Board Support Package (BSP) Developer's Guide

Richard Purdie, Intel Corporation
<richard.purdie@linuxfoundation.org>

by Richard Purdie

Table of Contents

Chapter 1. Board Support Packages (BSP) - Developers Guide

A Board Support Package (BSP) is a collection of information that defines how to support a particular hardware device, set of devices, or hardware platform. The BSP includes information about the hardware features present on the device and kernel configuration information along with any additional hardware drivers required. The BSP also lists any additional software components required in addition to a generic Linux software stack for both essential and optional platform features.

This section (or document if you are reading the BSP Developer's Guide) defines a structure for these components so that BSPs follow a commonly understood layout. Providing a common form allows end-users to understand and become familiar with the layout. A common form also encourages standardization of software support of hardware.

The proposed format does have elements that are specific to the Poky and OpenEmbedded build systems. It is intended that this information can be used by other systems besides Poky and OpenEmbedded and that it will be simple to extract information and convert it to other formats if required. Poky, through its standard layers mechanism, can directly accept the format described as a layer. The BSP captures all the hardware-specific details in one place in a standard format, which is useful for any person wishing to use the hardware platform regardless of the build system they are using.

The BSP specification does not include a build system or other tools - it is concerned with the hardware-specific components only. At the end distribution point you can ship the BSP combined with a build system and other tools. However, it is important to maintain the distinction that these are separate components that happen to be combined in certain end products.

1.1. Example Filesystem Layout

The BSP consists of a file structure inside a base directory, which uses the following naming convention:

```
meta-<bsp_name>
```

"bsp_name" is a placeholder for the machine or platform name. Here are some example base directory names:

```
meta-emenlow
meta-intel_n450
meta-beagleboard
```

Below is the common form for the file structure inside a base directory. While you can use this basic form for the standard, realize that the actual structures for specific BSPs could differ.

```
meta-<bsp_name>/
meta-<bsp_name>/<bsp_license_file>
meta-<bsp_name>/README
meta-<bsp_name>/binary/<bootable_images>
meta-<bsp_name>/conf/layer.conf
meta-<bsp_name>/conf/machine/*.conf
meta-<bsp_name>/recipes-bsp/*
meta-<bsp_name>/recipes-graphics/*
meta-<bsp_name>/recipes-kernel/linux/linux-yocto_git.bbappend
```

Below is an example of the crownbay BSP:

```
meta-crownbay/COPYING.MIT
meta-crownbay/README
meta-crownbay/binary/.gitignore
meta-crownbay/conf/layer.conf
meta-crownbay/conf/machine/crownbay.conf
meta-crownbay/recipes-bsp/formfactor/formfactor/crownbay/machconfig
meta-crownbay/recipes-bsp/formfactor/formfactor_0.0.bbappend
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-config/crownbay/xcorg.conf
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-config_0.1.bbappend
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd-bin/.gitignore
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd-bin_1.7.99.2.bb
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd/crosscompile.patch
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd/fix_open_max_preprocessor_error.pat
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd/macro_tweak.patch
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd/nodolt.patch
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd_1.7.99.2.bb
meta-crownbay/recipes-kernel/linux/linux-yocto_git.bbappend
```

The following sections describe each part of the proposed BSP format.

1.1.1. License Files

```
meta-<bsp_name>/<bsp_license_file>
```

These optional files satisfy licensing requirements for the BSP. The type or types of files here can vary depending on the licensing requirements. For example, in the crownbay BSP all licensing requirements are handled with the `COPYING.MIT` file.

Licensing files can be MIT, BSD, GPLv*, and so forth. These files are recommended for the BSP but are optional and totally up to the BSP developer.

1.1.2. README File

```
meta-<bsp_name>/README
```

This file provides information on how to boot the live images that are optionally included in the `/binary` directory. The `README` file also provides special information needed for building the image.

Technically speaking a `README` is optional but it is highly recommended that every BSP has one.

1.1.3. Pre-built User Binaries

```
meta-<bsp_name>/binary/<bootable_images>
```

This optional area contains useful pre-built kernels and user-space filesystem images appropriate to the target system. This directory contains the Application Development Toolkit (ADT) and minimal live images when the BSP is has been "tar-balled" and placed on the Yocto Project website. You can use these kernels and images to get a system running and quickly get started on development tasks.

The exact types of binaries present are highly hardware-dependent. However, a README file should be present in the BSP file structure that explains how to use the kernels and images with the target hardware. If pre-built binaries are present, source code to meet licensing requirements must also be provided in some form.

