



Starting with the Yocto Project



Alexandre Belloni

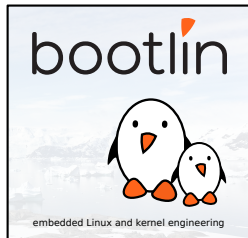
Bootlin

alexandre.belloni@bootlin.com

Put your business card in the box to participate in the raffle!



- ▶ Embedded Linux engineer at **Bootlin**
 - ▶ Embedded Linux **expertise**
 - ▶ **Development**, consulting and training
 - ▶ Strong open-source focus
- ▶ Open-source contributor
 - ▶ Contributing the **kernel support for Atmel ARM processors**
 - ▶ Contributing the **kernel support for Marvell ARM (Berlin) processors**
 - ▶ Maintainer of the Crystalfontz boards in the **meta-fsl-arm** layer



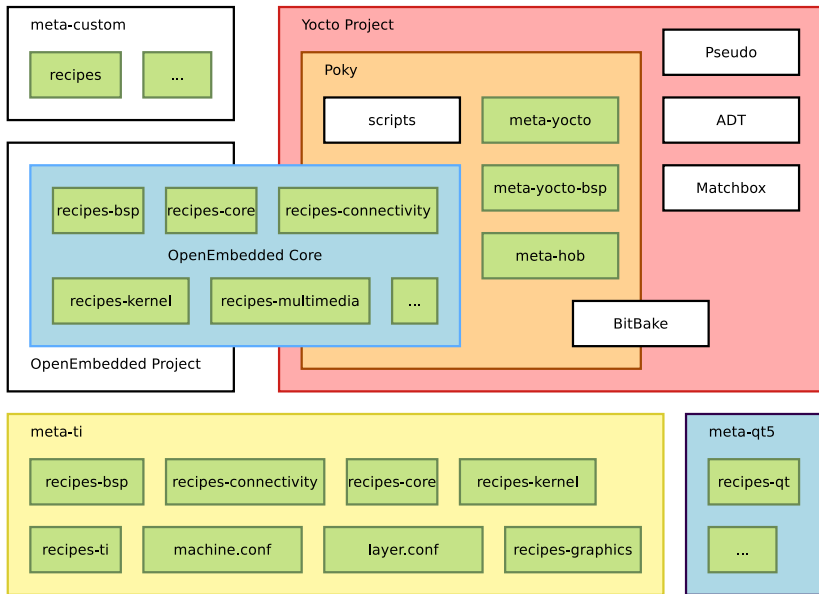


What is the Yocto Project ?

- ▶ Umbrella project, including:
 - ▶ pseudo
 - ▶ cross-prelink
 - ▶ matchbox
 - ▶ opkg
 - ▶ psplash
 - ▶ ...
- ▶ The core components of the Yocto Project are:
 - ▶ BitBake, the *build engine*. It is a task scheduler, like `make`. It interprets configuration files and recipes (also called *metadata*) to perform a set of tasks, to download, configure and build specified packages and filesystem images.
 - ▶ OpenEmbedded-Core, a set of base *layers*. It is a set of recipes, layers and classes which are shared between all OpenEmbedded based systems.
 - ▶ Poky, the *reference system*. It is a collection of projects and tools, used to bootstrap a new distribution based on the Yocto Project.



The Yocto Project lexicon





- ▶ Organization of OpenEmbedded-Core:
 - ▶ *Recipes* describe how to fetch, configure, compile and package applications and images. They have a specific syntax.
 - ▶ *Layers* are sets of recipes, matching a common purpose. For Texas Instruments board support, the *meta-ti* layer is used.
 - ▶ Multiple layers are used within a same distribution, depending on the requirements.
 - ▶ It supports the ARM, MIPS (32 and 64 bits), PowerPC and x86 (32 and 64 bits) architectures.
 - ▶ It supports QEMU emulated machines for these architectures.



Building an image



Environment setup

- ▶ All Poky files are left unchanged when building a custom image.
- ▶ Specific configuration files and build repositories are stored in a separate build directory.
- ▶ A script, `oe-init-build-env`, is provided to set up the build directory and the environment variables (needed to be able to use the `bitbake` command for example).



