QNX SDP 6.6.0 BSPs

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: Thursday, February 20, 2014

Table of Contents

About This Guide

QNX SDP 6.6.0 BSPs describes what's new with the QNX SDP 6.6.0 BSPs in general, and explains how to update your QNX 6.5.0 BSPs to be able to build and run them on QNX SDP 6.6.0. For specific BSPs, please refere to the BSP-specific guides that accompany each BSP.

To find out about:	See:
About QNX BSPs in general	*Overview of a BSP* (p. 9)
Where to find more documentation about BSPs and building embedded systems	*Before you begin* (p. 11)
About QNX 6.6 BSPs	*What's new with QNX SDP 6.6 BSPs* (p. 13)
The structure of QNX 6.6 BSPs	*Structure and contents of a QNX SDP 6.6 BSP* (p. 14)
How to prepare prepare a bootable SD card	*Preparing a bootable SD card* (p. 17)
Procedure to update a QNX BSP	*Process to update a BSP to QNX SDP 6.6* (p. 21)
How to get started updating your QNX BSP	*Download, extract, and build an existing BSP* (p. 23)
Reduce the number of warnings	*Clean Up Remaining Warnings* (p. 33)

If you are using this guide to help you update an existing QNX 6.5.0 or 6.5.0 SP1 BSP to work within a QNX SDP 6.6 environment, before you begin take a close look at the QNX webpage (www.qnx.com) to see if the BSP that you're attempting to update (or perhaps one that's very similar) has already been developed for QNX SDP 6.6. Check back often because new QNX BSPs are regularly developed and posted.

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`
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.

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

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 use a forward slash (/) as a delimiter in all 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
Overview of a BSP

After you've installed the QNX Neutrino RTOS, you can download processor-specific Board Support Packages (BSPs) from our website, *http://www.qnx.com/*. These BSPs are designed to help you get the QNX Neutrino RTOS running on certain platforms.

QNX SDP 6.6 includes updated BSPs on our download site for many of the most popular and current BSPs for `armle-v7` targets; these BSPs have been updated and validated to build and operate within a QNX SDP 6.6 target environment. Although the QNX BSP download page may contain QNX 6.5.0 or 6.5.0 SP1 BSPs that haven't been migrated to QNX SDP 6.6, you might want to use them in a QNX SDP 6.6 environment. This guide describes the types of issues you might encounter when you attempt to update an older QNX 6.5.0 or 6.5.0 SP1 BSP, and it provides instructions on how to cleanly build a QNX 6.5.0 or 6.5.0 SP1 BSP within the QNX SDP 6.6 environment.

This document is a guideline only; it will likely not cover all of the possible issues that you might encounter for every current QNX 6.5.0 or 6.5.0 SP1 BSP. However, its purpose is to address **the most common issues** that you might encounter when you attempt to build an existing QNX 6.5.0 or 6.5.0 SP1 BSP within your QNX SDP 6.6 environment.

A BSP typically includes the following:

- IPL — minimally configures the hardware to create an environment that will allow the startup program, and consequently the microkernel, to run.
- startup — it's purpose is to copy and decompress the image, if necessary, configure hardware, determine system configuration, and start the kernel.
- default buildfile — specifies any file and commands to include in the image, the startup order for the executables, the loading options for the files and executables, as well as the command-line arguments and environment variables for the executables.
- networking support
- board-specific device drivers, system managers, utilities, etc.

The BSP is contained in an archive named after the industry-recognized name of the board and/or reference platform that the BSP supports (*`soc_vendor-soc-board`*`.zip`). BSP packages are available for QNX Neutrino, Windows, or Linux hosts.

To use a BSP, you must either unzip the archive and build it on the command line, or import it into the IDE.

The BSP components are provided in source code form, unless there are restrictions on the source code, in which case the component is provided only in binary form. BSPs are provided in a `zip` archive.

The QNX community website has more information about BSPs; see *http://community.qnx.com/sf/sfmain/do/viewProject/projects.bsp*.

Chapter 2
Before you begin

A number of procedures and guidelines are common to all QNX SDP 6.6 BSPs. This information is presented here, rather than duplicating it in each BSP-specific user guide.

About the information in this guide and in the BSP-specific user guides

This user guide presents the guidelines and procedures that are common to all QNX SDP 6.6 BSPs. For information about a specific BSP, please refer to the user guide that accompanies that BSP.

Hardware documentation

Before you begin working with your BSP, you should review the documentation for your particular board's hardware and boot loader. This information is normally provided by the board vendor. The BSP-specific user guide provided with each QNX SDP 6.6 BSP includes, where possible, a link to a vendor web site, or third-party site where you'll find general information about the BSP's hardware and firmware.

Software documentation

QNX SDP 6.6 documentation can be found in the QNX Infocenter. Several documents in the Infocenter are directly relevant to embedding a QNX system on target hardware. Before starting work with your BSP, you should be familiar with the following documentation:

Manual	Chapter	Topics covered
Building Embedded Systems	"Overview of *Building Embedded Systems*"	General overview of the QNX OS on embedded targets
Building Embedded Systems	"Working with a BSP"	How to install and build QNX BSPs
Building Embedded Systems	"Making an OS image"	How to work with QNX Image File System (IFS) images and how to modify them
QNX Momentics IDE User's Guide	"Overview of the IDE"	Introduction to working with the QNX Momentics

Manual	Chapter	Topics covered
		Integrated Development Environment (IDE)
QNX Momentics IDE User's Guide	"Build images"	How to use the QNX Momentics IDE to generate images for embedded targets

Chapter 3
What's new with QNX SDP 6.6 BSPs

For the most part, BSPs for QNX SDP 6.6 work the same way as they did under previous QNX OS versions. Some important changes have been introduced, however.

About prebuilt binaries and libraries prior to QNX SDP 6.6

Prior to the release of SDP 6.6, QNX did not always supply prebuilt binaries and libraries for its BSPs. If the source code for a BSP component was included with the BSP, QNX didn't supply prebuilt binaries and libraries for that component. If the source code for a component was *not* included with a BSP, QNX supplied prebuilt binaries and libraries for that component.

