Getting Started

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/

Electronic edition published: April 30, 2015

Contents

About This Guide

Getting Started describes how to get started with the QNX SDK for Apps and Media. The QNX website has Apps and Media reference (evaluation) images. You can download the platform-specific reference image you need, copy it to your target and begin using the QNX Apps and Media system.

Depending on what you need to do, you should refer to different chapters in this guide:

- *Installing and Booting a Reference Image* (p. 11) explains how to transfer a QNX Apps and Media reference image to your target platform. You should start with this chapter.
- *Building Target Images* (p. 27) explains how to build a target image. Once you are familiar with what is in the QNX Apps and Media reference image, you can follow the instructions in this chapter to recreate a reference image. When you have built a new image, you can transfer it to your target, as explained in "Installing and Booting a Reference Image".
- *Understanding the mksysimage process* (p. 47) explains what **mksysimage.py** does when it generates a target image.
- *Startup* (p. 55) describes the startup process for a QNX Apps and Media system.
- *Modifying Target Images* (p. 61) explains how to modify your QNX Apps and Media target image. These instructions explain how to add or remove components or otherwise customize your Apps and Media image. When you have customized your image, you can build it, as explained in "Building Target Images".

The following table may help you find information quickly:

To find out about:	See:
Getting started with a reference image	*Installing and Booting a Reference Image* (p. 11)
How to transfer a reference image to your target	*Downloading and transferring a reference image* (p. 13)
How to add apps to your QNX Apps and Media project	*Apps and HMIs* (p. 19)
How to build an Apps and Media image	*Building Target Images* (p. 27)
The tasks you must complete to build a target image	*How to create a target image* (p. 30)
The process **mksysimage.py** uses to build an image	*Understanding the mksysimage process* (p. 47)
Scripts and utilities you use to build an Apps and Media image	*Scripts and utilities* (p. 35)
The QNX Apps and Media startup process	*Startup* (p. 55)
How to modify your target image	*Modifying Target Images* (p. 61)

In the interests of brevity, in this document target and reference images generated from the QNX SDK for Apps and Media may be referred to as simply as "QNX Apps and Media target (or reference) image", or even "Apps and Media image".

The variable *base_dir* used in this guide refers to the directory where you have installed QNX SDP on your host system.

We have included in the installation the version of Python that we tested for building Apps and Media.

For information about supported hardware, see the Installation Notes and Release Notes that are posted on the QNX *Download Center*.

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	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	NULL
Data types	**unsigned short**
Environment variables	*PATH*
File and pathnames	**/dev/null**
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
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
Installing and Booting a Reference Image

Platform-specific reference images offer a quick and convenient way to install a QNX Apps and Media system on a target board.

QNX Software Systems makes available platform-specific *reference images* (p. 12) built with the QNX SDK for Apps and Media. If you want to learn about the QNX SDK for Apps and Media, you should start by downloading the reference image for your target platform and transferring it to your target system.

Transferring a QNX Apps and Media reference image is a simple process. However, each supported platform requires some platform-specific actions.

If you are using a reference image, you shouldn't need to do anything with the BSP. The BSP included in the image is configured for QNX Apps and Media on the supported board. If you do need information about the BSP, see the BSP *User Guide* for your platform.

About reference images

A *reference image* is a convenient way to install a complex system on a target board.

A QNX Apps and Media reference image (which is stored as a **.img** file) contains the entire system from the Initial Program Loader (IPL) to the user applications. It also includes the BSP and the startup code for its targeted platform, and all the filesystems, libraries, resource managers, and other components, such as Qt runtime, needed to run a QNX Apps and Media system. As well, it includes a collection of HTML5 and Qt demo apps, which you can launch from the Home screen. For more information about these demo apps, see the QNK SDK for Apps and Media *User's Guide*.

You can transfer a reference image directly to a removable storage medium, such as an SD card or USB key, without having to modify partition information. Once the image is on the removable medium, you can simply insert the SD card or USB key into your target system, and boot.

To get started with a QNX for Apps and Media system, all you need to do is:

1. Download a platform-specific reference image to your host system.
2. Transfer the reference image to a removable storage medium.
3. Insert the removable medium with the target image into your target platform, and boot from this image.

Downloading and transferring a reference image

This section describes how to download an Apps and Media reference image and transfer it to your target platform.

Downloading a reference image

To download a QNX Apps and Media reference image:

1. On the QNX web site (*www.qnx.com*), navigate to the Developers > Downloads section.
2. Download the zip file with the appropriate reference image to your host system.
3. Extract this zip file to a convenient location.

The zip file for VMware contains a **.vmx** file, which can be opened with VMware. See "*Booting a reference image* (p. 15)." Reference images for other platforms contain a **.img** file, which you'll need to transfer to a micro SD card or USB key, depending on the platform.

These reference images use this naming pattern: ***platform-media*.img**. For example, **baytrail-usb.img** or **imx61sabresmart-sd.img**.

The zip files also contain one or more tar files. These are included as a convenience, so you can view the contents of the reference image on a host system that doesn't read QNX6 filesystems.

For more information about downloading a reference image, see the QNX SDK for Apps and Media 1.1 *Installation Notes*.

Transferring a reference image to removable media

The QNX Apps and Media reference images support only micro SD cards or USB keys, depending on the target platform. The reference images are approximately 500 MB when zipped, and expand to 4 GB when extracted. The minimum requirements for the removable storage are:

- micro SD card: 4 GB Class 10
- USB key: 4 GB USB 3.0

We recommend UHS-I cards for better read/write performance. These cards can be identified by a "U" with a number "1" inside it, as shown below:

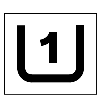

Figure 1: The UHS-I identifier

Different hardware platforms (boards) support different removable media devices. Use the type of device your platform requires:

- BeagleBone Black — micro SD card: SanDisk Ultra® microSDHC™ UHS-I SD card for mobile devices, 4 GB or larger
- i.MX6x SABRE Smart — micro SD card: SanDisk Ultra® microSDHC™ UHS-I SD card for mobile devices, 4 GB or larger (Use a micro to SD converter to insert the micro SD card into the SD card slot.)

- OMAP5 — micro SD card: SanDisk Ultra® microSDHC™ UHS-I SD card for mobile devices, 4 GB or larger
- VMware — not required
- x86 Bay Trail — USB key: Kingston Technology 8 GB DataTraveler G4 USB 3.0 flash drive

To transfer a QNX Apps and Media reference image to a micro SD card or USB key, follow the instructions for your host OS.

When you have finished transferring the reference image to the removable media, follow the platform-specific startup instructions.

Linux

On a Linux host system, use these command-line instructions to copy a reference image to removable storage:

```
sudo dd bs=1048576 if=your_image of=/dev/sdx
```

This command causes the `dd` utility to write 1 MB chunks of data to the disk at a time. This command assumes that **sdx** is the SD card (or other removable storage).

The device name shouldn't include a partition suffix. For example, do *not* use **/dev/sdx1**. However, on some Linux variants, the device name can be **/dev/mmcblk0**.

Windows

On a Windows system, to copy a reference image to removable storage:

1. If you don't already have Win32 Disk Imager on your system, download it from this site, then install it:

   http://sourceforge.net/projects/win32diskimager/

2. Run the Win32 Disk Imager.
3. Browse to the location where you extracted the the image from the zip archive, and click **Open**.
4. Click **Write** to write the **.img** file to your microSD card.
5. Click **Yes** to begin the process of writing the image. When it's complete, you'll see the message "Write successful."
6. Click **OK**, then exit Win32 Disk Imager.

Booting a reference image

After you have transferred your Apps and Media image to your target platform, you can boot your system and start running apps.

To boot from a QNX Apps and Media reference image, follow the instructions for your supported target platform. For detailed information about supported platforms, platform variants, and revisions, see the QNX SDK for Apps and Media *Release Notes*.

For information about connecting board power supplies, screens, and other peripheral devices, see the BSP *User Guide* for your board. For an explanation about board and peripheral support, see the *Release Notes*. For information about what's included in the reference image, see “A Guided Tour of the Reference Image” in the *User's Guide*.

If you use a physical keyboard connected to the target platform to get started, you must disconnect it from the target platform to be able to use the touch keyboard.

BeagleBone Black

To boot from the QNX Apps and Media reference image on your BeagleBone Black target:

1. Remove the micro SD card from your host system, and insert it into the target platform's SD card slot.
2. Hold down the target's **S2** switch to cause the target to boot from the SD card.
3. Connect the target platform's power supply.

Your Apps and Media reference image should boot, and you should see the QNX Apps and Media system running in the QNX environment.

i.MX6x SABRE Smart

To boot from the QNX Apps and Media reference image on your i.MX6x SABRE Smart target:

1. Configure the board's SW6 DIP switches as shown below:

Figure 2: SW6 DIP switch configuration to boot the smart device from the SD card slot (SD3)

2. Remove the micro SD card from your host system, and use a micro SD to SD card converter to insert it into the target platform's SD card slot.
3. Connect the target platform's power supply and power up the board.

Your Apps and Media reference image should boot, and you should see the QNX Apps and Media system running in the QNX environment.

OMAP5

To boot from the QNX Apps and Media reference image on your OMAP5 target:

1. Remove the micro SD card from your host system, and insert it into the target platform's SD card slot.
2. Connect the target platform's power supply and power up the board.

Your Apps and Media reference image should boot, and you should see the QNX Apps and Media system running in the QNX environment.

VMware

The QNX Apps and Media reference image for VMware is designed to run in VMware on your PC. Since it is an x86 image, you can open it in VMware directly.

The QNX Apps and Media supports:

- VMware Workstation 9.0 or higher
- VMware Player 5.0 or higher
- VMware Fusion 5 or higher

Virtual Box isn't supported.

To use the QNX Apps and Media reference image for VMware:

1. Start a supported version of VMware Workstation, Player, or Fusion.
2. Open a virtual machine, browse to the location where you saved the QNX Apps and Media reference image for VMware, then choose **qnxanm.vmx**.

   If VMware displays a dialog indicating that the virtual machine was moved, select **I copied it** (as recommended), then click **OK**.

3. Power on the virtual machine for this image.

Your Apps and Media reference image should boot, and you should see the QNX Apps and Media system running in the QNX environment.

x86 Bay Trail

For x86 Bay Trail targets, you should update your BIOS to version 0039 or higher. See: *https://downloadcenter.intel.com/SearchResult.aspx?lang=eng&FamilyId=36&LineId=3736&ProductID=3782&ProdId=3782*.

To boot from the Apps and Media reference image on your x86 target:

1. Remove the USB device from your host system, and insert it into a USB port on your x86 target.
2. Connect the target platform's power supply and power up the x86 target.
3. Enter the BIOS to configure the machine to boot from the USB drive that you've just inserted.
4. After configuring the BIOS to use the USB drive as the primary boot device, reboot the target.

Your Apps and Media reference image should boot, and you should see the QNX Apps and Media system running in the QNX environment.

Screen calibration

The first time you start a QNX Apps and Media reference image on a target, the system automatically prompts you to calibrate the screen. Subsequent startups will go directly to the QNX Apps and Media **Home** screen. For more information about how to calibrate the screen, use the software keyboard, or start an app, see “A Guided Tour of the Reference Image” in the *User's Guide*.

Chapter 2
Apps and HMIs

You can add apps to Apps and Media, either in the target image or afterwards when the system is up and running. You can also configure your Apps and Media system to use a simple, monolithic HMI instead of the default HMI with packaged apps.

