAP5432	<code>cp -r prebuilt\armle-v7 %QNX_CAR_DEPLOYMENT%\boards\omap5uevm</code>
J5 ECO	<code>cp -r prebuilt\armle-v7 %QNX_CAR_DEPLOYMENT%\boards\jacinto5eco</code>
SABRE Lite	<code>cp -r prebuilt\armle-v7 %QNX_CAR_DEPLOYMENT%\boards\imx6lsabre</code>

where `%QNX_CAR_DEPLOYMENT%` is `<install_location>\qnx660\deployment\qnx-car`.

To see the basic directory structure for an image generated for this platform release, see “[Sample image directory structure](#)”.

Chapter 7

Transferring an Image to a microSD card

For QNX CAR 2.1, a 4GB Class 4 SD card is the minimum requirement, but we recommend Class 10 cards. To identify this SD card type, look on the card for a “C” character with a number inside the character. Cards without a class indication are Class 0—don't use them.



The 16GB SanDisk Ultra[®] microSDHC[™] UHS-I SD cards have been tested with the QNX CAR platform and are recommended.

To transfer a QNX CAR image to a microSD card for your target, follow these OS-specific steps:

Host OS	Command / instructions
Linux	<pre>sudo dd bs=1048576 if=<i>your_image</i> of=/dev/<i>sdx</i></pre> <p>This command causes the <code>dd</code> utility to write 1MB chunks of data to the disk at a time. In addition, the command assumes that <code>sdx</code> is the SD card.</p> <hr/> <p> The device name shouldn't include a partition suffix. For example, don't use <code>/dev/sdx1</code>. However, the device name can be <code>/dev/mmcblk0</code> on some Linux distributions.</p>
Windows	<ol style="list-style-type: none">1. Download and install Win32 Disk Imager from this site: http://sourceforge.net/projects/win32diskimager/2. Using Win32 Disk Imager, write the <code>.img</code> file to your microSD card.

Now, boot the image on your target board. The instructions vary with the board type:

Freescale i.MX6Q SABRE Lite board

1. Insert the SD card you prepared earlier into the *lower slot* (i.e., the large, full-size SD card slot) on the SABRE Lite board.
2. Press the **Reset** button.
3. Interrupt the countdown by pressing any key during the first boot of the system.
4. Enter the following commands to prepare the *U-Boot* environment variables for booting the QNX CAR system:

```
MX6Q SABRELITE U-Boot> setenv sdslot '0'  
MX6Q SABRELITE U-Boot> setenv loadaddr '0x10800000'  
MX6Q SABRELITE U-Boot> setenv bootifs 'qnx-ifs'  
MX6Q SABRELITE U-Boot> setenv bootcmd_fatload 'mmc dev ${sdslot}; fatload mmc ${sdslot}:1\  
  ${loadaddr} ${bootifs}; go ${loadaddr}'  
MX6Q SABRELITE U-Boot> setenv bootcmd 'run bootcmd_fatload'  
MX6Q SABRELITE U-Boot> saveenv
```

5. Type `boot` or press the **Reset** button.

This action restarts the QNX CAR system.

6. Program the QNX IPL onto your board.

After the QNX CAR system starts and the HMI comes up, flash the QNX IPL. At the console prompt on your target, issue the following command:

```
# ipl-update-imx6.sh /dos/ipl-mx6q-sabrelite.bin
```

This step prevents issues with older boot loaders from occurring on the i.MX6Q SABRE Lite board. The **ipl-update-imx6.sh** script replaces the existing boot loader with the QNX boot loader. The next time you boot, the QNX IPL will load the QNX CAR system.

All other board types

1. Insert the SD card into your target board and power on the board.

The board automatically boots and prompts you to calibrate your screen.

2. Follow the instructions to calibrate the screen.

After you do this, the HMI appears on your display.



You can configure your reference board for boot optimization, as explained in the *Boot Optimization Guide*.

Index

[disk] 28

B

Board Support Packages, *See* BSPs

boot process 11

bootup process 11

BSP 9, 34

 building for your board 34

BSPs 33

 content 33

 obtaining 33

build 19

 image 19

buildfile 12

C

configure 25–26, 28, 30

 image 25

 image partitions 30

 mkimage.py 28

 mksysimage.py 26

 variants 26

cylinders 29

F

filesystem 11

 layout 11

G

gen-ifs 13, 16

 image generation process 16

gen-osversion 13, 16

generate 15

 image 15

generate process 16–17

 gen-ifs 16

 mkimage 17

 mksysimage.py 17

 mktar 17

H

heads 29

HMI 9

HTML5 apps 9

I

ifs 26

image 9, 11–13, 15–16, 19, 22, 25–28, 30–31

 artifacts 9

 binaries 9

 Browser 9

 build 19

 buildfile 12

 calculate image size 30

 calculate partition size 31

 configuration 25

 core OS 9

 customize partition size 30

 directory structure (example) 22

 filesystem layout 11

 gen-ifs script 13

 gen-osversion script 13

 generation process 16

 HMI 9

 HTML5 apps 9

 ifs 26

 image-config 27

 image-tars 26

 maximum size 28

 mkifs utility 13

 mkimage script 13

 mksysimage script 13

 mktar script 13

 number of partitions 28

 order of partitions 28

 partition size 28

 partition type 28

 path to partition 28

 process to generate 15

 profiles 26

 Qt runtime 9

 startup 11

 target 9

 variants 26

 WebKit 9

image-config 27

image-tars 26
Initial Program Loader, *See* IPL
IPL 11
IPL code 11

M

mkifs 13
mkimage 13, 17, 28
 image generation process 17
mkimage.py 17, 28–29
 configure 28
 cylinders 29
 example 28
 heads 29
 maximum size 28
 num_sectors 29
 number of partitions 28
 order 29
 partition order 28
 partition size 28
 partition type 28
 partition_name 29
 path 29
 path to partition order 28
 Python script 17
 sector_size 29
 sectors_per_track 29
 type 29
mkysimage.py 13, 15–17, 26
 configure 26
 gen-ifs 13
 gen-ifs overview 16
 gen-osversion 13
 image generation process 17
 mkifs 13
 mkimage 13
 mkimage overview 17
 mktar 13
 mktar overview 17
 Python script 15
 script overview 17
 supported platforms 16
mktar 13, 17
 image generation process 17

N

num_sectors 29
number of partitions 28

O

order (mkimage.py config) 29
order of partitions 28
OS 9
 image 9

P

partition 28, 30–31
 calculate size 31
 customize size 30
 number 28
 order 28
 path to 28
 size 28
 type 28
partition type 28
partition_name 29
path (mkimage.py config) 29
path to partition 28
platforms 16
procnto 11
 starting 11
profiles 26

Q

QNX_CAR_DEPLOYMENT 15
Qt runtime 9

R

reset vector 11

S

script 32
 startup 32
script files 32
scripts 13
 gen-ifs 13
 gen-osversopn 13
 mkifs 13
 mkimage 13

scripts (*continued*)

- `mksysimage.py` 13
 - `mktar` 13
- `sector_size` 29
- `sectors_per_track` 29
- size of image 28
- size of partitions 28
- SLM 32
- startup 11
 - image 11
 - IPL 11
 - reset vector 11
- startup script 32
- supported platforms 16
- System Launch and Monitor 32

T

- target 9, 19, 32
 - build image 19
 - image 9
 - startup script 32
- technical support 8
- type (`mkimage.py` config) 29
- typographical conventions 6

V

- variants 26

W

- Webkit 9