QNX SDP 6.6 prebuilt binaries and libraries

Starting with QNX SDP 6.6, QNX supplies prebuilt binaries and libraries for *all* BSP components (device drivers, libraries, utilities, etc.). These are prebuilt and packaged with the BSP, regardless of whether or not the source code for those components is included with the BSP. This new packaging model ensures that our customers receive exactly the same checksum-compatible BSP components that we tested. Even if you choose to modify a BSP component by changing the source code and building your own version, you'll always have a copy of the original BSP version available in the BSP archive.

Ensuring that modified components are included in the IFS image

As with previous QNX OS versions, with QNX SDP 6.6, when a BSP is built, the entire contents of the BSP's `/prebuilt` directory gets copied into its `/install` directory. This means that all BSP components mentioned above can be found in the `/install` directory. This directory is the first one searched when a target IFS image is generated.

However, the binaries, libraries, and utilities that get built from the source code included with the BSP are no longer copied to the `/install` directory by default. This means that if you want to modify the BSP source code, making changes to specific device drivers or other components, you must execute a `make install` command when building that code. This command is required so that the resultant binaries, libraries, and utilities get copied into the BSP's `/install` directory, where they can be found and included when the IFS image is built.

Structure and contents of a QNX SDP 6.6 BSP

BSPs can be extracted to, and built from, any directory. You can do this with the QNX Momentics IDE or with the command line.

BSPs and the QNX SDP 6.6 host installation

QNX BSPs are packaged to operate independently of the host and target environments of a standard QNX SDP 6.6 installation. BSPs can be extracted to, and built from, any directory.

When the BSP attempts to build a target image (for example, if your IFS build file includes a component that is not contained within the BSP itself) the BSP may scan QNX SDP's `/target` directory for binaries or libraries, but it won't overwrite anything in your QNX SDP 6.6 installation's `/host` or `/target` directories. Any binaries, libraries, or images that were generated when the BSP was built will remain in the BSP directory from which where they were extracted.

Using the QNX Momentics IDE or the command line

You can use the QNX Momentics Integrated Development Environment (IDE), or the command line, to work with QNX BSPs. The procedures for importing, extracting, building, and modifying QNX BSPs using either method are documented in the *Building Embedded Systems* guide. For instructions on how to use the IDE to work with your BSP, see “Using BSPs in the IDE”. For instructions on how to use the the command line to work with your BSP, “Using BSPs on the command line”.

If you are working with the IDE and modify your BSP, you will still need to use the command line to build the BSP.

Location of the BSP contents

If you used the QNX Momentics IDE to extract the BSP contents, these contents will be found in the location you specified when importing the BSP. By default, it will be in a sub-directory within the `ide- 5.0-workspace` directory of your host environment. If you manually extracted the BSP, using the command-line `unzip` utility, the BSP contents will be in the directory where you extracted the BSP archive.

Whether you use the IDE or the command line, the stucture and contents of the BSP are the same. Typically, extracting a BSP's `.zip` archive creates the following directories:

- `/src`
- `/prebuilt`
- `/install`

- `/images`

`/src`

The `/src` directory contains all source code that ships with the BSP. Source code is organised into sub-directories, such as `/hardware`, `/lib`, `/utils`, etc. When the BSP is built, all the source code within the BSP (including the source code in these sub-directories) is built.

`/prebuilt`

The BSP archive is provided with all of the BSP components prebuilt and stored in the `/prebuilt` directory. If you build the BSP, the contents of the `/prebuilt` directory are copied to the BSP's `/install` directory.

`/install`

The `/install` directory is empty until the BSP is built. When the BSP is built, the entire contents of the `/prebuilt` directory are copied into the `/install` directory. When the IFS image is being generated, the`/install` directory is the first location scanned for components that are specified in the BSP's build file. If the IFS generation process doesn't find a component in the`/install` directory, it scans the QNX SDP 6.6 host's `/target` directory for the component.

If you want to modify BSP components by changing or building any of the BSP source code, you must use the `make install` command when building the relevant source code. This command will cause your modified components to be copied into the BSP's `/install` directory, overwriting the prebuilt versions that were supplied with the BSP.

`/images`

The`/images` is the location of:

- the BSP's default build file after the BSP has been built
- any boot images (such as the IFS image, the IPL binary, etc.) after they have been generated

The table below shows the default locations of common BSP components. `${BSP_ROOT_DIR}` is the name of the directory where you extracted the BSP archive. `${CPU_VARIANT}` is the specific CPU architecture that this BSP is targeted for. For SDP 6.6, the supported CPU variants are **armle-v7** and **x86**.

File	Location
Prebuilt OS image	`${BSP_ROOT_DIR}/images`

File	Location
IPL and/or startup binaries	`${BSP_ROOT_DIR}/install/${CPU_VARIANT}/boot/sys`

Source code for device drivers, libraries, and utilities	`${BSP_ROOT_DIR}/src/hardware/...`
	`${BSP_ROOT_DIR}/src/lib/...`
	`${BSP_ROOT_DIR}/src/utils/...`

Prebuilt libraries	`${BSP_ROOT_DIR}/prebuilt/${CPU_VARIANT}/lib/dll}` `${BSP_ROOT_DIR}/prebuilt/${CPU_VARIANT}/usr/lib}`

Prebuilt binaries	`${BSP_ROOT_DIR}/prebuilt/${CPU_VARIANT}/sbin`
	`${BSP_ROOT_DIR}/prebuilt/${CPU_VARIANT}/bin`
	`${BSP_ROOT_DIR}/prebuilt/${CPU_VARIANT}/usr/bin`

Generic header files (not architecture-specific)	`${BSP_ROOT_DIR}/install/usr/include`

About the prebuilt boot image

After you have extracted the BSP (but before you've built it), you will find a prebuilt QNX IFS image in the BSP's `/images` directory. This prebuilt IFS image is provided as a convenience. It allows you to quickly verify the operation of the QNX OS on your target board, by simply transferring the prebuilt image directly to your target, skipping the step of building the BSP.