Adding apps without modifying the Apps and Media image

If you are an HTML5 or Qt app developer who just wants to get apps into a QNX Apps and Media system, you don't need to modify and rebuild a target image. You can add apps to a system after it is up and running. For instructions, see “Packaging, Installing, and Launching Apps” in the *Application and Window Management* guide.

Full screen HMI

The reference image includes a second HMI that demonstrates how to run a single, monolithic HMI without packaging apps into Blackberry ARchive (BAR) files.

The default HMI delivered with QNX Apps and Media reference images uses *Application and Window Management* components, such as the *Authorization Manager* (**authman**) and *Application Launcher* (**launcher**), and apps packed into BAR files. If you don't want to use this HMI model, you can configure your system to use a second HMI, also written in Qt.

To configure your system to display this simple HMI instead of the default Home screen:

1. On your target, select **Settings** to get the IP address for your target platform.
2. On your host system, use SSH to connect to the target (username: `root`, password: `root`).
3. Use `elvis` or `vi` to change the relevant line in **/var/etc/services-enabled** to disable the QT home screen. Change:

   ```
   QTHOMESCREEN:true
   ```

   to

   ```
   QTHOMESCREEN:false
   ```

4. Save the **services-enabled** file.
5. Restart your board.

After the board restarts, the simple HMI showing the QNX logo and the target's IP address appears:

Figure 3: A simple HMI that consists of a white background, the QNX logo, and the target's IP address.

To make this change persistent during image rebuilds, modify the **services-enabled** file in your deployment workspace: ***$QNX_DEPLOYMENT_WORKSPACE*/target/product/AnM/boards/*board-specific*/var/etc**

You can use the same method to enable and disable other services:

```
# service:true | false
```

For example:

```
WIFI:false
QTHOMESCREEN:false
USBCAM:true
```

It's expected that developers write their own full screen HMI, as the sample provided is intended just to demonstrate how to start Qt runtime and its dependencies (e.g., graphics/**Screen**, multimedia/**mm-renderer**). For more information, see “Writing an HMI” and “Source code for sample Qt apps” in the *QT Developer's Guide.*

Replacing the full screen HMI

You can replace the QNX full screen HMI with one you have developed using the QNX Qt Development Framework.

SLM launches the full screen HMI via a script, **qthmi-start.sh**. This action is configured in ***$QNX_DEPLOYMENT_WORKSPACE*/target/product/AnM/etc/slm-config-all.xml**:

```
<SLM:component name="hmi">
   <SLM:command>/base/scripts/qthmi-start.sh</SLM:command>
   <SLM:stop stop="signal">SIGTERM</SLM:stop>
   <SLM:envvar>APP_MEDIA_KB_PATH=/base/usr/anm/bin/keyboard</SLM:envvar>
   <SLM:envvar>APP_MEDIA_ASSET_PATH=/base/usr/anm/share/appmedia11</SLM:envvar>
   <SLM:envvar>QQNX_PHYSICAL_SCREEN_SIZE=150,90</SLM:envvar>
   <SLM:depend>symlink_hmi</SLM:depend>
   <SLM:depend>symlink_base</SLM:depend>
   <SLM:depend>iopkt</SLM:depend>
   <SLM:depend>pps</SLM:depend>
   <SLM:depend>calib-done</SLM:depend>
   <SLM:depend>keyboard</SLM:depend>
   <SLM:depend>screen-ready</SLM:depend>
</SLM:component>
```

The **qthmi-start.sh** script is located in ***$QNX_DEPLOYMENT_WORKSPACE*/target/product/AnM/scripts/**. It invokes the executable called **QtSimpleHMI**, which is located in the ***base_dir*/target/qnx6/*architecture*/usr/anm/bin** directory.

The **QtSimpleHMI** is included in the **basefs.anm.qt.xml** fileset in the ***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/filesets/** directory.

To replace the default full screen HMI with your own HMI:

1. Place your HMI executable file in the ***base_dir*/target/qnx6/*architecture*/usr/anm/bin** directory.
2. In the **services-enabled** file in the ***$QNX_DEPLOYMENT_WORKSPACE*/target/product/AnM/boards/*board-specific*/var/etc** directory, enable the full screen HMI by changing:

   ```
   QTHOMESCREEN:true
   ```

   to

   ```
   QTHOMESCREEN:false
   ```

3. In the **basefs.anm.qt.xml** fileset in the ***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/filesets/** directory, replace **QtSimpleHMI** with the name of your HMI executable.
4. In the **qthmi-start.sh** script located in the ***$QNX_DEPLOYMENT_WORKSPACE*/target/product/AnM/scripts/** directory, replace **QtSimpleHMI** with the name of your HMI executable.
5. Rebuild your target image by following the instructions for your board in "*Building a target image* (p. 38)."

6. Follow the instructions for *transferring an image* (p. 13) and *booting your board* (p. 15).

Pre-installing apps

You can pre-install apps by adding them to your Apps and Media target image and re-building the image.

If you are a system developer who needs to deliver pre-installed apps, you can package your apps in BAR files, add them to your target image configuration profile, then rebuild the image.

Before you begin, if you aren't familiar with how the QNX SDK for Apps and Media uses filesets and profiles, see "*Filesets and profiles* (p. 62)".

Unless otherwise specified, these instructions assume a Linux host. If you are working on a Windows host, run the commands in the `bash` shell.

To include pre-installed apps in a target image, you need to modify the `mksysimage` configuration files used to generate the image, by doing the following:

1. Create a **.bar** file (archive) for each of your new apps.

   For information about creating **.bar** files, see "Packaging an HTML5 app" in the *HTML5 Developer's Guide* and "Packaging the app into a BAR file from Qt Creator" in the *Qt Developer's Guide*.

2. Copy your **.bar** file(s) to the following directory (or one of its subdirectories):

   ***base_dir*/target/qnx6/appinstall/bars/unsigned**

3. In the profile file for your board (such as **os.xml**), specify the location where you copied the **.bar** file(s). Profile files are located under:

   ***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/*platform*.*variant*/**

   In the `<application>` section of the profile file, add an `<include-application>` entry for each **.bar** file. In this entry, you must specify the archive name and location (relative to the **/unsigned** directory). For example, if you copied the file *myapp***.bar** to the **/qt** subdirectory, your entry in the profile file would be:

   ```
   <include-application name="qt/myapp.bar" secure="unsigned">
   ```

4. To generate a new image that includes your app(s), go to the ***$QNX_DEPLOYMENT_WORKSPACE*/infra/utils/scripts/** directory and run the appropriate `mksysimage` command:

   For Linux:

   ```
   mksysimage.sh -o output_path platform.ext -P AnM -f
   ```

   For Windows:

   ```
   mksysimage.bat -o output_path platform.ext -P AnM -f
   ```

   After you run this command, your app(s) will be included in the generated image.

The *QNX_DEPLOYMENT_WORKSPACE* environment variable must be set to the location where you copied ***base_dir*/product_deployment/QSAM/1.1/deployment/**. For information about this variable, see "*Environment variables* (p. 33)".

For more detailed information about modifying target images, see "*Modifying Target Images* (p. 61)".

Chapter 3
Building Target Images

To generate a QNX SDK for Apps and Media target image you only need to run a single script that launches other scripts that build the image.

Overview of the process

The diagram below shows the process used to generate a QNX Apps and Media target image. You only need to run the **mksysimage.sh** (Linux) or **mksysimage.bat** (Windows) file to start the process:

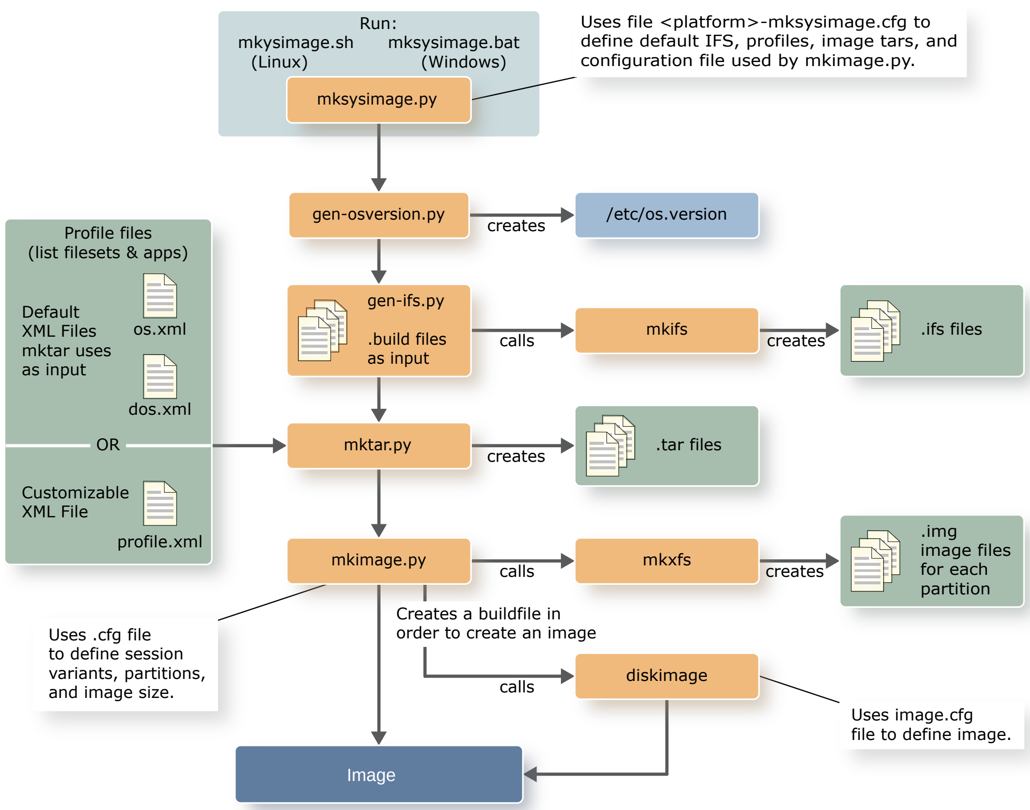

Figure 4: Image generation process for the QNX SDK for Apps and Media platform

The only script you should run to generate an image is **mksysimage.sh** (Linux) or **mksysimage.bat** (Windows). These scripts look after the process of running (in the proper sequence) the other scripts that build your image.

Final image

The final step in the image-generation process 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.

Image artifacts

When you complete a project with the QNX SDK for Apps and Media, you must build a target image to install your project on the target hardware.

To build a target image, you need to package all the following components:

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

Typically, you don't build all these components yourself. Instead, you:

- use the prebuilt binaries which you put in your development environment when you installed the QNX SDK for Apps and Media platform
- install additional packages
- download any required BSPs

Artifact organization

When you design a system, you must choose how to organize components into one or more IFS files and partition images. You then combine partition images to produce target images:

Image filesystems (IFS)

Image Filesystems (or **.ifs** files) are created by `mkifs`. An IFS is a file that contains a bootable OS image. An IFS is a single binary that is loaded into RAM, generally by a bootloader or IPL on bootup. IFS files are usually quite minimal; they include 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 in 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.

In the case of the QNX SDK for Apps and Media platform, everything required to start a QNX kernel and mount secondary storage is stored in an IFS, but the HMI and apps are loaded from a storage device. The primary binaries and libraries in the IFS are automatically mounted to **/proc/boot**.

Partition images

Partition images (or **.image** files) are created by `mkxfs` (a wrapper utility for various filesystem generation utilities; see “M” in the *Utilities Reference*). These files contain the contents of a partition that is written to a storage device.