1.1.4. Layer Configuration File

`meta-<bsp_name>/conf/layer.conf`

This file identifies the structure as a Poky layer, identifies the contents of the layer, and contains information about how Poky should use it. Generally, a standard boilerplate file such as the following works. In the following example you would replace "bsp" and "_bsp" with the actual name of the BSP (i.e. <bsp_name> from the example template).

```
# We have a conf directory, add to BBPATH
BBPATH := "${BBPATH}:${LAYERDIR}"

# We have a recipes directory containing .bb and .bbappend files, add to BBFILES
BBFILES := "${BBFILES} ${LAYERDIR}/recipes/*/*.bb \ ${LAYERDIR}/recipes/*/*.bbappend"

BBFILE_COLLECTIONS += "bsp"
BBFILE_PATTERN_bsp := "^${LAYERDIR}/"
BBFILE_PRIORITY_bsp = "5"
```

This file simply makes BitBake aware of the recipes and configuration directories. This file must exist so that Poky can recognize the BSP.

1.1.5. Hardware Configuration Options

`meta-<bsp_name>/conf/machine/*.conf`

The machine files bind together all the information contained elsewhere in the BSP into a format that Poky can understand. If the BSP supports multiple machines, multiple machine configuration files can be present. These filenames correspond to the values to which users have set the MACHINE variable.

These files define things such as the kernel package to use (PREFERRED_PROVIDER of virtual/kernel), the hardware drivers to include in different types of images, any special software components that are needed, any bootloader information, and also any special image format requirements.

At least one machine file is required for a BSP layer. However, you can supply more than one file.

This directory could also contain shared hardware "tuning" definitions that are commonly used to pass specific optimization flags to the compiler. An example is `tune-atom.inc`:

```
BASE_PACKAGE_ARCH = "core2"
TARGET_CC_ARCH = "-m32 -march=core2 -msse3 -mtune=generic -mfpmath=sse"
```

This example defines a new package architecture called "core2" and uses the specified optimization flags, which are carefully chosen to give best performance on atom processors.

The tune file would be included by the machine definition and can be contained in the BSP or referenced from one of the standard core set of files included with Poky itself.

Both the base package architecture file and the tune file are optional for a Poky BSP layer.

1.1.6. Miscellaneous Recipe Files

`meta-<bsp_name>/recipes-bsp/*`

This optional directory contains miscellaneous recipe files for the BSP. Most notably would be the formfactor files. For example, in the crownbay BSP there is a `machconfig` file and a `formfactor_0.0.bbappend` file:

```
meta-crownbay/recipes-bsp/formfactor/formfactor/crownbay/machconfig
meta-crownbay/recipes-bsp/formfactor/formfactor_0.0.bbappend
```

Note

If a BSP does not have a formfactor entry, defaults are established according to the configuration script.

1.1.7. Display Support Files

```
meta-<bsp_name>/recipes-graphics/*
```

This optional directory contains recipes for the BSP if it has special requirements for graphics support. All files that are needed for the BSP to support a display are kept here. For example, in the crownbay BSP several display support files exist:

```
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-config/crownbay/xcorg.conf
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-config_0.1.bbappend
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd-bin/.gitignore
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd-bin_1.7.99.2.bb
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd/crosscompile.patch
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd/fix_open_max_preprocessor_error.pat
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd/macro_tweak.patch
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd/nodolt.patch
meta-crownbay/recipes-graphics/xorg-xserver/xserver-xf86-emgd_1.7.99.2.bb
```

1.1.8. Linux Kernel Configuration

```
meta-<bsp_name>/recipes-kernel/linux/linux-yocto_git.bbappend
```

This file appends your specific changes to the kernel you are using.

For your BSP you typically want to use an existing Poky kernel found in the Poky repository at `meta/recipes-kernel/kernel`. You can append your specific changes to the kernel recipe by using an append file, which is located in the `meta-<bsp_name>/recipes-kernel/linux` directory.

Suppose you use a BSP that uses the `linux-yocto_git.bb` kernel, which is the preferred kernel to use for developing a new BSP using the Yocto Project. In other words, you have selected the kernel in your `<bsp_name>.conf` file by adding the following statement:

```
PREFERRED_PROVIDER_virtual/kernel ?= "linux-yocto"
```

You would use the `linux-yocto_git.bbappend` file to append specific BSP settings to the kernel, thus configuring the kernel for your particular BSP.

Now take a look at the existing "crownbay" BSP. The append file used is:

```
meta-crownbay/recipes-kernel/linux/linux-yocto_git.bbappend
```

The file contains the following:

```
FILESEXTRAPATHS := "${THISDIR}/${PN}"
COMPATIBLE_MACHINE_crownbay = "crownbay"
```

```
KMACHINE_crownbay = "yocto/standard/crownbay"
```

This append file adds "crownbay" as a compatible machine, and additionally sets a Yocto Kernel-specific variable that identifies the name of the BSP branch to use in the GIT repository to find configuration information.

One thing missing in this particular BSP, which you will typically need when developing a BSP, is the kernel configuration (.config) for your BSP. When developing a BSP, you probably have a kernel configuration file or a set of kernel configuration files that, when taken together, define the kernel configuration for your BSP. You can accomplish this definition by putting the configurations in a file or a set of files inside a directory located at the same level as your append file and having the same name as the kernel. With all these conditions met simply reference those files in a SRC_URI statement in the append file.