When you actually build the BSP, this prebuilt image will be overwritten by a new IFS that gets generated as part of the BSP build process. You may, therefore, want to make a copy of the prebuilt image for future reference. If you forget to make a copy of the prebuilt image, you can still recover it, by simply extracting the BSP from the original `.zip` archive into a new directory.

Preparing a bootable SD card

Many target systems supported by QNX SDP BSPs are capable of loading the QNX IFS image from and SD or micro-SD card. You can prepare a bootable SD card from either of the supported SDP 6.6.0 host environments (Linux and Windows).

About preparing a bootable SD card

Many of the target systems supported by QNX SDP 6.6 BSPs are capable of loading the QNX IFS boot image from an SD or micro-SD card. Many of these targets are also capable of executing the primary boot loader (whether it's U-Boot or the QNX Initial Program Loader (IPL)), directly from the SD card. In most cases, a DOS-compatible FAT32 partition is required on the SD card.

This section presents instructions for preparing your SD card with a DOS FAT32 filesystem, for each of the supported SDP 6.6 host environments (Windows and Linux). In either case, you will need a host PC that has an SD slot, or else use an external USB SD card adapter.

Preparing a bootable SD card (Windows 7)

If your SD card does not already contain a bootable (active) FAT32 partition, you can use the Windows 7 Disk Management utility to create one.

Creating a bootable FAT32 partition in a Windows 7 environment

The default Windows formatting option that appears when you insert a blank (or unrecognized) SD card into a Windows PC is not sufficient to format the SD card with a bootable partition. While it can create a DOS/FAT32 partition, it does not provide an option for making this partition “Active”, or bootable. If your SD card does not already contain a bootable (active) FAT32 partition, you can create one with the Windows 7 Disk Management utility, as follows:

1. Open the Windows Control Panel, and run **Administrative Tools**.
2. Run the **Computer Management** application.
3. On the left pane, under the **Storage** category, select **Disk Management**.
4. Insert the SD card that you will use to put the QNX images on, and identify the correct volume as it appears in the “Disk Management” window.
5. Select the correct drive by right-clicking on it, and choose the **Format** option. Once the format operation is complete, under the “Status” column, it should show something like: Healthy (Active, Primary Partition).

Marking the FAT32 bootable partition active

If the SD card doesn't show the partition as “Active”, you will need to set it to active. With many SD cards, all you need to do is:

1. Right-click on the partition icon.
2. Select **Mark Partition as Active**.

With some SD cards, however, after you have performed the steps to format the card, the option to **Mark Partition as Active** will not be available. (It will be greyed out). If this is the case for your SD card, you will need to complete some additional steps to make the bootable FAT32 partition active:

1. Open a DOS command prompt (**Start Menu > All Programs > Accessories > Command Prompt**).
2. Run the **diskpart** utility and run the **list disk** command to get a list of available drives, then identify your SD card on this list (in this example, Disk 3):

```
DISKPART> list disk

Disk ###     Status       Size       Free      Gyn      Gpt
--------     ------       ----       ----      ---      ---
Disk 0       Online       40 GB      20 GB
Disk 1       No Media      0  B       0  B
Disk 2       No Media      0  B       0  B
Disk 3       Online       14 GB       0  B
```

3. Select the disk to target for additional activities, by running the following command:

```
DISKPART > select disk 3
Disk 3 is now the selected disk.
DISKPART >
```

4. Run the **list partition** command, to see a list of the available partitions (we'll use Partition 1):

```
DISKPART > list partition

Partition ###     Type        Size         Offset
-------------     -------     ------       -------
Partition 1       Primary     14 GB        4096 KB
```

5. Select this partition:

```
DISKPART> select partition 1
Partition 1 is now the selected partition.
DISKPART>
```

6. Make the partition active:

```
DISKPART> active
DiskPart marked the current partition as active.
```

7. Verify that the partition is active:

```
DISKPART> list partition
Partition ###     Type        Size         Offset
-------------     -------     ------       -------
*Partition 1      Primary     14 GB        4096 KB
```

The "*" beside the partition name indicates that the partition is active. You can now eject the SD card, and proceed with copying files to the DOS/FAT32 partition on the disk.

Preparing a bootable SD card (Linux Ubuntu)

If your SD card does not already contain a bootable (active) FAT32 partition, you can create one from a Linux Ubuntu host.

Creating a bootable FAT32 partition in a Linux Ubuntu environment

The following procedure provides a quick, step-by-step example of the procedure you can use from a Linux Ubuntu terminal to prepare an SD card with a bootable DOS / FAT32 filesystem.

- This example uses the SD card `/dev/sdd`. You can use the `mount` command to determine your SD card's actual device name.
- We are working with the SD card as a whole (`/dev/sdd`), not a partition on the SD card (e.g. `/dev/sdd1`).

1. Display disk information and show the existing partitions, if any:

```
/home/user/> sudo fdisk /dev/sdd

Command (m for help): p

Disk /dev/sdd: 15.9 GB, 15931539456 bytes
64 heads, 32 sectors/track, 15193 cylinders, total 31116288
sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x938b698a

Device Boot      Start         End      Blocks   Id  System
```

2. Change default unit to cylinders:

```
Command (m for help): u
Changing display/entry units to cylinders
```

3. Create a new, empty DOS partition table:

```
Command (m for help): o
Building a new DOS disklabel with disk identifier 0xdf0e79d5.
Changes will remain in memory only, until you decide to write
them.
After that, of course, the previous content won't be recoverable.


Warning: invalid flag 0x0000 of partition table 4 will be
corrected by w(rite)
```

4. Add a new partition (Instructions and comments are in parenthesis):

```
Command (m for help): n
Partition type:
p   primary (0 primary, 0 extended, 4 free)
```

```
e   extended
Select (default p): p    (The new partition is a primary
partition.)
Partition number (1-4, default 1): (Press Enter for default.)
Using default value 1
First cylinder (2-15193, default 2): (Press Enter for default.)
Using default value 2
Last cylinder, +cylinders or +size{K,M,G} (2-15193, default
15193): (Press Enter for default.)
Using default value 15193
```