Partition images can contain a variety of file types including IFS files. For the QNX SDK for Apps and Media platform, for non-x86 platforms, the primary IFS is stored in a FAT32 partition because most targets can read FAT32 with their default bootloader.

For x86 platforms, the QNX IPL is used and it loads an IFS from a QNX filesystem.

An image may consist of a **maximum of four primary partitions**. For example, an SD image for the QNX SDK for Apps and Media platform contains three (3) partitions: a FAT32 partition for booting and two Power-Safe (**fs-qnx6.so**) partitions (one for system data and another for user data).

Target images

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 a micro SD card or a USB key, without having to modify partition information.

Typically, the resulting image is stored in non-removal storage; however, SD cards or USB keys are easier to begin with.

How to create a target image

The QNX website includes reference (evaluation) images for you to explore and use; however, as a system integrator or the developer of a custom solution, you might want to generate your own images.

If you want to use a custom image, you'll have to create an image for your target board. When you have your board booting with this custom image, you can modify the filesets to include additional packages and applications in the generated image.

The image generation process

Depending on your board, you will need to follow these high-level steps to create your target image:

Step 1

Download a board-specific BSP (which contains drivers and prebuilt files) from the QNX website. For information about QNX BSPs, see "*Board Support Packages (BSPs)* (p. 49)" and the BSP *User Guide*.

Step 2

Extract your BSP, then copy the files from the **/prebuilt** directory. For instructions about extracting BSPs and where to copy the files, see the platform-specific instructions under "*Building a target image* (p. 38)."

Step 3

(Optional) Depending on the board and boot loader used, you may require an Initial Program Loader (IPL, which loads the IFS) or an MLO (multimedia card loader for OMAP5 EVM).

To generate the IPL, run the `mkflashimage` script. For more information, see the BSP *User Guide* for your board.

Step 4

(Optional) You can customize the image to:

- change its contents (e.g., add or remove apps)
- include binaries and files in filesets
- modify the configuration profile files (e.g., **os.xml**, **dos-sd.xml**) for the board-specific package
- modify the startup process using **.build** files and System Launch and Monitor (SLM)

Step 5

Run the **mksysimage.sh** (Linux) or **mksysimage.bat** (Windows) script to generate your target image.

The following diagram shows an overview of the process used to create a target image for the QNX SDK for Apps and Media. This process is discussed in detail in the chapters that follow.

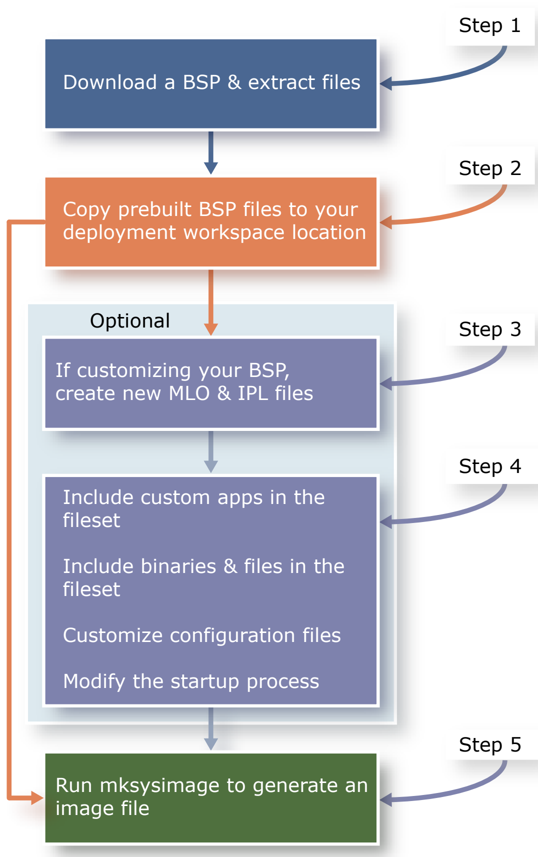

Figure 5: Process to create a QNX SDK for Apps and Media target image

The `mksysimage.py` image generation script

Because generating the various files needed for a complete target image is a time-consuming and error-prone process, the QNX SDK for Apps and Media platform includes the `mksysimage.py` Python script that handles the entire process of generating a system image. To ensure that you use the correct version of Python (included with the Apps and Media installation), always use `mksysimage.sh` (Linux) or `mksysimage.bat` (Windows) to run this Python Script.

For more information about `mksysimage.py` and other relevant scripts and utilities, see *Building Target Images* (p. 27) in this guide, Image Generation Utilities in the *System Services Reference*, and the QNX SDP *Utilities Reference*.

You also have the option of generating a system image manually; you can run the individual utilities manually to generate any of the following:

- IFS
- TAR file
- partition images
- disk image

Before you begin

To create a QNX Apps and Media image, you need to have the QNX SDP 6.6 installed on your system.

After you unzip a BSP, a prebuilt IFS image is available in the BSP's **/images** directory. This prebuilt image is configured for the various BSP device drivers already running. (The prebuilt IFS only demonstrates what's in the core OS and not Apps and Media.) When you build the BSP, the prebuilt image will be overwritten with a new image that is generated by the BSP build process, so you may want to make a copy of the prebuilt image for future reference. However, if you forget to make a copy of the prebuilt image, you can still recover the original one—simply extract the BSP from the zip archive into a new directory.

Before you begin the process of creating an image for QNX SDK for Apps and Media, make sure that you:

- have installed the QNX Software Development Platform 6.6 for either a Windows or Linux host
- have downloaded the QNX Qt runtime packages and followed the installation instructions for these packages
- have a target with a touchscreen for use with the browser app
- can build and run the QNX SDP 6.6 BSP for your target platform without any issues

---

ELF executables and shared objects are automatically marked as executable (unless you specify `[+raw]`).

Codecs for video playback

For some platforms, you need to get and install codecs or decoders for video playback:

BeagleBone Black

Video playback requires installation of the Ittiam software video decoder. You can obtain the necessary files in the **ittiam-*datestamp*.zip** package.

i.MX6Q SABRE Smart

Video playback requires installation of the Freescale video codecs. You can obtain the necessary files in the **freescale-*datestamp*.zip** package.

OMAP5432 EVM

Video playback requires installation of the Texas Instruments video codecs. You can obtain the necessary files in the **ti-*datestamp*.zip** package.

You can obtain the packages from the *QNX Download Center*. Install them according to the installation instructions provided with the packages.

Environment variables

Before you create an image, you must make sure the environment variables relevant to the image creation process are properly set.

Environment variables relevant to image creation

Listed below are the environment variables associated with the image creation process. Note that *PATH*, *QNX_HOST*, and *QNX_TARGET* are set when you run the script that defines the environment variables:

BSP_ROOT_DIR

The name of the directory where you extracted the BSP archive.

CPU_VARIANT

The CPU architecture for which the BSP is designed. For QNX Neutrino RTOS SDP 6.6, for example, the supported CPU variants are **armle-v7** or **x86**.

PATH

A colon-separated list of directories that are searched when the shell looks for commands. For more information, see `ksh` in the *Utilities Reference*.

QNX_DEPLOYMENT_WORKSPACE

The path to the QNX deployment work space where you copied the files required to build the image. It determines the location for the files that go onto the target. Before you begin working:

1. Copy to another location the directory and all the contents of:

   ***base_dir*/product_deployment/QSAM/1.1/deployment/**

   where *base_dir* is the directory where you have installed the QNX Neutrino SDP.

2. Set the *QNX_DEPLOYMENT_WORKSPACE* environment variable to the location where you copied **deployment** and its contents.

You can also use this environment variable when you have custom hardware (boards) that don't currently exist in the **boards** directory. If you want to create your own copy of the QNX **deployment** directory structure for your requirements, you can use this environment variable to reference your specific source control.

QNX_HOST

The location of host-specific files for all development hosts.

QNX_PRODUCT

An optional environment variable that identifies the default product to use, such as **AnM** for QNX SDK for Apps and Media. If you don't set this environment variable, when you generate a target image, set the `-P` option as `-P AnM` (for Apps and Media).

QNX_PYTHON_PATH

An optional environment variable that specifies the location of the Python interpreter used to generate images. This variable is set by running **mksysimage.sh** (Linux) or **mksysimage.bat** (Windows).

QNX_QT

An optional environment variable that defines a default location for the installed version of Qt that you want to use. By default it's not set; however, it should reference the architecture-dependent location where Qt is installed on the host computer. If you don't set this environment variable, when you generate a target image, use the `-Q` option.

QNX_TARGET

The location of SDP (OS) target content on the host device.

To see a detailed list of environment variables used in QNX SDP and QNX for Apps and Media, see the appendix Commonly Used Environment Variables in the QNX SDP *Utilities Reference*.

Set environment variables for image creation

To set environment variables for the QNX SDK for Apps and Media, type the following at the command prompt:

Linux:

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

Windows:

```
base_dir\qnx660-env.bat
```

where *base_dir* is the directory where you installed the QNX 6.6.0 SDP.

CAUTION: When you run **qnx660-env.sh** (Linux) or **qnx660-env.bat** (Windows), the variables are only set locally. Therefore, every time you open a shell or a command-line prompt, you must run the command to set the environment variables.

Scripts and utilities

Several scripts and utilities are used by `mksysimage.sh` (Linux) or `mksysimage.bat` (Windows) to build an Apps and Media image.

`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:

***$QNX_DEPLOYMENT_WORKSPACE*/infra/utils/scripts/mksysimage.py**

where *QNX_DEPLOYMENT_WORKSPACE* is the location where you copied:

***base_dir*/product_deployment/QSAM/1.1/deployment**

By default, `mksysimage.py` reads a platform-specific configuration file (***platform*. *variant*/mksysimage/ *platform*-mksysimage.cfg**) from the following directory:

***QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/**

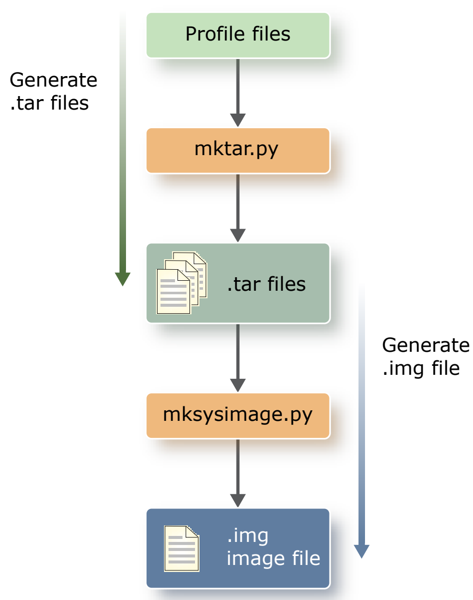

Figure 6: `mktar.py` uses profile files (that list filesets and apps) to create tar files, then `mksysimage.py` generates an image file (`.img`) from these tar files.

Configuration files specify image variants for each platform; they specify which tar (**.tar**) files and images are generated. These tar files are intermediate containers used in the image generation process. For information, see "*Image configuration settings* (p. 68)" and "*Filesets and profiles* (p. 62)."

For all platforms, `mksysimage.py` generates a tar file and an image:

***platform*-os.tar**

The content that goes into a QNX Power-Safe filesystem. Typically, this content includes all files except the ones needed for internal booting, such as MLO and IFS files.

***platform*-*image_variant*.img**

The image file for the bootable media used by the platform (USB stick or micro SD card).