- ▶ Modifies the environment: has to be sourced!
- ▶ Adds environment variables, used by the build engine.
- ▶ Allows you to use commands provided in Poky.
- ▶ `source ./oe-init-build-env [builddir]`
- ▶ Sets up a basic build directory, named `builddir` if it is not found. If not provided, the default name is `build`.



Common targets

- ▶ Common targets are listed when sourcing the script:
 - core-image-minimal** A small image to boot a device and have access to core command line commands and services.
 - core-image-sato** Image with Sato support. Sato is a GNOME mobile-based user interface.
 - meta-toolchain** Includes development headers and libraries to develop directly on the target.
 - adt-installer** Build the application development toolkit installer.
 - meta-ide-support** Generates the cross-toolchain. Useful when working with the SDK.



The `build/conf/` directory

- ▶ The `conf/` directory in `build` holds build specific configuration.
 - `bblayers.conf` Explicitly list the available layers.
 - `local.conf` Set up the configuration variables relative to the current user for the build. Configuration variables can be overridden there.



Configuring the build

- ▶ The `conf/local.conf` configuration file holds local user configuration variables:
 - BB_NUMBER_THREADS** How many tasks BitBake should perform in parallel.
 - PARALLEL_MAKE** How many processes should be used when compiling.
 - MACHINE** The machine the target is built for, e.g. `beaglebone`.
 - PACKAGE_CLASSES** Packages format (`deb`, `ipk` or `rpm`).



Compilation

- ▶ The compilation is handled by the BitBake *build engine*.
- ▶ Usage: `bitbake [options] [recipeName/target ...]`
- ▶ To build a target: `bitbake [target]`
- ▶ Building a minimal image: `bitbake core-image-minimal`
 - ▶ This will run a full build for the selected target.



Results

`tmp/buildstats/` Build statistics for all packages built (CPU usage, elapsed time, host, timestamps. . .).

`tmp/deploy/` Final output of the build.

`tmp/deploy/images/` Contains the complete images built by the OpenEmbedded build system. These images are used to flash the target.

`tmp/work/` Set of specific work directories, split by architecture. They are used to unpack, configure and build the packages. Contains the patched sources, generated objects and logs.

`tmp/sysroots/` Shared libraries and headers used to compile packages for the target but also for the host.



Summary

- ▶ Initialize the build environment

```
$ source poky/oe-init-build-env build
```

- ▶ Configure your `local.conf`

```
BB_NUMBER_THREADS = "16"  
PARALLEL_MAKE = "-j 16"  
MACHINE ?= "imx28evk"
```

- ▶ build the image

```
$ bitbake core-image-minimal
```



Layers



Layer creation

To make modifications, it is necessary to create a new layer:

- ▶ create a `meta-<machine>` directory
- ▶ inside that directory, create a `conf/layer.conf` file

Alternatively, you could use:

- ▶ `yocto-layer create` and select a high priority
- ▶ or `yocto-bsp create`



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

BBFILES += "${LAYERDIR}/recipes-*/**/*.bb \
           ${LAYERDIR}/recipes-*/**/*.bbappend"

BBFILE_COLLECTIONS += "crystalfontz"
BBFILE_PATTERN_crystalfontz := "^${LAYERDIR}/"
BBFILE_PRIORITY_crystalfontz = "10"