5. Make a partition active, or bootable:

```
Command (m for help): a
Partition number (1-4): 1   (Select Partition 1 to be active.)
```

6. Change the partition type to FAT32. (The value **c** is hexadecimal (decimal 12), the type for a FAT32 partition.):

```
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32 (LBA))
```

7. Write the new partition information:

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

8. Format the new partition with a DOS FAT32 filesystem. Note that now we specify the partition number (`/dev/sdd1`):

```
/home/user> sudo mkfs.vfat -F32 /dev/sdd1
mkfs.vfat 3.0.13 (30 Jun 2012)
/home/user>
```

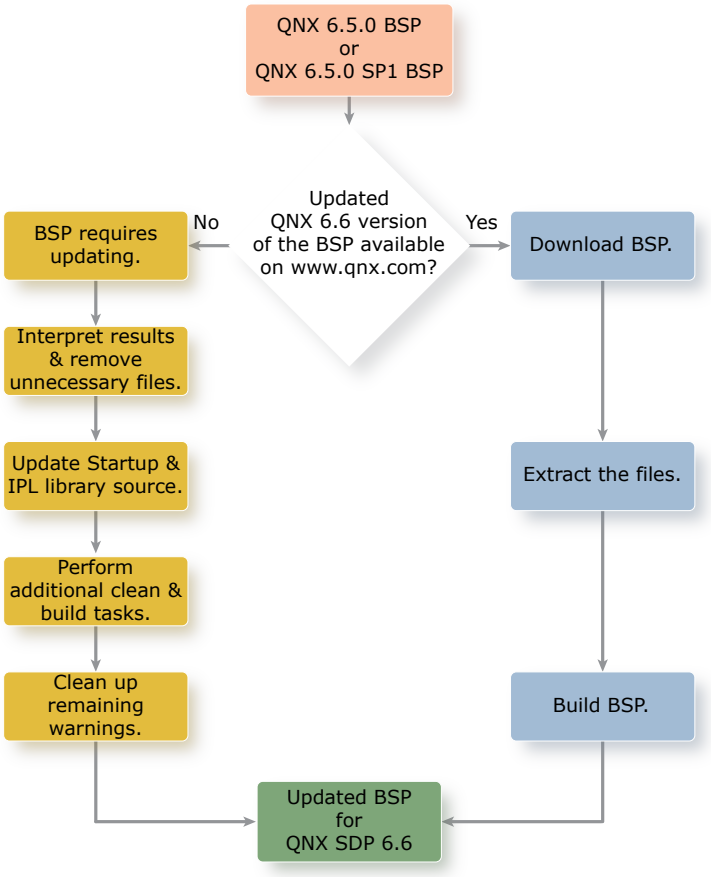
Your SD card is now ready to use with the BSP boot images.

Chapter 4
Process to update a BSP to QNX SDP 6.6

To best illustrate the types of issues you might encounter when you update a QNX 6.5.0 or QNX 6.5.0 SP1 BSP to QNX SDP 6.6, we'll go through an exercise of taking an existing QNX 6.5.0 SP1 BSP and working through the various build issues. We'll also look at code examples taken from the same BSP, which has already been updated by QNX Software Systems to build and run properly under QNX SDP 6.6.

For this example, we'll use the Freescale i.MX6Q SabreLite BSP, which has versions available for both QNX 6.5.0 SP1 and QNX SDP 6.6.

The following illustration shows the process you'd use to determine how to update BSPs.



Prerequisites

These update instructions assume that you're familiar with the structure and layout of QNX BSPs in general and that you have:

- installed QNX SDP 6.6, in either a Windows or Linux environment
- verified that you are able to build and run existing QNX SDP 6.6 BSPs without any issues
- root privilege on your host system, to avoid any issues with permissions when attempting to edit, copy, move, or delete files

Download, extract, and build an existing BSP

To download, extract, and build an existing BSP that you want to update:

1. Download the QNX 6.5.0 or 6.5.0 SP1 BSP that you want to update to QNX SDP 6.6, and extract it to your desired directory within the QNX SDP 6.6 host environment.

   For this example, we'll use the following directory structure:

   `/root/bsps/sabrelite_650/`

   After downloading the BSP, you should have something similar to the following:

   `/root/bsps/sabrelite_650/BSP_freescale-imx6q-sabrelite_br-650_be-650sp1_SVN`*xxxxxx*`_JBN`*yy*`.zip`

   where *xxxxxx* is the SVN ID for the BSP and *yy* is a unique ID.

2. Extract the BSP archive using the following command at the prompt:

   ```
   # unzip BSP_freescale-imx6q-sabrelite_br-650_be-650sp1_SVNxxxxxx_JBNyy.zip
   ```

   For this example, we'll also have the QNX SDP 6.6 version of the same BSP, extracted to the following location:

   `/root/bsps/sabrelite_660/`

3. To build the QNX 6.5.0 BSP, type the following command at the prompt:

   ```
   /root/bsps/sabrelite_650/> # make
   ```

At this point, the BSP installation process copies various header files and prebuilt binaries and libraries from the `/prebuilt` directory for the BSP, to the equivalent location within the `/install` directory for the BSP. The build process will then attempt to recurse through the various source code directories in the BSP, compiling the code along the way. Typically, this step won't progress too far before the build process halts due to the initial errors it encounters. The output below shows the errors from the first attempt to compile the QNX 6.5.0 SP1 Sabrelite BSP in an QNX 6.6 environment:

```
In file included from
/root/bsps/sabrelite_650/src/lib/dma/../../../install/usr/include/sys/platform.h:265:0,
from /root/qnx660/target/qnx6/usr/include/stdio.h:30,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:26,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/sys/target_nto.h:93:2: error: #error __OFF_BITS__ value
                        is unsupported
In file included from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:26:0,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/stdio.h:219:37: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/stdio.h:220:1: error: unknown type name 'off_t'
In file included from /root/qnx660/target/qnx6/usr/include/strings.h:33:0,
from /root/qnx660/target/qnx6/usr/include/string.h:36,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:29,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/string.h:153:1: error: unknown type name 'errno_t'
/root/qnx660/target/qnx6/usr/include/string.h:153:46: error: unknown type name 'rsize_t'
/root/qnx660/target/qnx6/usr/include/string.h:153:73: error: unknown type name 'rsize_t'
In file included from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:33:0,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/sys/mman.h:184:88: error: unknown type name 'off_t'
```

```
/root/qnx660/target/qnx6/usr/include/sys/mman.h:253:71: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/sys/mman.h:277:75: error: unknown type name 'off_t'
In file included from /root/qnx660/target/qnx6/usr/include/fcntl.h:36:0,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:39,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/unistd.h:419:1: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:419:34: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:423:80: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:424:87: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:444:37: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:531:44: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:538:41: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:565:1: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:568:1: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/unistd.h:568:35: error: unknown type name 'off_t'
In file included from /root/qnx660/target/qnx6/usr/include/fcntl.h:40:0,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:39,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/sys/stat.h:152:3: error: #error __OFF_BITS__ value
                          is unsupported
/root/qnx660/target/qnx6/usr/include/sys/stat.h:179:3: error: #error __OFF_BITS__ value
                          is unsupported
In file included from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:39:0,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/fcntl.h:153:3: error: #error __OFF_BITS__ value
                          is unsupported
In file included from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:39:0,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/fcntl.h:243:3: error: #error __OFF_BITS__ value
                          is unsupported
In file included from /root/bsps/sabrelite_650/src/lib/dma/sdma/sdma.h:39:0,
from /root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:23:
/root/qnx660/target/qnx6/usr/include/fcntl.h:316:36: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/fcntl.h:316:52: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/fcntl.h:317:38: error: unknown type name 'off_t'
/root/qnx660/target/qnx6/usr/include/fcntl.h:317:54: error: unknown type name 'off_t'
/root/bsps/sabrelite_650/src/lib/dma/sdma/api.c: In function 'ccb_create':
/root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:89:9: warning: implicit declaration
                          of function 'mmap' [-Wimplicit-function-declaration]
/root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:89:17: warning: assignment makes
                          pointer from integer without a cast [enabled by default]
/root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:98:9: warning: implicit declaration
                          of function 'mem_offset64' [-Wimplicit-function-declaration]
/root/bsps/sabrelite_650/src/lib/dma/sdma/api.c: In function 'chan_create':
/root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:147:22: warning: assignment makes
                          pointer from integer without a cast [enabled by default]
/root/bsps/sabrelite_650/src/lib/dma/sdma/api.c:174:23: warning: assignment makes
                          pointer from integer without a cast [enabled by default]
cc: /root/qnx660/host/linux/x86/usr/lib/gcc/arm-unknown-nto-qnx8.0.0eabi/4.7.3/cc1 error 1
make[7]: *** [api.o] Error 1
make[6]: *** [install] Error 2
make[5]: *** [install] Error 2
make[4]: *** [install] Error 2
make[3]: *** [install] Error 2
make[2]: *** [install] Error 2
make[1]: *** [install] Error 2
make: *** [install] Error 2
```

The first error we see is a conflict between the `constant __OFF_BITS__`, as defined in:

`/root/qnx660/target/qnx6/usr/include/sys/target_nto.h`

and references to this constant within the BSP source code here:

`/src/lib/dma/../../../install/usr/include/sys/platform.h`

For detailed information about interpreting these errors, see *Interpret initial results and remove unnecessary files* (p. 25)

Interpret initial results and remove unnecessary files

Typically, files that appear under ***$QNX_TARGET***`/usr/include/sys` within the host development environment would not be duplicated within the BSP. However, over the lifespan of QNX Momentics 6.5.0, it was necessary to make changes or updates to these and other files that were found under the `/usr/include/sys/` directory. To maintain compatibility between source code that we included with various BSPs, and the updated versions of these header files, QNX typically includes the updated header file(s) with any BSP that requires them, in order to build the corresponding source code. These header files override the ones included in the QNX 6.5.0 SP1 installation.

However, with the advent of a new version of the operating system (i.e., QNX SDP 6.6), the latest versions of these header files are now present within the host environment itself, and so we no longer need to include those same files within the BSP archive. In fact, the older BSP versions of these files will now themselves cause conflicts, and need to be removed so that the base QNX SDP 6.6 versions are the only ones present in the system. The same is true for all files found under the BSP directory `prebuilt/usr/include/sys`.

To process and continue cleaning the BSP:

1. To clean up the first batch of errors, you'll need to type the following commands at the prompt:

   ```
   # cd /root/bsps/sabrelite_650/prebuilt/usr/include/sys/
   # ls
   asound.h  can_dcmd.h  platform.h   srcversion.h
   #
   ```

2. Look in the equivalent directory of the QNX SDP 6.6 host environment (under `/root/qnx660/target/qnx6/usr/include/sys`).
   You'll see that new versions of all of these files (and many others) are included with QNX SDP 6.6. Therefore, they can be safely deleted from the BSP.
3. To remove these files, navigate to the directory and remove these files by typing the following commands at the prompt:

   ```
   # cd /root/bsps/sabrelite_650/prebuilt/usr/include/sys/
   # rm *
   ```

Rebuild the BSP

Now, let's try building the same BSP once again by typing the following commands:

You'll likely repeat this next step several times throughout the process of updating the BSP.

```
# cd /root/bsps/sabrelite_650/
# make clean  (this command will remove all of the contents of the /install directory)
# make
```