For ARM platforms only, `mksysimage.py` also generates a second tar file:

***platform*-dos-*image_variant*.tar**

The content that goes into a FAT32 filesystem that includes all boot files, such as MLO and IFS files. FAT32 is compatible with most boot loaders.

The generated image includes the tar files mentioned above. Linux and Windows hosts can't read the final image, so the tar files allow you to see what will be included in an image.

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*.

gen-osversion.py

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

gen-ifs.py

The `gen-ifs.py` utility script consolidates various **.build** file segments into a single buildfile before calling the `mkifs` utility to create the **.ifs** file(s) that will be 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.py` in the *System Services Reference*.

mktar.py

The `mktar.py` utility creates a tar file containing the files, directories, symbolic links, and their permissions as specified in the filesets. These tar files contain the QNX Apps and Media files for the specified platform variant and are used to generate the QNX Power-Safe and FAT32 filesystems included in the QNX Apps and Media target image.

As input, the `mktar.py` utility uses the **dos-*variant*.xml** and **os.xml** files; otherwise, it uses the default **profile.xml** file. These files specify which filesets to include, and for **os.xml**, the **.bar** files to pre-install.

The contents of a partition come from these generated tar files.

`mkimage.py`

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

- The `mkimage.py` utility script processes 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` 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 the partition image files (***partition_name*.image**) into a single image.

`mkflashimage`

The `mkflashimage` script is included in BSPs for the i.MX6x SABRE Smart Devices and OMAP5 platforms. It is used to generate IPLs for these targets.

Building a target image

If you modify the deployment configuration or target contents of your QNX Apps and Media system, you can use command-line instructions to build an image to include your modifications.

Before you begin building your QNX Apps and Media target image, familiarize yourself with the available scripts and configuration files, then prepare your working environment, as described below in "*Setting up* (p. 38)."

- Unless otherwise specified, these instructions assume that you are working in the command line on a Linux host.
- For supported board variants and peripheral devices, see the *Release Notes*.
- For more information about the target platform and the QNX BSP for this platform, see the BSP *User Guide*.
- For additional information about what happens when you build a QNX Apps and Media target image, see *Building Target Images* (p. 27).

Scripts and configuration files

The QNX SDK for Apps and Media installation process creates a workspace that contains the scripts and configuration files you'll use when you build your target image.

Scripts are under the following directories (assuming that you have already set *$QNX_DEPLOYMENT_WORKSPACE* as instructed in "*Setting up* (p. 38)"):

***$QNX_DEPLOYMENT_WORKSPACE*/infra/utils/scripts**

The (optional) configuration files are in:

***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards**

At this location, under ***platform*.*variant***, there are two important directories:

***platform*.*variant*/mksysimage**

Contains files for `mksysimage` configuration.

***platform*.*variant*/ifs**

Files in this directory specify how to generate the IFS.

Setting up

When you build a custom image, start with the following preliminary steps:

1. Copy to another location the directory and all the contents of:

   ***base_dir*/product_deployment/QSAM/1.1/deployment/**

   (where *base_dir* is the directory where you have installed the QNX Neutrino SDP).

2. Set the *QNX_DEPLOYMENT_WORKSPACE* environment variable to the location of the new copy of the **deployment** directory (include the **/deployment** directory in the path).

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 `mksysimage`, or the script won't generate the image.
4. Set the *QNX_QT* environment variable to the architecture-specific path of your QNX Qt Development Framework installation. This environment variable tells the `mksysimage` script (which generates the target image) where the QNX Qt runtime is installed on your host.

   For example, on a Linux host for an ARMLE-v7 target:

   ```
   export QNX_QT=qt_base_dir/QNX-qt/Qt-5.3.1-armle-v7
   ```

   On a Windows host for an x86 target:

   ```
   set QNX_QT=qt_base_dir\QNX-qt\Qt-5.3.1-x86
   ```

   where *qt_base_dir* is the directory where you installed the QNX Qt Development Framework.

Later, when you run `mksysimage`, redirect its output to a file and look for any warning and error messages about missing files. For example:

```
Warning: host file filename missing.
```

When you run `mksysimage.sh` (Linux) or `mksysimage.bat` (Windows) to generate a system image file (**.img**), you must set these options:

- You must specify the argument for the `-P` option as `-P AnM` (for Apps and Media).
- The `mksysimage.py` script needs to know the full path to the QNX Qt runtime on your target, so it can find the Qt libraries and binaries. If you haven't set the *QNX_QT* environment variable, when you run `mksysimage`, use the `-Q` option to specify where the QNX Qt runtime is installed on your host.
- Run the **mksysimage.py** script with the `-f` option to force it to overwrite existing tar files.

BeagleBone Black

These instructions describe how to build a QNX Apps and Media target image for BeagleBone Black platforms.

Before building your target image, you should understand the available scripts and configuration files, and prepare your working environment, as described in "*Setting up* (p. 38)." You should also understand the general procedure for extracting and building a BSP, as discussed in "*Board Support Packages (BSPs)* (p. 49)." Be aware, however, that the steps that follow are not the same as the general BSP build steps.

To build your own QNX Apps and Media image for Texas Instruments AM335x BeagleBone Black platforms, on your host system:

1. Set up the environment variables for the QNX 6.6.0 SDP, and the development environment for QNX Apps and Media:

Linux:

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

Windows:

```
base_dir\qnx660-env.bat
```

where *base_dir* is the directory where you installed the QNX 6.6.0 SDP.

2. Set the *QNX_QT* environment variable to the location of the Qt runtime on your host system:

   Linux:

   ```
   export QNX_QT=qt_base_dir/QNX-qt/Qt-5.3.1-armle-v7
   ```

   Windows:

   ```
   set QNX_QT=qt_base_dir\QNX-qt\Qt-5.3.1-armle-v7
   ```

3. Extract a BSP, then copy everything from the **/prebuilt** directory to the board-specific directory in the ***QNX_DEPLOYMENT_WORKSPACE*** path, as follows. We'll refer to the directory where you extracted the BSP as *bsp_dir*:

   ```
   cd bsp_dir
   cp -r prebuilt/* $QNX_DEPLOYMENT_WORKSPACE/target/boards/beaglebone/
   ```

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

   ```
   mkdir output_dir
   ```

5. From the ***$QNX_DEPLOYMENT_WORKSPACE*/infra/utils/scripts/** directory, run `mksysimage.sh` (Linux) or `mksysimage.bat` (Windows) to generate a system image file (**.img**):

   ```
   mksysimage.sh -P AnM -o output_dir beaglebone.ext -f
   ```

   where *output_dir* is the location of the new image.

You should now have an image file (**.img**) ready to write to a micro SD card so you can transfer it to your target. For instructions, see "*Downloading and transferring a reference image* (p. 13)."

i.MX6x SABRE Smart Devices

These instructions describe how to build a QNX Apps and Media target image for i.MX6x SABRE Smart Devices platforms.

Before building your target image, you should understand the available scripts and configuration files, and prepare your working environment, as described in "*Setting up* (p. 38)." You should also understand the general procedure for extracting and building a BSP, as discussed in "*Board Support Packages (BSPs)* (p. 49)." Be aware, however, that the steps that follow are not the same as the general BSP build steps.

To build your own QNX Apps and Media image for Freescale i.MX6x SABRE Smart Devices platforms, on your host system:

1. Set up the environment variables for the QNX SDP, and the development environment for QNX Apps and Media:

   Linux:

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

   Windows:

   ```
   base_dir\qnx660-env.bat
   ```

   where *base_dir* is the directory where you installed the QNX 6.6.0 SDP.

2. Set the *QNX_QT* environment variable to the location of the Qt runtime on your host system:

   Linux:

   ```
   export QNX_QT=qt_base_dir/QNX-qt/Qt-5.3.1-armle-v7
   ```

   Windows:

   ```
   set QNX_QT=qt_base_dir\QNX-qt\Qt-5.3.1-armle-v7
   ```

3. Extract a BSP, then copy everything from the **/prebuilt** directory to the board-specific directory in the ***QNX_DEPLOYMENT_WORKSPACE*** path, as follows. We'll refer to the directory where you extracted the BSP as *bsp_dir*:

   ```
   cd bsp_dir
   cp -r prebuilt/* $QNX_DEPLOYMENT_WORKSPACE/target/boards/imx61sabresmart/
   ```

4. From the BSP directory, run `make`, then from the **/images** subdirectory, run `mkflashimage` to generate an IPL:

   Linux:

   ```
   cd bsp_dir
   make
   cd bsp_dir/images
   mkflashimge
   ```

   Windows:

   ```
   cd bsp_dir
   make
   cd bsp_dir/images
   sh mkflashimge
   ```

   This utility script is shipped in the BSP. It creates the IPL as the following binary file:

***bsp_dir*/images/ipl-mx6q-sabresmart.bin**

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

   ```
   mkdir output_dir
   ```

6. From the ***$QNX_DEPLOYMENT_WORKSPACE*/infra/utils/scripts/** directory, run the following command to generate a system image file (**.img**):

   Linux:

   ```
   mksysimage.sh -P AnM -o output_dir imx61sabresmart.ext -f
   ```

   Windows:

   ```
   mksysimage.bat -P AnM -o output_dir imx61sabresmart.ext -f
   ```

   where *output_dir* is the location of the new image.

7. Copy the IPL to offset 1024 of the image file (**.img**) you just created:

   ```
   dd if=bsp_dir/images/ipl-mx6q-sabresmart.bin
       of=output_dir/imx61sabresmart-sd.img
       bs=512 seek=2 skip=2 conv=notrunc
   ```

   The `dd` utility isn't provided with Windows. To perform this step on Windows, download a native Windows implementation of `dd`.

You should now have an image file (**.img**) ready to write to a micro SD card so you can transfer it to your target. For instructions, see "*Downloading and transferring a reference image* (p. 13)."

OMAP5 EVM

These instructions describe how to build a QNX Apps and Media target image for OMAP5 EVM target platforms.

Before building your target image, you should understand the available scripts and configuration files, and prepare your working environment, as described in "*Setting up* (p. 38)." You should also understand the general procedure for extracting and building a BSP, as discussed in "*Board Support Packages (BSPs)* (p. 49)." Be aware, however, that the steps that follow are not the same as the general BSP build steps.

To build your own QNX Apps and Media target image for Texas Instruments OMAP5432 EVM platforms, on your host system:

1. Set up the environment variables for the QNX SDP, and the development environment for QNX Apps and Media:

   Linux:

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

Windows:

```
base_dir\qnx660-env.bat
```

where *base_dir* is the directory where you installed the QNX 6.6.0 SDP.

