
# Board Support Package (BSP) Developers Guide

Richard Purdie, Intel Corporation **<richard@linux.intel.com>**

# Board Support Package (BSP) Developers Guide

by Richard Purdie

# Table of Contents

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

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

The intent of this document is to define a structure for these components so that BSPs follow a commonly understood layout, allowing them to be provided in a common form that everyone understands. It also allows end-users to become familiar with one common format and encourages standardisation 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/OpenEmbedded and that it will be simple to extract information and convert to other formats if required. The format described can be directly accepted as a layer by Poky using its standard layers mechanism, but it is important to recognise that 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 in use.

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 the BSP may be shipped combined with a build system and other tools, but it is important to maintain the distinction that these are separate components which may just be combined in certain end products.

## 1.1. Example Filesystem Layout

The BSP consists of a file structure inside a base directory, meta-bsp in this example, where "bsp" is a placeholder for the machine or platform name. Examples of some files that it could contain are:

```
meta-bsp/
meta-bsp/binary/zImage
meta-bsp/binary/poky-image-minimal.directdisk
meta-bsp/conf/layer.conf
meta-bsp/conf/machine/*.conf
meta-bsp/conf/machine/include/tune-*.inc
meta-bsp/packages/bootloader/bootloader_0.1.bb
meta-bsp/packages/linux/linux-bsp-2.6.50/*.patch
meta-bsp/packages/linux/linux-bsp-2.6.50/defconfig-bsp
meta-bsp/packages/linux/linux-bsp_2.6.50.bb
meta-bsp/packages/modem/modem-driver_0.1.bb
meta-bsp/packages/modem/modem-daemon_0.1.bb
meta-bsp/packages/image-creator/image-creator-native_0.1.bb
meta-bsp/prebuilds/
```

The following sections detail what these files and directories could contain.

## 1.2. Prebuilt User Binaries (meta-bsp/binary/*)

This optional area contains useful prebuilt kernels and userspace filesystem images appropriate to the target system. Users could use these to get a system running and quickly get started on development tasks. The exact types of binaries present will be highly hardware-dependent but a README file should be present explaining how to use them with the target hardware. If prebuilt binaries are present, source code to meet licensing requirements must also be provided in some form.

## 1.3. Layer Configuration (meta-bsp/conf/layer.conf)

This file identifies the structure as a Poky layer. This file identifies the contents of the layer and contains information about how Poky should use it. In general it will most likely be a standard boilerplate file consisting of:

```
# 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"
```

which simply makes bitbake aware of the recipes and conf directories.

This file is required for recognition of the BSP by Poky.

## 1.4. Hardware Configuration Options (meta-bsp/conf/machine/*.conf)

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

These files would define things like which kernel package to use (PREFERRED_PROVIDER of virtual/kernel), which 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 Poky BSP layer but more than one may be present.

## 1.5. Hardware Optimisation Options (meta-bsp/conf/machine/include/tune-*.inc)

These are shared hardware "tuning" definitions and are commonly used to pass specific optimisation 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"
```

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

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

These files are optional for a Poky BSP layer.

## 1.6. Linux Kernel Configuration (meta-bsp/packages/linux/*)

These files make up the definition of a kernel to use with this hardware. In this case it is a complete self-contained kernel with its own configuration and patches but kernels can be shared between many machines as well. Taking some specific example files:

```
meta-bsp/packages/linux/linux-bsp_2.6.50.bb
```

which is the core kernel recipe which firstly details where to get the kernel source from. All standard source code locations are supported so this could be a release tarball, some git repository, or source included in the directory within the BSP itself. It then contains information about which patches to apply and how to configure and build it. It can reuse the main Poky kernel build class, so the definitions here can remain very simple.

```
linux-bsp-2.6.50/*.patch
```

which are patches which may be applied against the base kernel, wherever they may have been obtained from.

```
meta-bsp/packages/linux/linux-bsp-2.6.50/defconfig-bsp
```

which is the configuration information to use to configure the kernel.

Examples of kernel recipes are available in Poky itself. These files are optional since a kernel from Poky itself could be selected, although it would be unusual not to have a kernel configuration.

## 1.7. Other Software (meta-bsp/packages/*)

This area includes other pieces of software which the hardware may need for best operation. These are just examples of the kind of things that may be encountered. These are standard .bb file recipes in the usual Poky format, so for examples, see standard Poky recipes. The source can be included directly, referred to in source control systems or release tarballs of external software projects.