As you can see from the results below, many of the initial errors are now fixed and the build is much further along in the process. The following output shows the console output where the next error in the process reveals an issue with a missing library:

```
/root/qnx660/host/linux/x86/usr/bin/qcc -Vgcc_ntoarmv7 -c -Os -Wc,-Wall -DNDEBUG   -I.
-I/root/bsps/sabrelite_650/src/hardware/devc/sermx1/arm/le.v7
-I/root/bsps/sabrelite_650/src/hardware/devc/sermx1/arm
-I/root/bsps/sabrelite_650/src/hardware/devc/sermx1
-I/root/bsps/sabrelite_650/src/hardware/devc
-I/root/bsps/sabrelite_650/src/hardware/devc/../../../install/usr/include/xilinx
-I/root/bsps/sabrelite_650/src/hardware/devc/../../../install/usr/include
-I/root/qnx660/target/qnx6/usr/include    -EL
      -DVARIANT_le -DVARIANT_v7 -DVARIANT_sermx1 -DBUILDENV_qss
      /root/bsps/sabrelite_650/src/hardware/devc/sermx1/tto.c
            make[6]: *** No rule to make target `libpm.a', needed by
                         `/root/bsps/sabrelite_650/src/hardware/devc/
                         sermx1/arm/le.v7/devc-sermx1'.  Stop.
            make[6]: Leaving directory `/root/bsps/sabrelite_650/src/hardware/devc/sermx1/arm/le.v7'
            make[5]: *** [install] Error 2
            make[5]: Leaving directory `/root/bsps/sabrelite_650/src/hardware/devc/sermx1/arm'
            make[4]: *** [install] Error 2
            make[4]: Leaving directory `/root/bsps/sabrelite_650/src/hardware/devc/sermx1'
            make[3]: *** [install] Error 2
            make[3]: Leaving directory `/root/bsps/sabrelite_650/src/hardware/devc'
            make[2]: *** [install] Error 2
            make[2]: Leaving directory `/root/bsps/sabrelite_650/src/hardware'
            make[1]: *** [install] Error 2
            make[1]: Leaving directory `/root/bsps/sabrelite_650/src'
            make: *** [install] Error 2
```

Missing library

The issue here is a missing library (`libpm.a`), which previously shipped with QNX Momentics 6.5.0, but no longer ships with QNX SDP 6.6. So, you'll need to modify the serial driver source code that attempts to use this library. Comparing the `/devc` source directory contained within the QNX 6.5 version of the BSP to the equivalent directory contained within the (already ported) QNX 6.6 version, we see the first diff:

```
diff -r -b /root/bsps/sabrelite_660/src/hardware/devc /root/bsps/sabrelite_650/src/hardware/devc
< LIBS+=io-char drvr
---
> LIBS+=io-char pm ps drvr
```

So, in the QNX 6.5.0 version of the BSP, the `common.mk` file for the serial driver source code is including this `libpm.a`. Notice that from the QNX SDP 6.6 version of the BSP, this library (and a corresponding `libps.a`) is no longer needed. Therefore,

you'll need to edit the file `common.mk` to change the line with `LIBS` as shown above, to remove the references to `pm` and `ps`:

```
# LIBS+=io-char pm ps drvr
LIBS+=io-char drvr
```

Again, looking at the BSP root directory (`/root/bsps/sabrelite_650`), and performing a `make clean`; `make` operation, the BSP build is able to advance much further along in the build process.

Touch/input driver

The next issue encountered with the build process for the BSP occurs with the attempt to compile the touchscreen driver contained within the QNX 6.5.0 version of the BSP. The issue here is that the touch driver in the QNX QNX 6.5.0 BSP is based on the older `devi` input framework, which has not been carried forward to QNX SDP 6.6. Describing what needs to be done to migrate an input/touch driver to the new `mtouch` framework is beyond the scope of this document, but there are several QNX SDP 6.6 BSPs available that include `mtouch` touchscreen drivers, so these can be used as a reference for updating your own touch/input driver, if necessary. In this particular case, we will simply remove the `/devi` directory altogether:

```
#cd /root/bsps/sabrelite_650/src/hardware
# rm –Rf devi
```

Returning to the root of the BSP directory, and doing a `make clean`; `make` operation, we see that the next issue with our example BSP is that the network driver source that's included with the BSP has many issues regarding re-declaration of enumerators, conflicting types, and so on:

```
I../../../../../io-pkt/lib/socket/public -I/root/bsps/sabrelite_650/src/hardware/devnp/mx6x/../../../../
    install/usr/include
-I/root/qnx660/target/qnx6/usr/include       -EL      -shared
    -DVARIANT_dll -DVARIANT_le -DVARIANT_v7 -DBUILDENV_qss
    /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c
            In file included from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/mx6q.h:48:0,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c:22:
            /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/../../../../
                install/usr/include/io-pkt/drvr/mdi.h:290:16:
                error: redeclaration of enumerator 'WaitBusy'
            In file included from /root/qnx660/target/qnx6/usr/include/io-pkt/iopkt_driver.h:59:0,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/mx6q.h:27,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c:22:
            /root/qnx660/target/qnx6/usr/include/netdrvr/mdi.h:290:16: note: previous definition
                 of 'WaitBusy' was here
            In file included from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/mx6q.h:48:0,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c:22:
            /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/../../../../install/usr/include/
                 io-pkt/drvr/mdi.h:290:30:
                 error: redeclaration of enumerator 'NoWait'
            In file included from /root/qnx660/target/qnx6/usr/include/io-pkt/iopkt_driver.h:59:0,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/mx6q.h:27,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c:22:
            /root/qnx660/target/qnx6/usr/include/netdrvr/mdi.h:290:30: note: previous definition of
                 'NoWait' was here
            In file included from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/mx6q.h:48:0,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c:22:
            /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/../../../../install/usr/include/io-pkt/
                 drvr/mdi.h:290:38:
                 error: redeclaration of enumerator 'IrqNoWait'
            In file included from /root/qnx660/target/qnx6/usr/include/io-pkt/iopkt_driver.h:59:0,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/mx6q.h:27,
            from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c:22:
            /root/qnx660/target/qnx6/usr/include/netdrvr/mdi.h:290:38: note: previous definition of
```

```
     'IrqNoWait' was here
In file included from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/mx6q.h:48:0,
from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c:22:
/root/bsps/sabrelite_650/src/hardware/devnp/mx6x/../../../../install/usr/include/io-pkt/
     drvr/mdi.h:290:50:
     error: conflicting types for 'MDIWaitType'
In file included from /root/qnx660/target/qnx6/usr/include/io-pkt/iopkt_driver.h:59:0,
from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/mx6q.h:27,
from /root/bsps/sabrelite_650/src/hardware/devnp/mx6x/bsd_media.c:22:
/root/qnx660/target/qnx6/usr/include/netdrvr/mdi.h:290:50: note: previous declaration of
     'MDIWaitType' was here


...
...
...
```