2. Set the *QNX_QT* environment variable to the location of the Qt runtime on your host system:

   Linux:

   ```
   export QNX_QT=qt_base_dir/QNX-qt/Qt-5.3.1-armle-v7
   ```

   Windows:

   ```
   set QNX_QT=qt_base_dir\QNX-qt\Qt-5.3.1-armle-v7
   ```

3. Extract a BSP, then copy everything from the **/prebuilt** directory to the board-specific directory in the ***QNX_DEPLOYMENT_WORKSPACE*** path, as follows. We'll refer to the directory where you extracted the BSP as *bsp_dir*:

   ```
   cd bsp_dir
   cp -r prebuilt/* $QNX_DEPLOYMENT_WORKSPACE/target/boards/omap5uevm/
   ```

4. From the BSP directory, run `make`, then from the **/images** subdirectory, run `mkflashimage` to generate an IPL:

   ```
   cd bsp_dir
   make
   cd bsp_dir/images
   mkflashimge
   ```

   This utility script is shipped in the BSP. It creates the IPL as the following binary file:

   ***bsp_dir*/images/sd-ipl-omap5-uevm5432.bin**

5. Copy the IPL to the **sd-boot** directory:

   ```
   cp bsp_dir/images/sd-ipl-omap5-uevm5432.bin
       $QNX_DEPLOYMENT_WORKSPACE/target/boards/omap5uevm/sd-boot/MLO
   ```

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

   ```
   mkdir output_dir
   ```

7. From the ***$QNX_DEPLOYMENT_WORKSPACE*/infra/utils/scripts/** directory, run `mksysimage.sh` (Linux) or `mksysimage.bat` (Windows) to generate a system image file (**.img**):

   ```
   mksysimage.sh -P AnM -o output_dir omap5uevm.ext -f
   ```

   where *output_dir* is the location of the new image.

You should now have an image file (**.img**) ready to write to a micro SD card so you can transfer it to your target. For instructions, see "*Downloading and transferring a reference image* (p. 13)."

VMware

These instructions describe how to build a QNX Apps and Media target image which you can use in VMware on your computer.

Before building your target image, you should understand the available scripts and configuration files, and prepare your working environment, as described in "*Setting up* (p. 38)."

To build your own QNX Apps and Media target image for VMware, on your host system:

1. Set up the environment variables for the QNX SDP, and the development environment for QNX Apps and Media:

   Linux:

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

   Windows:

   ```
   base_dir\qnx660-env.bat
   ```

   where *base_dir* is the directory where you installed the QNX 6.6.0 SDP.

2. Set the *QNX_QT* environment variable to the location of the Qt runtime on your host system:

   Linux:

   ```
   export QNX_QT=qt_base_dir/QNX-qt/Qt-5.3.1-x86
   ```

   Windows:

   ```
   set QNX_QT=qt_base_dir\QNX-qt\Qt-5.3.1-x86
   ```

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

   ```
   mkdir output_dir
   ```

4. From the ***$QNX_DEPLOYMENT_WORKSPACE*/infra/utils/scripts/** directory, run `mksysimage.sh` (Linux) or `mksysimage.bat` (Windows) to generate a system image file (**.img**), using the path to the x86 IPL:

   ```
   mksysimage.sh -P AnM -o output_dir -k
       "-b base_dir/target/qnx6/x86/boot/sys/ipl-diskpc1" vmware.ext -f
   ```

   where *output_dir* is the location of the new image.

5. Copy the following two files:

   - **qnxAnM.vmx**
   - **vmware-qnxAnM.vmdk**

from:

***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/vmware.ext/vm-config/**

into the same directory as the **.img** file.

You should now be able to launch a VMware session and open the **qnxAnM.vmx** file to launch QNX Apps and Media.

x86 Bay Trail

These instructions describe how to build a QNX Apps and Media target image for x86 Bay Trail platforms.

Before building your target image, you should understand the available scripts and configuration files, and prepare your working environment, as described in "*Setting up* (p. 38)." You should also understand the general procedure for extracting and building a BSP, as discussed in "*Board Support Packages (BSPs)* (p. 49)." Be aware, however, that the steps that follow are not the same as the general BSP build steps.

To build your own QNX Apps and Media target image for Intel x86 Bay Trail platforms, on your host system:

1. Set up the environment variables for the QNX SDP, and the development environment for QNX Apps and Media:

   Linux:

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

   Windows:

   ```
   base_dir\qnx660-env.bat
   ```

   where *base_dir* is the directory where you installed the QNX 6.6.0 SDP.

2. Set the *QNX_QT* environment variable to the location of the Qt runtime on your host system:

   Linux:

   ```
   export QNX_QT=qt_base_dir/QNX-qt/Qt-5.3.1-x86
   ```

   Windows:

   ```
   set QNX_QT=qt_base_dir\QNX-qt\Qt-5.3.1-x86
   ```

3. Extract a BSP, then copy everything from the **/prebuilt** directory to the board-specific directory in the ***QNX_DEPLOYMENT_WORKSPACE*** path, as follows. We'll refer to the directory where you extracted the BSP as *bsp_dir*:

   ```
   cd bsp_dir
   cp -r prebuilt/* $QNX_DEPLOYMENT_WORKSPACE/target/boards/baytrail/
   ```

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

   ```
   mkdir output_dir
   ```

5. From the ***QNX_DEPLOYMENT_WORKSPACE*/infra/utils/scripts/** directory, run `mksysimage.sh` (Linux) or `mksysimage.bat` (Windows) to generate a system image file (**.img**), using the path to the x86 IPL:

   ```
   mksysimage.sh -P AnM -o output_dir -k "-b
       base_dir/target/qnx6/x86/boot/sys/ipl-diskpc1" baytrail.ext -f
   ```

   where *output_dir* is the location of the new image.

You should now have an image file (**.img**) ready to write to a USB key so you can transfer it to your target. For instructions, see "*Downloading and transferring a reference image* (p. 13)."

Chapter 4
Understanding the `mksysimage` process

If you are creating target images, you should understand how **mksysimage** generates these images.

The following explanations help you understand what **mksysimage.py** does when it generates an image: how it uses BSPs, the QNX Apps and Media directory structure, IFS files, and search paths.

The QNX Apps and Media directory structure

It is important to understand the directory structure used in the QNX SDK for Apps and Media. You need to know where components are located when you create your target image.

The deployment tree

After it has been set, the *QNX_DEPLOYMENT_WORKSPACE* environment variable references the **deployment** directory. This directory has two branches:

infra

Contains files that determine what goes into an image (the configuration files used to generate an image).

target

Contains the binaries, libraries, and configuration files that may be included in a target image.

Both **infra** and **target** have subdirectories, as shown in the figure below.

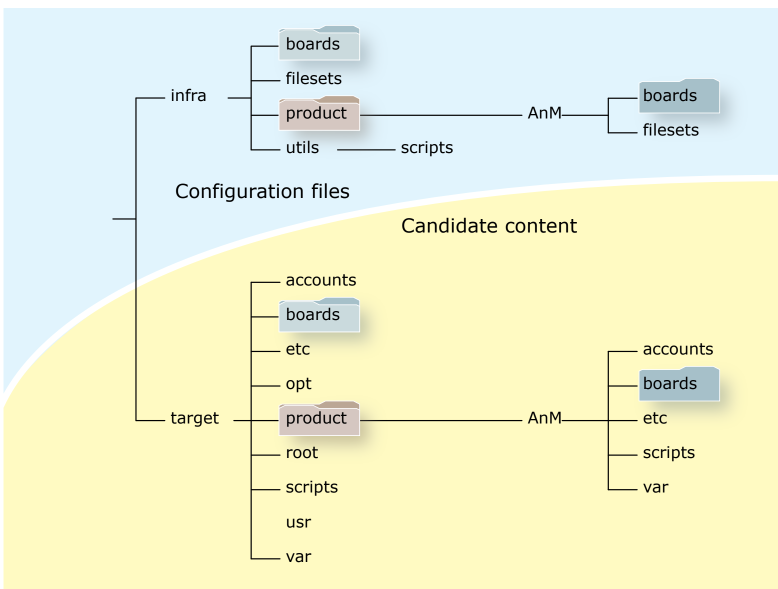

Figure 7: The directory structure for an Apps and Media image.

The files in the **deployment** subdirectories must be listed in a fileset to be included in the image. If a file goes on a target, use ***$QNX_DEPLOYMENT_WORKSPACE*/target/**. If a file is *not* part of a target image, use the directory structure located under ***$QNX_DEPLOYMENT_WORKSPACE*/infra/**.

Board Support Packages (BSPs)

If you are rebuilding a QNX Apps and Media target image, you will need to download and build the QNX BSP for your target platform.

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. To learn more about BSPs, see "Working with a BSP" in the *Building Embedded Systems* guide.

> If you haven't modified the BSP, you can use the prebuilt binaries provided by QNX.

Building a BSP (command line)

After you have installed the QNX SDP on your host system, you can download platform-specific BSPs from the QNX website. You can then either unzip the archive and build it on the command line, or import it into the IDE and unzip and build it there.

These instructions are for building a BSP from the command line on either a Linux or Windows host system. You may also want to refer to the BSP *User Guide* for the BSP for your target platform. These guides provide details about switch settings, drivers commands, and so on.

> Unless otherwise specified, these instructions assume a Linux host. If you are working on Windows host run the commands in the `bash` shell.

To build a BSP for QNX SDP 6.6:

1. Set your environment variables, as instructed in "*Set environment variables for image creation* (p. 34)".
2. Download a QNX SDP 6.6 BSP from the QNX website at ***http://community.qnx.com/sf/sfmain/do/viewProject/projects.bsp*** to a new directory in the SDP host environment (the archive unzips to the current directory).

   For example, you can use the following directory structure:

   ***$QNX_TARGET*/root/bsps/ *my_bsp*/**

   The BSP file will be named like this:

   BSP_ *board_name*_SVN *xxxxxx*_JBN *yy*.zip

   where ***board_name*** is the name of the board, ***xxxxxx*** is the SVN ID for the BSP, and ***yy*** is a unique ID for the BSP.

3. Navigate to the directory where you saved the BSP and extract the BSP archive file:

   ```
   unzip bsp_filename
   ```

4. Change to the root directory of the unzipped BSP, then make and install the BSP:

```
make
make install
```

5. To use the newly generated BSP binaries, copy ***bsp_base_dir*/install** to ***$QNX_DEPLOYMENT_WORKSPACE*/target/boards/** ***board_name***.

For information about the BSP directory structure and where to find key files, see "*BSP directory structure* (p. 50)".

Permissions on a Windows host

When running on a Windows host, `mkifs` can't get the *execute*(*x*), *setuid* (set user ID), or *setgid* (set group ID) permissions from the file, when modifying **.build** files. Use the `perms` attribute to specify these permissions explicitly. You might also have to use the `uid` and `gid` attributes to set the ownership correctly. To determine whether a utility needs to have the *setuid* or *setgid* permission set, see the utility's entry in the QNX SDP *Utilities Reference*.

BSP directory structure

The information below should help you find files you need in a BSP.

BSP directories

When a BSP is extracted from its zip file, it is organized into the following directories:

***bsp_base_dir*/image**

Directory for the QNX IFS, which is the image suitable for running on the target device. Any other related binaries (such as an IPL or combined IPL/IFS image) are also created in this directory. In addition, the generated IFS buildfile will also reside in this directory after the BSP builds. By default, this directory also contains a prebuilt OS image.

***bsp_base_dir*/install**

Location to which the contents of the ***bsp_base_dir*/prebuilt** directory are copied when a BSP is built. Any binaries generated as a result of building the BSP source (contained in the BSP's ***bsp_base_dir*/src** directory) are also copied to the ***bsp_base_dir*/install** directory. (The `mkifs` utility will gather its content from the **deployment** directory and its subdirectories.)

***bsp_base_dir*/prebuilt**

Various header files necessary for building the source components of the BSP, as well as prebuilt binaries or libraries whose source code is not included with the BSP.

***bsp_base_dir*/src**

The source code for device drivers, libraries, and utilities.