```
meta-bsp/packages/bootloader/bootloader_0.1.bb
```

Some kind of bootloader recipe which may be used to generate a new bootloader binary. Sometimes these are included in the final image format and needed to reflash hardware.

```
meta-bsp/packages/modem/modem-driver_0.1.bb
meta-bsp/packages/modem/modem-daemon_0.1.bb
```

These are examples of a hardware driver and also a hardware daemon which may need to be included in images to make the hardware useful. "modem" is one example but there may be other components needed like firmware.

```
meta-bsp/packages/image-creator/image-creator-native_0.1.bb
```

Sometimes the device will need an image in a very specific format for its update mechanism to accept and reflash with it. Recipes to build the tools needed to do this can be included with the BSP.

These files only need be provided if the platform requires them.

# 1.8. Append BSP specific information to existing recipes

Say you have a recipe like pointercal which has machine-specific information in it, and then you have your new BSP code in a layer. Before the .bbappend extension was introduced, you'd have to copy the whole pointercal recipe and files into your layer, and then add the single file for your machine, which is ugly. .bbappend makes the above work much easier, to allow BSP-specific information to be merged with the original recipe easily. When bitbake finds any X.bbappend files, they will be included after bitbake loads X.bb but before finalise or anonymous methods run. This allows the BSP layer to poke around and do whatever it might want to customise the original recipe. If your recipe needs to reference extra files it can use the FILESEXTRAPATH variable to specify their location. The example below shows extra files contained in a folder called ${PN} (the package name).

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

Then the BSP could add machine-specific config files in layer directory, which will be added by bitbake. You can look at meta-emenlow/packages/formfactor as an example.

# 1.9. Prebuild Data (meta-bsp/prebuilds/*)

The location can contain a precompiled representation of the source code contained elsewhere in the BSP layer. It can be processed and used by Poky to provide much faster build times, assuming a compatible configuration is used.

These files are optional.

# 1.10. BSP 'Click-through' Licensing Procedure

## Note

This section is here as a description of how click-through licensing is expected to work, and is not yet not impemented.

In some cases, a BSP may contain separately licensed IP (Intellectual Property) for a component, which imposes upon the user a requirement to accept the terms of a 'click-through' license. Once the license is accepted (in whatever form that may be, see details below) the Poky build system can then build and include the corresponding component in the final BSP image. Some affected components may 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. Other components may be simply 'good-to-have' or purely elective, or if essential nonetheless have a 'free' (possibly less-capable) version which may substituted for in the BSP recipe.

For the latter cases, where it is possible to do so from a functionality perspective, the Poky website will make available a 'de-featured' BSP completely free of encumbered IP, which can be used directly and without any further licensing requirements. If present, this fully 'de-featured' BSP will be named meta-bsp (i.e. the normal default naming convention). This is the simplest and therefore preferred option if available, assuming the resulting functionality meets requirements.

If however, a non-encumbered version is unavailable or the 'free' version would provide unsuitable functionality or quality, an encumbered version can be used. Encumbered versions of a BSP are given names of the form meta-bsp-nonfree. There are several ways within the Poky build system to satisfy the licensing requirements for an encumbered BSP, in roughly the following order of preference:

- Get a license key (or keys) for the encumbered BSP by visiting https://pokylinux.org/bsp-keys.html and give the web form there the name of the BSP and your e-mail address.

```
[screenshot of dialog box]
```

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

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

This will 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 may be multiple licenses in effect for a give BSP; a given <keydomain> in such cases corresponds to a particular license. In order for an encumbered BSP encompassing 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.

- Do nothing - build as you normally would, and follow any license prompts that originate from the encumbered BSP (the build will cleanly stop at this point). These usually take the form of instructions needed to manually fetch the encumbered package(s) and md5 sums into e.g. the poky/build/downloads directory. Once the manual package fetch has been completed, restarting the build will continue where it left off, this time without the prompt since the license requirements will have been satisfied.
- Get a full-featured BSP recipe rather than a key, by visiting https://pokylinux.org/bsps.html. Accepting the license agreement(s) presented will subsequently allow you to download a tarball containing a full-featured BSP legally cleared for your use by the just-given license agreement(s). This method will also allow the encumbered image to be built with no change at all to the normal build process.

Note that method 3 is also the only option available when downloading pre-compiled images generated from non-free BSPs. Those images are likewise available at https://pokylinux.org/bsps.html.

# Index