Network updates

The reason for these errors is that since QNX 6.5.0 was released, there have been significant updates to the network driver library routines. In order for the current driver source to compile correctly, various (updated) network driver-related header files needed to be included in the BSP. One of the changes was to separate out the network-specific library functions from the generic driver functions, so that network-related functions that were previously found in `libdrvr.a` are now found in `libnetdrvr.a`. With the advent of QNX SDP 6.6, additional updates were made to these routines, and the latest version of `libnetdrvr.a` (as well as its corresponding header files) is included with QNX SDP 6.6. As a result, to get the network driver source building, you'll need to do the following steps:

1. Delete all `io-pkt` related headers, and the prebuilt `libdrvrS.a`, from the BSP:

   ```
   # cd root/bsp/sabrelite_650/prebuilt/usr/include/
   # rm –Rf io-pkt
   # cd root/bsp_sabrelite_650/prebuilt/armle-v7/usr/lib
   # rm libdrvrS.a
   ```

2. Modify the `common.mk` file for the mx6x network driver to include `libnetdrvr`:

   ```
   # cd root/bsp/sabrelite_650/src/hardware/devnp/mx6x/
   # vim common.mk (edit as shown below)

   before:
           LIBS = drvrS cacheS
   after:
           LIBS = netdrvrS drvrS cacheS
   ```

3. Edit the `mx6q.h` file, which includes several header files previously associated with `libdrvr`, but are now instead associated with `libnetdrvr`:

   ```
   Before:
                   #include <drvr/mdi.h>
                   #include <drvr/eth.h>
                   #include <drvr/nicsupport.h>
                   #include <drvr/common.h>

   After:
                   #include <netdrvr/mdi.h>
                   #include <netdrvr/eth.h>
                   #include <netdrvr/nicsupport.h>
                   #include <netdrvr/common.h>
   ```

After completing these steps, return to the BSP root, and do the `make clean`; `make` again; at this point, we see that the network driver builds without errors. We also see that the BSP now builds to completion (although there are still warnings, which we'll discuss later).

Update the startup and IPL library source

Every QNX BSP includes a current snapshot of the QNX startup library code, and for those BSPs that also include an Initial Program Loader (IPL), we also include the source to the IPL library. Since we're updating a QNX 6.5.0 BSP to QNX SDP 6.6 and because the QNX SDP 6.6 version of the startup library (and IPL library) code includes various QNX SDP 6.6 BSPs, it's a simple operation to do a replacement of the QNX 6.5.0 library source with the equivalent QNX SDP 6.6 version.

The startup library code has undergone a number of significant updates, which correspond to various new features and functionality of the QNX SDP 6.6 kernel, so updating the startup library code will definitely reduce the likelihood of compatibility issues and crashes when you attempt to run your updated QNX 6.5.0 BSP in a QNX SDP 6.6 environment.

To update the startup and IPL library source:

1. Assuming you've already downloaded a current QNX SDP 6.6 BSP for the same CPU architecture as the BSP that you're updating, you can type the following commands:

```
# cd /root/bsps/sabrelite_650/src/hardware/startup/
# rm -Rf lib
# cp -R /root/bsps/sabrelite_660/src/hardware/startup/lib .
# cd ../ipl
# rm -Rf lib
# cp -R /root/bsps/sabrelite_660/src/hardware/ipl/lib
```

2. Return to the BSP root directory and run the following commands again:

```
# cd /root/bsps/sabrelite_650/
# make clean  (this command will remove all of the contents of
 the /install directory)
# make
```

Now, your startup code should be compatible with the QNX SDP 6.6 kernel and environment. In addition, you'll gain the added benefit of replacing library source code that would have otherwise generated many warnings (due to compiling with a newer, more strict version of the GNU/gcc toolchain), with library source code that's already been modified for compilation under the QNX SDP 6.6 toolchain.

In general, the same procedure can (and should) be done for any other source that typically ships with QNX BSPs (for example, DMA library source, standard device drivers, etc.). It may save you significant time and effort to take a look through the various QNX SDP 6.6 BSP offerings, and see what source code is available that might be compatible with devices or hardware that you're attempting to port from an existing QNX 6.5.0 BSP. As an example; earlier, we compared the serial driver source code from the original QNX 6.5.0 BSP to the same source code in the current QNX SDP 6.6 BSP, to see what changes were necessary to get the 6.5.0 version to build.

Although this was useful for illustration purposes, a better solution would be to replace the original QNX 6.5.0 version of the serial driver with the QNX SDP 6.6 version because the newer version is available.

Update BSPs containing prebuilt kernels and other OS components

Typically, QNX BSPs don't include binaries for common OS components such as the QNX kernel (`procnto-*`), C library (`libc.so`), or any other components that are included as part of a QNX Software Development Platform. However, there have been occasions where important kernel or `libc` updates (or updates to other critical components) were necessary for certain BSPs to operate correctly. Rather than make a separate patch available, we have included the updated components directly in the BSP archive to ensure that the proper version is used with the BSP. While it's unlikely that the BSP you're updating will contain any prebuilt OS components such as these, it's always good practice to **verify and remove them if they exist**. Typically, if they're included, these types of components would be located in the `/prebuilt` directory for the BSPs in the following location(s):

```
prebuilt/armle-v7/boot/sys
prebuilt/armle-v7/lib
```

Search these directories for any binaries named `procnto`, `procnto-smp`, and so on, or for libraries such as `libc.so`. If they're present, delete them before building your BSP. Similarly, if you see other prebuilt components in the directory `prebuilt/armle-v7`, and those same components also exist in the directory `/root/qnx660/target/qnx6/armle-v7/`, it's safe to assume that you want to use the ones found in `qnx660`, and delete the older versions from the BSP.