Location of key files

After you build the BSP, you'll find key files in the following locations, where **$*BSP_ROOT_DIR*** is the name of the directory you extracted the BSP archive in, and **$*CPU_VARIANT*** is the CPU architecture for which the BSP is designed (e.g., **armle-v7** or **x86**):

File(s)	Location
Buildfile (core OS)	**$*BSP_ROOT_DIR*/images**
IPL	**$*BSP_ROOT_DIR*/install/$*CPU_VARIANT*/boot/sys** The files in this location are generated only when you run **mkflashimage**.
Libraries (DLL drivers), such as audio, graphics, and network	**$*BSP_ROOT_DIR*/install/$*CPU_VARIANT*/lib/dll** The files in this location are generated only when you compile the libraries.
Generic header files (not architecture-specific)	**$*BSP_ROOT_DIR*/install/usr/include**
Source code for different drivers (**sbin** drivers), such as serial, flash, block, PCI, PCMCIA, and USB	**$*BSP_ROOT_DIR*/install/$*CPU_VARIANT*/sbin**

Understanding search paths

To find the files needed to build an image, `mksysimage` searches a specific set of paths.

Search paths are a simple method for specifying which files get included in an image when it is built. When `mksysimage.sh` or `mksysimage.bat` calls `mksysimage.py`, this Python script uses **mktar.py** and **gen-ifs.py** to build an image, these scripts examine the search paths in sequence and use the first instance found of the file they need.

For example, if `gen-ifs.py` needs **foo.bin** to build an image and there are two copies of this file:

***$QNX_DEPLOYMENT_WORKSPACE*/target/product/AnM/boards/omap5uevm.ext/ifs/foo.bin**

and:

***$QNX_DEPLOYMENT_WORKSPACE*/target/foo.bin**

assuming that these search paths are in the order above, `mksysimage.py` will use the file in:

***$QNX_DEPLOYMENT_WORKSPACE*/target/product/AnM/boards/omap5uevm.ext/ifs/foo.bin**

because this search path is first in the list.

Search paths for QNX Apps and Media start from the most specific (product and board) and work down to the most general (the QNX Neutrino OS). If a file **foo.bin** is available for both a specific product and board but also the OS, `mksysimage.py` will use the product and board version of the file (because it is in a search path listed earlier).

Thus, when customizing your target image, you can add files to paths at the top of the list to replace files found in paths listed later.

Organization of search paths

For QNX Apps and Media, search paths are generally organized as follows:

1. product-specific files
    a. board-specific files
        a. *cpu_dir*
    b. non-board-specific files
        a. *cpu_dir*
2. non-product-specific files
    a. board-specific files
        a. *cpu_dir*
    b. non-board-specific files
        a. *cpu_dir*
    c. *cpu_dir*
3. deployment files and SDP-specific files (those located in **qnx660/target/qnx6**)
    a. CPU-specific

Example search path list

Below is an example list of search paths. Note that the *QNX_TARGET* enviroment variable is set to ***base_dir*/target**, and that the paths for the OS and Qt are last in the list, so the build will use files in these paths only if no file of the same name has been found in the more specific paths.

```
$QNX_DEPLOYMENT_WORKSPACE/target/product/AnM/boards/omap5uevm.ext/armle-v7
$QNX_DEPLOYMENT_WORKSPACE/target/product/AnM/boards/omap5uevm.ext/ifs
$QNX_DEPLOYMENT_WORKSPACE/target/product/AnM/boards/omap5uevm.ext
$QNX_DEPLOYMENT_WORKSPACE/target/product/AnM/boards/common/ifs
$QNX_DEPLOYMENT_WORKSPACE/target/product/AnM/boards/common
$QNX_DEPLOYMENT_WORKSPACE/target/product/AnM/armle-v7
$QNX_DEPLOYMENT_WORKSPACE/target/product/AnM
$QNX_DEPLOYMENT_WORKSPACE/target/boards/omap5uevm/armle-v7
$QNX_DEPLOYMENT_WORKSPACE/target/boards/omap5uevm/
$QNX_DEPLOYMENT_WORKSPACE/target/boards/common
$QNX_DEPLOYMENT_WORKSPACE/target/armle-v7
$QNX_DEPLOYMENT_WORKSPACE/deployment/target
$QNX_DEPLOYMENT_WORKSPACE/target/qnx6/armle-v7
$QNX_DEPLOYMENT_WORKSPACE/target/qnx6
qt_base_dir/QNX-qt/Qt-5.3.1-armle-v7
```

To see all the paths used for the files that `mksysimage.py` uses, specify a verbosity greater than six. For example, use `-vvvvvvv`, and the output will display the search paths used by `gen-ifs.py` and `mktar.py`.

The **mktar.py** and **gen-ifs.py** scripts will not display a search path if it doesn't exist. In otherwords, they don't issue any kind of "Missing search path!" error.

Chapter 5
Startup

A QNX Apps and Media system starts up in several stages.

The startup process involves multiple tasks, completed in sequence. These tasks can be grouped into three stages:

1. Hardware startup — board and chip dependent; software has no control over this stage.
2. Initial startup — the IPL, IFS, and any services and applications that must be accessed immediately. To customize this stage, you need to modify the build file and rebuild the target image.
3. Everything else — The System Launch and Monitor (SLM) service uses its configuration files to know what to launch to prepare the environment for the HMI. You don't need to rebuild the target image to change the launch order of applications; you just need to change the SLM configuration files.

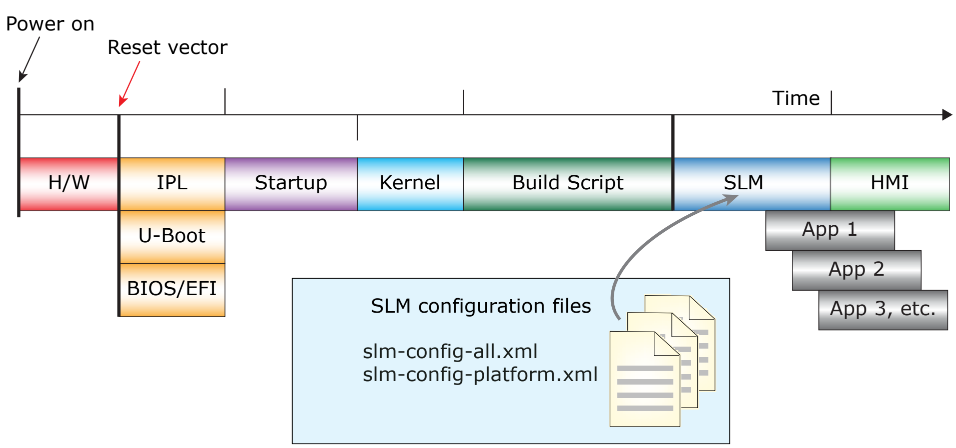

Figure 8: The startup sequence for a QNX Apps and Media system

Phase locked loop (PLL)

PLL is part of the hardware startup process. It refers to how long it takes for the first instruction to begin executing after power is applied to the processor.

Most CPUs have a PLL that divides the main crystal frequency into all the timers used by the chip. The time that the PLL takes to settle to the desired frequencies often represents the largest portion of the chip's startup time.

The PLL stage is independent of any OS and varies from CPU to CPU; in some cases, it takes as long as 32 milliseconds. Consult your CPU user manual for the exact timing.

Initial Program Loader (IPL)

QNX provides a standard, bare-bones IPL that performs the fewest steps necessary to configure the memory controller, initialize the chip selects and/or PCI controller, and configure other required CPU settings. Once these steps are complete, the IPL copies the startup program from the image filesystem (IFS) into RAM and jumps to it to continue execution.

The IFS contains the OS image, which consists of the startup program, the kernel, the build scripts, and any other drivers, applications, and binaries that the system requires. Because you can control what the IFS contains, the time for the copying stage varies, but it typically constitutes the longest part of the kernel boot process. In extreme cases where the system contains a very large image and has no filesystem other than the IFS, this stage can take a long time (10 seconds or more).

You can indirectly adjust the length of this phase by reducing the size of the IFS. To add, remove, or configure files stored in the IFS, modify the **.build** files used by `gen-ifs`. You can also compress the image to make the IFS smaller (with the additional overhead of decompression, which you can speed up by enabling the cache in the IPL).

Typically, the bootloader executes for at least 6 milliseconds before it starts to load the OS image. The actual amount of time depends on the CPU architecture, on what the board requires for minimal configuration, and on what the chosen bootloader does before it passes control to the startup program.

Some boards come with another bootloader, such as U-Boot. These bootloaders aren't as fast as the QNX IPL, because this IPL has been specifically tuned for QNX systems.

For more information on the IPL and how to modify it for your purposes, see “Writing an IPL Program” in the *Building Embedded Systems* guide.

Startup program (including the kernel)

The first program in a bootable OS image is a startup program whose purpose is to initialize the hardware, the system page, and callouts, then load and transfer control to the kernel (**procnto** or **procnto-smp**). If the OS image isn't in its final destination in RAM, the startup program copies it there and decompresses it, if required.

During bootup, the kernel initializes the memory management unit (MMU); creates structures to handle paging, processes and exceptions; and enables interrupts. Once this phase is complete, the kernel is fully operational and can begin to load and run user processes from the build scripts.

Build scripts

Each board has a different set of build scripts to support different configurations. The build scripts let you specify which drivers and applications to start, and in what order.

You can use the build scripts to launch services or utilities that need to be running very early or that need extra time to load (for example, PPS or disk drivers). Wherever possible, these processes should be started in the background to optimize concurrency and maintain the highest possible utilization of the CPU until the HMI is fully operational.

It's also important to limit what goes into the build script because the build script is included in the IFS, and everything that's added to it increases the IFS size and, therefore, the loading time. Furthermore, SLM is more efficient at launching services and also allows you to monitor and restart services as required.

In the QNX SDK for Apps and Media platform, the build scripts start the following:

- Screen
- audio service

- disk drivers (and then mount the disks)
- the PPS service
- debugging utilities, such as **slogger** and **dumper**
- BSP drivers, such as the serial driver, realtime clock, and other hardware utilities
- SLM and the system debug console

System Launch and Monitor (SLM)

SLM is a service that starts any processes required for the HMI (e.g., **io-pkt**), then starts the HMI. At this point, SLM waits for further instructions. SLM is controlled by a set of configuration files (**slm-config-all.xml** and **slm-config-platform.xml**) that tell it what modules to start and whether there are dependencies within or between those modules. The dependencies of the HMI are defined in the `anm-init` module of the file **slm-config-all.xml**. For more information, see the entry for SLM in the *System Services Reference*.

Initial startup process

You can modify the startup process to improve startup times and customize the launch order of services and applications that are started before SLM takes over.

The buildfile

When a BlackBerry 10 OS system is built, it uses a *buildfile* to generate an IFS. This buildfile specifies:

- the files and commands to include in the IFS
- the startup order for the executables
- the loading options for the files and executables
- the command-line arguments and environment variables for the executables

Overview of the initial startup process

The following illustration shows an overview of the IFS startup process:

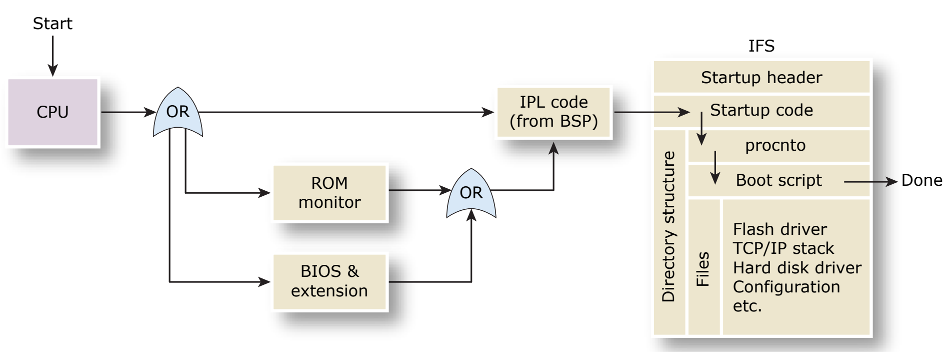

Figure 9: The startup sequence for a BlackBerry 10 OS system

The QNX Apps and Media buildfiles include many smaller **.build** files that the `gen-ifs.py` utility script combines into an output IFS file. For more information, see “*IFS files* (p. 73).”