For example, suppose you had a set of configuration options in a file called `defconfig`. If you put that file inside a directory named `/linux-yocto` and then added a SRC_URI statement such as the following to the append file, those configuration options will be picked up and applied when the kernel is built.

```
SRC_URI += "file://defconfig"
```

As mentioned earlier, you can group related configurations into multiple files and name them all in the SRC_URI statement as well. For example, you could group separate configurations specifically for Ethernet and graphics into their own files and add those by using a SRC_URI statement like the following in your append file:

```
SRC_URI += "file://defconfig \
            file://eth.cfg \
            file://gfx.cfg"
```

The FILESEXTRAPATHS variable is in boilerplate form here in order to make it easy to do that. It basically allows those configuration files to be found by the build process.

Note

Other methods exist to accomplish grouping and defining configuration options. For example, you could directly add configuration options to the Yocto kernel `meta` branch for your BSP. The configuration options will likely end up in that location anyway if the BSP gets added to the Yocto Project. For information on how to add these configurations directly, see the "Yocto Project Kernel Architecture and Use Manual" on the Yocto Project website Documentation Page [http://yoctoproject.org/community/documentation]

In general, however, the Yocto Project maintainers take care of moving the SRC_URI-specified configuration options to the `meta` branch. Not only is it easier for BSP developers to not have to worry about putting those configurations in the branch, but having the maintainers do it allows them to apply 'global' knowledge about the kinds of common configuration options multiple BSPs in the tree are typically using. This allows for promotion of common configurations into common features.

1.2. BSP 'Click-Through' Licensing Procedure

Note

This section describes how click-through licensing is expected to work. Currently, this functionality is not yet implemented.

In some cases, a BSP contains separately licensed IP (Intellectual Property) for a component that imposes upon the user a requirement to accept the terms of a 'click-through' license. Once the license is accepted the Poky build system can then build and include the corresponding component in the final BSP image. Some affected components might be essential to the normal functioning of the system

and have no 'free' replacement (i.e. the resulting system would be non-functional without them). On the other hand, other components might be simply 'good-to-have' or purely elective, or if essential nonetheless have a 'free' (possibly less-capable) version that could be used as a in the BSP recipe.

For cases where you can substitute something and still maintain functionality, the Yocto Project website at http://yoctoproject.org/download/board-support-package-bsp-downloads will make available a 'de-featured' BSP completely free of the encumbered IP. In that case you can use the substitution directly and without any further licensing requirements. If present, this fully 'de-featured' BSP will be named appropriately different than the normal encumbered BSP. If available, this substitution is the simplest and most preferred option. This, of course, assumes the resulting functionality meets requirements.

If however, a non-encumbered version is unavailable or the 'free' version would provide unsuitable functionality or quality, you can use an encumbered version.

Several methods exist within the Poky build system to satisfy the licensing requirements for an encumbered BSP. The following list describes them in preferential order:

1. Get a license key (or keys) for the encumbered BSP by visiting a website and providing the name of the BSP and your email address through a web form.

   After agreeing to any applicable license terms, the BSP key(s) will be immediately sent to the address you gave and you can use them by specifying BSPKEY_<keydomain> environment variables when building the image:

   ```
   $ BSPKEY_<keydomain>=<key> bitbake poky-image-sato
   ```

   These steps allow the encumbered image to be built with no change at all to the normal build process.

   Equivalently and probably more conveniently, a line for each key can instead be put into the user's `local.conf` file.

   The <keydomain> component of the BSPKEY_<keydomain> is required because there might be multiple licenses in effect for a given BSP. In such cases, a given <keydomain> corresponds to a particular license. In order for an encumbered BSP that encompasses multiple key domains to be built successfully, a <keydomain> entry for each applicable license must be present in `local.conf` or supplied on the command-line.

2. Do nothing - build as you normally would. When a license is needed the build will stop and prompt you with instructions. Follow the license prompts that originate from the encumbered BSP. These prompts usually take the form of instructions needed to manually fetch the encumbered package(s) and md5 sums into the required directory (e.g. the `poky/build/downloads`). Once the manual package fetch has been completed, restart the build to continue where it left off. During the build the prompt will not appear again since you have satisfied the requirement.

3. Get a full-featured BSP recipe rather than a key. You can do this by visiting the applicable BSP download page from the Yocto Project website at http://yoctoproject.org/download/board-support-package-bsp-downloads. BSP tarballs that have proprietary information can be downloaded after agreeing to licensing requirements as part of the download process. Obtaining the code this way allows you to build an encumbered image with no changes at all as compared to the normal build.

Note that the third method is also the only option available when downloading pre-compiled images generated from non-free BSPs. Those images are likewise available at from the Yocto Project website.