Chapter 5
Clean Up Remaining Warnings

Now our example BSP builds to completion and you'll have already replaced as much of the QNX 6.5.0 BSP code as possible with the equivalent updated versions found in the existing QNX SDP 6.6 BSPs. At this point, you should be able to generate an IFS image from your BSP and test it on your target hardware. However, before doing that, you may notice that there are several warnings generated when you compile the BSP source this final time. The following output shows the list of warnings generated when you compile the QNX 6.5.0 SabreLite BSP example in its current state:

```
/root/bsps/sabrelite_650/src/hardware/deva/ctrl/mx/mxssi_dll.c: In function 'mx_capture_trigger':
/root/bsps/sabrelite_650/src/hardware/deva/ctrl/mx/mxssi_dll.c:414:13: warning: variable 'tmp' set but not used [-Wunused-but-set-variable]
/root/bsps/sabrelite_650/src/hardware/deva/ctrl/mx/mxssi_dll.c: In function 'mx_capture_trigger2':
/root/bsps/sabrelite_650/src/hardware/deva/ctrl/mx/mxssi_dll.c:714:13: warning: variable 'tmp' set but not used [-Wunused-but-set-variable]
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/mmcsd.c: In function 'mmc_dump':
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/mmcsd.c:83:7: warning: variable 'skipping' set but not used [-Wunused-but-set-variable]
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/mmcsd.c: In function 'mmc_erase':
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/mmcsd.c:187:16: warning: variable 'ext' set but not used [-Wunused-but-set-variable]
sim_bs.c: In function 'mx6dq_detect':
sim_bs.c:36:14: warning: variable 'mx35' set but not used [-Wunused-but-set-variable]
sim_bs.c: In function 'bs_dinit':
sim_bs.c:112:14: warning: variable 'mx35' set but not used [-Wunused-but-set-variable]
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/sim_mmc.c: In function 'mmc_evpd_inquiry':
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/sim_mmc.c:278:17: warning: variable 'is' set but not used [-Wunused-but-set-variable]
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/sim_mmc.c: In function 'mmc_fake_inquiry':
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/sim_mmc.c:407:16: warning: variable 'ext' set but not used [-Wunused-but-set-variable]
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/sim_mmc.c: In function 'mmc_pio_initiate':
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/sim_mmc.c:941:11: warning: variable 'blkno' set but not used [-Wunused-but-set-variable]
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/chipsets/sim_mx35.c: In function 'mx35_command':
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/chipsets/sim_mx35.c:257:18: warning: variable 'intrstien' set but not used
     [-Wunused-but-set-variable]
/root/bsps/sabrelite_650/src/hardware/devb/mmcsd/chipsets/sim_mx35.c:256:18: warning: variable 'intrsten' set but not used
     [-Wunused-but-set-variable]
../../fujitsu/f29f100_ident.c: In function 'f3s_f29f100_ident':
../../fujitsu/f29f100_ident.c:52:34: warning: variable 'amd_mult' set but not used [-Wunused-but-set-variable]
../../intel/i28f008_suspend.c: In function 'f3s_i28f008_suspend':
../../intel/i28f008_suspend.c:44:11: warning: variable 'status' set but not used [-Wunused-but-set-variable]
/root/qnx660/host/linux/x86/usr/bin/ntoarmv7-ar: creating /root/bsps/sabrelite_650/src/hardware/flash/mtd-flash/arm/a.le.v7/libmtd-flash.a
/root/qnx660/host/linux/x86/usr/bin/ntoarmv7-ar: creating /root/bsps/sabrelite_650/src/hardware/ipl/lib/arm/a.le.v7/libipl.a
/root/bsps/sabrelite_650/src/hardware/ipl/boards/mx6q-sabrelite/fat-fs.c:39:1: warning: conflicting types for built-in function 'memcpy'
     [enabled by default]
/root/bsps/sabrelite_650/src/hardware/ipl/boards/mx6q-sabrelite/fat-fs.c:339:1: warning: 'read_cluster' defined but not used
     [-Wunused-function]
/root/bsps/sabrelite_650/src/hardware/spi/mx51ecspi/mxecspi.c: In function 'mx51_options':
/root/bsps/sabrelite_650/src/hardware/spi/mx51ecspi/mxecspi.c:204:20: warning: variable 'err' set but not used [-Wunused-but-set-variable]
/root/qnx660/host/linux/x86/usr/bin/ntoarmv7-ar: creating /root/bsps/sabrelite_650/src/hardware/startup/lib/arm/a.le.v7/libstartup.a
/root/bsps/sabrelite_650/src/utils/r/rtc/qnxrtc.c: In function 'main':
/root/bsps/sabrelite_650/src/utils/r/rtc/qnxrtc.c:487:6: warning: format '%ld' expects argument of type 'long int', but argument 2
     has type '_Int32t' [-Wformat]
/root/bsps/sabrelite_650/src/utils/r/rtc/qnxrtc.c:487:6: warning: format '%ld' expects argument of type 'long int', but argument 3
     has type '_Uint32t' [-Wformat]
/root/bsps/sabrelite_650/src/utils/r/rtc/qnxrtc.c:496:7: warning: format '%ld' expects argument of type 'long int', but argument 2
     has type '_Uint32t' [-Wformat]
```

As indicated earlier, warnings that didn't appear when the same BSP was compiled in a QNX 6.5.0 environment but appear when the same code is compiled in a QNX SDP 6.6.0 environment are due to the updated GNU toolchain (QNX 6.5.0 and QNX 6.5.1 SP1 uses version 4.4.2, which was updated to version 4.7.3 in QNX SDP 6.6). Typically, the gcc compiler becomes more strict about coding practices with each updated version. Many of the warnings seen in the output above relate to variables or functions that are declared, but not subsequently used. Other warnings relate to typecasting and to general coding practices.

Since the GNU toolchain is widely used in computing environments outside of QNX, considerable information exists on the internet that describes in detail

how to fix the warnings that appear when you compile your updated BSP, so we won't go into that level of detail here.

Index

B

C

D

E

F

G

I

K

L

M

P

S

T

U