After the hardware has initialized, startup proceeds as follows:

1. The processor begins executing at the *reset vector*. 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.

   These instructions can be a BIOS, a ROM monitor, or an IPL. If they are a BIOS, then the code will 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 U-Boot, then the ROM monitor jumps to the IPL code.

2. The IPL minimally configures the hardware to create an environment that allows the *startup program* microkernel to run, then locates the IFS and transfers control to the startup program in the image.

The IFS is a file with a directory structure; it contains the OS, your executables, and any data files related to your programs.

3. The startup program configures the system and transfers control to the `procnto` module, which is a combined microkernel and process manager.
4. 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 specified by the `mkifs` buildfile.

When this process is complete, control is handed to the SLM service.

System Launch and Monitor (SLM)

You can use the System Launch and Monitor (SLM) service to modify the launch sequence of applications and services without rebuilding the target image.

The SLM service starts processes required for the HMI (e.g., `io-pkt`), then the HMI itself. It automates process management by running early in the boot sequence and launching complex applications consisting of many processes that must start in a specific order.

SLM uses XML configuration files to determine the appropriate order for starting processes. These files list all the processes for SLM to manage, any dependencies between the processes, the commands for launching the processes, and other properties. The files are located in these subdirectories:

***$QNX_DEPLOYMENT_WORKSPACE*/target/product/AnM/**

The files are:

slm-config-all.xml

Configures services common to all hardware platforms. Located in: **etc/**.

slm-config-modules.xml

An example of how to add new modules. Located in: **etc/**. This file is included in **slm-config-all.xml**.

slm-config-platform.xml

Platform-specific servics, such as board-specific drivers. Located in: **boards/*platform*.*variant*/etc/**

For more information, see the System Launch and Monitor (SLM) entry in the *System Services Reference*.

Example SLM configuration

Below is a section taken from the current contents of the SLM configuration file **slm-config-all.xml**; this section defines support for the PPS service:

```
<!-- pps starts in the IFS, this just blocks until /pps is available -->
<SLM:component name="pps">
    <SLM:command launch="builtin">no_op</SLM:command>
    <SLM:args></SLM:args>
    <SLM:waitfor wait="pathname">/pps</SLM:waitfor>
    <SLM:stop stop="signal">SIGTERM</SLM:stop>
</SLM:component>
```

Chapter 6
Modifying Target Images

If you add pre-installed apps to your system or customize it in any other way, you need to include the modifications in a new image, which you must build and transfer to your target system.

The following explanations should help you understand QNX Apps and Media target images and how they are built so you can incorporate new apps, or implement other modifications. For more information about adding apps, see "*Apps and HMIs* (p. 19)".

Filesets and profiles

Filesets let you specify files, directories, symbolic links, and their permissions to include in the image. Profile files determine which filesets and other components are included in the tar file from which the image is built.

Filesets

The `mksysimage.` script uses *fileset* files to group files into sets that can be easily included in or excluded from an image. Filesets are placed in a number of locations. For a list of the filesets in each location by default, see "*Filesets in the reference image* (p. 62)."

Profile files

When it generates a tar file, **mktar.py** uses profile files to determine what to include in the tar file from which the target image will be created. A *profile* file specifies:

- the filesets to include in the tar file
- the prepackaged applications to include in the tar file
- which fileset goes into which partition in the image

If a profile file specifies an app that is packaged as a **.bar** file, **mktar.py** includes that app in the tar file from which it generates the target image. This app will be included in the image and won't require installation after the system is running.

For example, the default profile files for the OMAP5 EVM board are:

```
[sd]
. . .
profiles=os.xml,dos-sd.xml
. . .
```

They are located under:

$QNX_DEPLOYMENT_WORKSPACE`/infra/product/AnM/boards/`*platform*`.`*variant*`/`

Filesets in the reference image

The tables below list the filesets that are included by default in the QNX Apps and Media reference images.

Filesets in *base_dir*`/deployment/infra/filesets`

The filesets listed in the following table are present only if you have installed the corresponding additional packages from the QNX *Download Center*.

Fileset	Description
basefs.armle.extra.freescale.xml	Third-party video decoder components from Freescale required for the i.MX6 Sabre Smart Devices platform.
basefs.mm.extra.ipod.xml	Configuration and drivers for iPod connectivity.
basefs.omap5uevm.video.xml and **basefs.jacinto6evm.video.xml**	TI video decoder components for the OMAP5 and Jacinto hardware platforms.
basefs.mm.extra.ittiam.xml	Ittiam software video codecs ARMLE-v7 platforms.
basefs.mm.extra.wma9.xml	Software decoding for Microsoft Windows Media Audio 9 format files and streams.

Filesets in *$QNX_DEPLOYMENT_WORKSPACE*`/infra/boards/`*board-specific*

Fileset	Description
basefs.baytrail.video.xml	Video codecs for the x86 Bay Trail.
basefs.imx61sabre.dvfs.xml	DVFS for the i.MX6 SABRE Smart Devices.
basefs.omap5.dvfs.xml	DVFS for the OMAP5 EVM.
basefs.*board-specific*.xml	Board-specific drivers and graphics components.
dosfs.*board-specific*.boot.xml	Board-specific IFS files (depending on the board, may also include MLO and U-boot).
dosfs.omap5uevm.lvds.boot.xml	LVDS for the OMAP 5 EVM.
rootfs.*board-specific*.xml	Board-specific IFS files.

Filesets in *$QNX_DEPLOYMENT_WORKSPACE*`/infra/product/AnM/boards/beaglebone.ext`

Fileset	Description
rootfs.beaglebone.anm.media.xml	Sample multimedia files for the QNX SDK for Apps and Media for the BeagleBone Black.

Filesets in *$QNX_DEPLOYMENT_WORKSPACE*`/infra/product/AnM/filesets/`

Fileset	Description
basefs.anm.multimedia.xml	Base fileset for the QNX SDK for Apps and Media. Includes the HMI, icons, and associated configuration.

Fileset	Description
basefs.anm.os.xml	Apps and media components in the **scripts** and **usr/lib** directories.
basefs.anm.qt.xml	Qt components for the QNX SDK for Apps and Media.
rootfs.anm.media.xml	Sample multimedia files for the QNX SDK for Apps and Media.

Filesets in *$QNX_DEPLOYMENT_WORKSPACE*/`infra/filesets`

Fileset	Description
basefs.armle.common.debug.os.xml	Debug utilities for ARMLE-v7.
basefs.common.debug.os.xml	OS debug tools.
basefs.common.developer.networking.xml	Networking components, including CURL, FTP, and Telnet.
basefs.common.directories.xml	Highest level OS partition directories.
basefs.common.etc.xml	Base filesystem. Includes configuration for fonts, users and groups, networking, `authman`, `launcher`, OpenSSL, SLM, and other base components.
basefs.common.geolocation.xml	Geolocation service (**ip-provider**).
basefs.common.inputservice.xml	Input service (**libinput**).
basefs.common.multimedia.xml	Multimedia components, such as **mm-renderer**, **mm-sync**, and so on.
basefs.common.os.launcher.xml	The **launcher** service.
basefs.common.os.xml	Base filesystem. Includes components in the **bin**, **lib**, **usr/bin**, **usr/sbin**, and **usr/share** directories.
basefs.common.qt.xml	Base components of Qt.
basefs.common.scripts.xml	Common scripts, including `bar-install` and `shutdown`.
basefs.common.util.xml	Utilities such as the Korn shell (`ksh`), **pipe**, and **cksum**.
basefs.common.webkit.xml	Core WebKit libraries.
basefs.common.weblauncher.xml	The **weblauncher** service.
basefs.fonts.common.xml	Currently Used fonts.

Fileset	Description
basefs.fonts.dejavu.xml	DejaVu fonts package.
basefs.fonts.han.xml	Source Han Sans font package.
basefs.html5.common.torch.webkit.xml	Additional WebKit libraries including the WebPlatform JavaScript API.
basefs.html5.common.webinspector.xml	WebKit Web Inspector components.
basefs.mm.aac.xml	Multimedia AAC components.
basefs.mm.sw.xml	Multimedia screen writer.
rootfs.common.accounts.xml	The **accounts** directory.
rootfs.common.certificates.xml	Common **certmgr** certificates.
rootfs.common.directories.xml	Common directories such as **accounts**, **root**, **apps**, and **var**.
rootfs.common.etc.xml	Components in the **var/etc** directory.
rootfs.common.geolocation.xml	Geolocation configuration.
rootfs.common.os.xml	The **root** and **var** directories.
rootfs.common.pps.xml	The PPS filesystem for the base components (audio, multimedia, application launcher, etc.)

Adding and modifying filesets

You can add new filesets, and new groups and users to existing filesets.

You need to ensure that all files you want to include in an image are added to filesets.

Adding a new fileset

To add a new fileset to an image:

1. Determine which partition the fileset belongs to (e.g., **/base**).
2. In the **basefs.*fileset_name_to_create*.xml** file, add the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<fileset name="basefs.appsandmedia.fileset_name">
```

3. Add any files or symbolic links (symlinks) that you require. For information about adding symbolic links, see "*Adding symbolic links* (p. 67)."
4. Ensure that you terminate the file with the tag `</fileset>`.
5. Add the new fileset to a profile file, such as **os.xml** or **dos-*variant*.xml**.

Adding new groups and users to a fileset

The **groups.xml** and **users.xml** files are the configuration files specific to filesets. When you want to add a new group and/or user, you must update the **groups.xml** and/or **user.xml** file located in ***$QNX_DEPLOYMENT_WORKSPACE*/infra/filesets/config**.

Before you can reference a user (`uid`) or group (`gid`) in a fileset, those entries must exist in the appropriate file.

To add a new group, enter a `<group>` element as follows:

```
<group name="my_group" gid="511"/>
```

To add a new user, enter a `<user>` element as follows:

```
<user name="some_user" uid="900" gid="511"/ home="/some_destination
  shell="/bin fullname="some_name"/>
```

Any new user or group must be added to the corresponding **.xml** file; otherwise, the image generation process will fail.

Some `gid` values are reserved. See the comments in **groups.xml** for the number ranges that are reserved.

Changing file destinations

When a file's location in the host's *$QNX_TARGET* directory structure is different from its location on the target, you can set a new destination for it on the target.

To do this, you can modify a fileset to specify a different destination on the target system. In most cases, you would organize files on the host exaclty as they will be organized on the target. However, the Qt runtime directory structure on the host is not the same as the directory structure required for the target, so this feature is useful for working with Qt runtime.

Files on the target that are not in the same path as on the host system add a level of complexity to your system. We recommend using this feature only if strcitly required, such as for Qt runtime.

Unless otherwise specified, these instructions assume a Linux host. If you are working on a Windows host, run the commands in the `bash` shell

---

To change the destination of files:

1. Navigate to where the filesets are located:

   ```
   cd $QNX_DEPLOYMENT_WORKSPACE/infra/filesets
   ```

2. Open one of the filesets that you wish to modify in an editor (e.g.,**base.common.qt.xml**).

3. For a `file` element, add `“dest”=` and specify a valid location:

   `dest="`*some_location*`"`

   For example, to set an alternative destination for Qt, you might add the following for `dest`:

```
<file name="lib/libQt5QnxAfExtras.so.5.3.2"
        dest="usr/qt5-5.3/lib/libQt5QnxAfExtras.so.5.3.2"
        uid="root" gid="nto" mode="0755"/>
```

4. Save the file.
5. Generate the image. For instructions, see “*Building a target image* (p. 38).”

The new location for the file will be used for the file on the target.

Adding symbolic links

You can instruct `mktar.py` to create symbolic links when it generates an image from a fileset.

Unless otherwise specified, these instructions assume a Linux host. If you are working on a Windows host, run the commands in the `bash` shell.

To add a symbolic link:

1. Locate the filesets in this directory:

   ***$QNX_DEPLOYMENT_WORKSPACE*/infra/filesets**

2. In an editor, open one of the filesets that you wish to modify (e.g., **base.common.multimedia.xml**).
3. Add a `symlink` element. The following example shows three symlink elements:

```
<symlink name="usr/qt5-5.3/lib/libQt5QnxAfExtras.so.5.3"
    target="libQt5QnxAfExtras.so.5.3.2" uid="root" gid="nto" mode="0755"/>
<symlink name="usr/qt5-5.3/lib/libQt5QnxAfExtras.so.5"
    target="libQt5QnxAfExtras.so.5.3.2" uid="root" gid="nto" mode="0755"/>
<symlink name="usr/qt5-5.3/lib/libQt5QnxAfExtras.so"
    target="libQt5QnxAfExtras.so.5.3.2" uid="root" gid="nto" mode="0755"/>
```

4. Save the file.

When `mktar.py` generates an image, it will create the symbolic link.

Image configuration settings

Configuration files let you define files for the system image for a specific platform type, and provide size and partition information.

When you create your own OS image for your platform, you can modify various options in the configuration files used by the ***mksysimage.py*** (p. 68) and ***mkimage.py*** (p. 69) utilities.

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` utility script, see `mksysimage.py` in the *System Services Reference*.

Unless otherwise specified, these instructions assume a Linux host.

An `mksysimage.py` configuration file defines these components:

- an IFS file renamed to **qnx-ifs** and used as the default boot file
- the tar files to generate
- the tar files to include in the image
- a configuration file that defines the image partition sizes

You can find the default configuration file at:

$QNX_DEPLOYMENT_WORKSPACE/**infra/product/AnM/boards/*****platform.variant/ mksysimage/ platform*****-mksysimage.cfg**

For example, the default configuration file for the OMAP5 EVM 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
```

The contents of this configuration file reveal that this OMAP5432 board has an SD image variant called `sd`, which defines the following:

`default-ifs`

The `ifs` image name used as the default bootup IFS (`qnx-ifs`).

`profiles`

The `mktar.py` profiles used to generate tar files. The `image-tars` elements correlate directly with the profiles. The names of the generated tar files have names of the form: *platform-profile_name***.tar**.

`image-tars`

The tar files included in the image.

image-config

The configuration file used for specifying the size of each partition in the resulting image file. The image-configuration files must be in the ***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/*platform.variant*/mksysimage/** directory.

Configuration file for `mkimage.py`

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

For information about running this script, 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 each partition
- path to each partition

For example, the contents of the OMAP5 configuration file (**omap5uevm-sd.cfg**) look like this:

```
[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=1253376
order=2

[data]
path=/
type=178
num_sectors=5322752
order=3
```

The sections of this file define the following:

[disk]

This section doesn't specify a partition, but determines the size of the image and of the partitions. This section is required, must not be empty, must appear first in the file, and must be called `[disk]`.

heads

The number of heads for the data medium used.

sectors_per_track

The number of sectors for each track for the data medium used.

cylinders

The number of cylinders for the data medium.

sector_size

The size of the sectors used to store the data.

[*partition_name*]

A partition in the image. In the example above, `[boot]` is the first partition and contains boot information.

path

Identifies the mountpoint of the partition.

type

Identifies the type of partition. For information about partition types, see "Partitions" in the *System Architecture* guide for BlackBerry 10 OS.

num_sectors

The number of sectors for the partition.

order

The order for the specified partition in the image. If the order is 1, it's the bootable partition.

The example above shows three partitions:

- `[boot]` is of type 12 (FAT), has a partition order of 1 (meaning the first partition in the image), and is 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 Power-Safe), has a partition order of 2, and is located at **/base**.
- `[data]` is of type 178 (QNX Power-Safe), has a partition order of 3, and is located at the root **/**.

Changing partitions

You can edit your board's ***board-media*.cfg** file to change the partitions in your image.

Unless otherwise specified, these instructions assume a Linux host. If you are working on a Windows host, run the commands in the `bash` shell.

To change the partitions in an image:

1. Locate the **.cfg** file for your board. For example, for OMAP5, the file is **omap5uevm-sd.cfg** and it's located in: ***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/omap5uevm.ext/mksysimage/**
2. Open this file in a text editor and modify the various settings as required.

   For example, the OMAP5 configuration file contains this partition:

   ```
   [base]
   path=/base
   type=179
   num_sectors=1253376
   order=2
   ```

   The line `path=/base` identifies the mountpoint for the partition. You can change the partition settings as required. For details on the various settings, see the *configuration file settings list* (p. 69).

Changing image and partition sizes

To change the size of your target image or partitions, you must modify a variant-specific configuration file.

The file that you must modify is: ***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/*platform*/mksysimage/*platform*-*image_variant*.cfg**

For x86 platforms, you should note the following when creating partitions:

- The `mksysimage` utility requires a different argument to be passed in.
- The boot process is different from ARMLE-v7 platforms.
- The IFS gets installed to a different location.

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:

```
    heads
  x sectors_per_track
  x cylinders
  x sector_size
------------------------------
 total maximum size of image
```

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

For the OMAP5432 example for the `sd` variant, the maximum size of the image is 3.9 GB (3.63 GB actual) and is 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

The size of a partition in the example above is calculated as follows:

```
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 the partition is 3724 MB.

IFS files

Image Filesystems (IFS) files are created by **mkifs**. An IFS file contains a bootable OS image. In a QNX Apps and Media system, the **gen-ifs.py** utility generates a build file and then provides it as input to **mkifs** to build an IFS.

Standard QNX BSPs are built by **mkifs**, which reads the build information from a single, large build (**.build**) file. The complexity of the QNX SDK for Apps and Media makes this approach to building an IFS difficult to implement: the same build process must support many different products and platforms.

To make these complex builds easier to manage and configure, QNX Apps and Media uses many small build files, with each file configuring a specific part of the build. These small build files can be re-used and combined for different products and platforms.

IFS directories

In the directory structure defined for generating images, there are five locations where you'll find IFS directories.

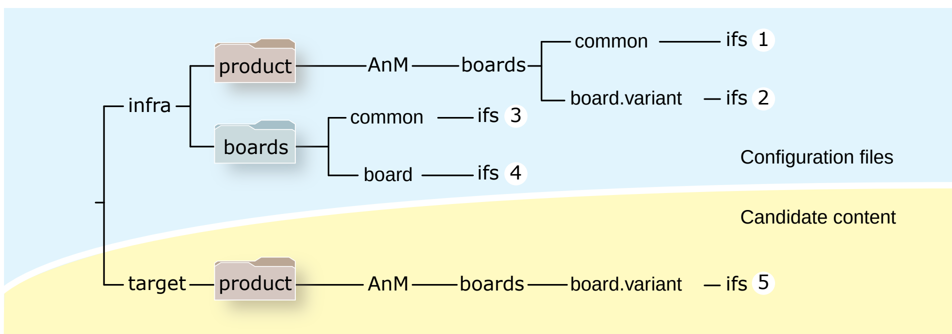

Figure 10: The `ifs` directories in a product image

The preceeding illustration shows the location inside ***$QNX_DEPLOYMENT_WORKSPACE*** of various IFS directories related to the generation of IFS files. Instances 1 to 4 in the "Configuration files" part of the diagram are the directories with the *input* **.build** files, as follows:

1. product-specific but non-board-specific files
2. product-specific and board-specific files
3. non-board-specific and non-product-specific files
4. board-specific but non-product-specific files

Instance 5 in the "Candidate content" part of the diagram is the **ifs** directory where the image generation scripts put the final generated IFS image.

Modifying the IFS for a specific board

To modify the IFS for a specific board, do the following:

1. Modify this IFS config file:
   - ***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/*platform*/ifs/**

2. Modify the appropriate buildfiles, which are located in these directories:
   - ***$QNX_DEPLOYMENT_WORKSPACE*/infra/boards/*platform*/ifs/**
   - ***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/*platform*/ifs/**

Adding an IFS

If you need to add a new IFS to the build, you can edit the IFS configuration file: ***platform*-ifs.cfg**, which is located at:

***$QNX_DEPLOYMENT_WORKSPACE*/infra/product/AnM/boards/*platform*/ifs/**

In this configuration file, you can add a new IFS, change the order of the buildfile, and add a new buildfile to the IFS.

For example, the configuration file for OMAP5 (**omap5uevm-ifs.cfg**) contains the following default buildfiles:

```
[omap5-sd.ifs]
omap5-base.build
omap5-start-i2c.build
omap5-start-disk-sd.build
screen-omap5uevm-diskLibs.build
omap5-apps-and-media.build
apps-and-media.build
omap5-start-audio.build
screen-common-diskLibs.build
omap5uevm.build
generic.build
script.build
```

To add a new IFS:

1. Create a new `[`*variant*`.ifs]` section in the configuration file.
2. Add this section to the fileset file as well.
3. Run `mksysimage` to regenerate the image. This script calls other scripts to combine the buildfiles listed in the **ifs** section (in the order that they are listed), create the **.image** files for the partitions on the target, and combine these image files to create the target image as a single **.img** file.

IFS for x86 platforms

The **/.boot/** directory contains the generated IFS files.

The current boot support is for x86 PC partition-table-based (the same base system as current booting) with a BIOS that supports INT13X (LBA). Files placed into the **.boot** directory are assumed to be boot images created with `mkifs`. The name of the file should describe the boot image.

For more information, see the BSP *User Guide* for your platform.

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 script file 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*.

Troubleshooting tips

You can use various **mksysimage.py** options to have this script complete only select parts of the image generation process.

Building an entire target image that includes everything from the IFS to the apps can be time-consuming. When you modify components or troubleshoot, you can use combinations of **mksysimage.py** options to build only parts of the image:

`--no-mkimage`

Prevents `mksysimage.py` from generating image files. The `mksysimage.py` script stops after it has generated the tar files. Use this option when you need to confirm that your changes will be correctly incorporated into the image after you have changed a component such as a fileset or a profile.

`--no-mkimage --no-mktar`

Prevents `mksysimage.py` from generating tar files or image files. The `mksysimage.py` script stops after it has generated the IFS files. Use this option when you need to confirm that your IFS changes will work.

`--keep-partition-images`

Prevents `mksysimage.py` from deleting intermediate partition images. Use this option when you need to keep the existing partition images. For example, if you are customizing the configurations used by **mkimage.py** and want to examine them before incorporating them into an image (**.img**) file.

Index