LAYERDEPENDS_crystalfontz = "fsl-arm fsl-arm-extra"
```



Adding the layer to the build

- ▶ The main drawback of having a layer separate from your silicon vendor is that your customers will have to add it to their configuration to use it.
- ▶ That configuration is done in `<builddir>/conf/bblayers.conf`. Add your layer to the `BBLAYERS` variable:

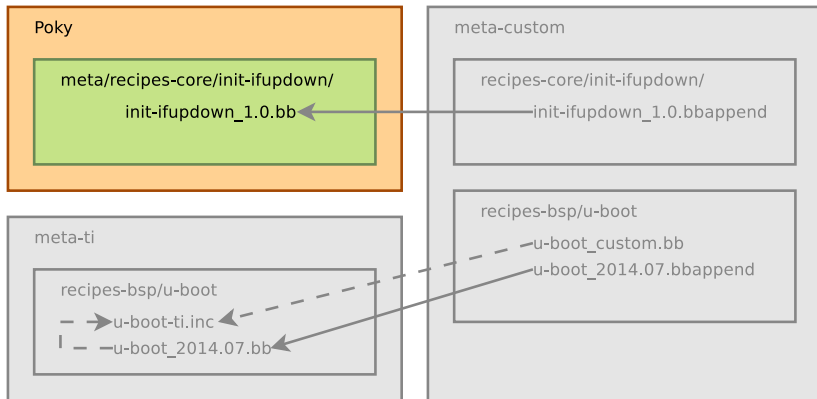
```
BBLAYERS += "${BSPDIR}/sources/meta-crystalfontz "
```



Recipes



Recipes



- extend
- - include/require



- ▶ Recipes describe how to handle a given package.
- ▶ A recipe is a set of instructions to describe how to retrieve, patch, compile, install and generate binary packages for a given application.
- ▶ It also defines what build or runtime dependencies are required.
- ▶ The recipes are parsed by BitBake
- ▶ The format of a recipe file name is `<package-name>_<version>.bb`

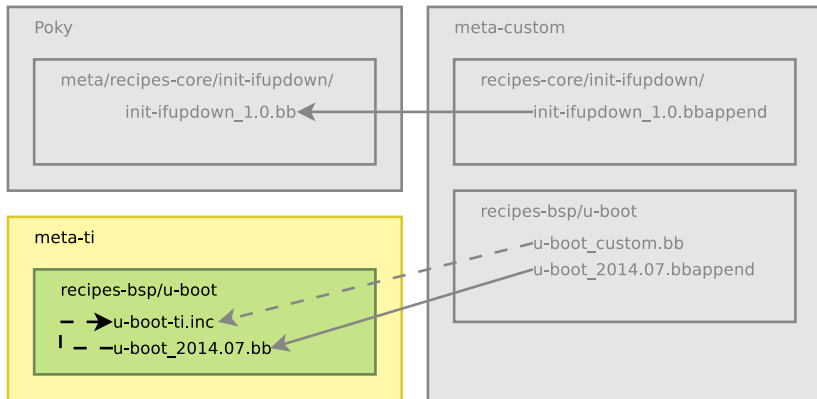


Content of a recipe

- ▶ A recipe contains configuration variables: name, license, dependencies, path to retrieve the source code. . .
- ▶ It also contains functions that can be run (fetch, configure, compile. . .) which are called **tasks**.
- ▶ Tasks provide a set of actions to perform.
- ▶ `bitbake -c <task> <package>`



Organization of a recipe



- extend
- - include/require



Organization of a recipe

- ▶ Many packages have more than one recipe, to support different versions. In that case the common metadata is included in each version specific recipe and is in a `.inc` file:
 - ▶ `<package>.inc`: version agnostic metadata.
 - ▶ `<package>_<version>.bb`: `require <package>.inc` and version specific metadata.
- ▶ We can divide a recipe into three main parts:
 - ▶ The header: what/who
 - ▶ The sources: where
 - ▶ The tasks: how



The header

- ▶ Configuration variables to describe the package:
 - DESCRIPTION** describes what the software is about
 - HOMEPAGE** URL to the project's homepage
 - PRIORITY** defaults to `optional`
 - SECTION** package category (e.g. `console/utils`)
 - LICENSE** the package's license



The source locations

- ▶ We need to retrieve both the raw sources from an official location and the resources needed to configure, patch or install the package.
- ▶ `SRC_URI` defines where and how to retrieve the needed elements. It is a set of URI schemes pointing to the resource locations (local or remote).
- ▶ For the local files, the searched paths are defined in the `FILESPATH` variable, custom ones can be added using `FILESEXTRAPATHS`. BitBake will also search in subfolders listed in the `OVERRIDES` variables in those paths.
- ▶ Files ending in `.patch`, `.diff` or having the `apply=yes` parameter will be applied after the sources are retrieved and extracted.
- ▶ Patches are applied in the order they are found.



Dependencies

- ▶ A package can have dependencies during the build or at runtime. To reflect these requirements in the recipe, two variables are used:
 - DEPENDS** List of the package build-time dependencies.
 - RDEPENDS** List of the package runtime dependencies. Must be package specific (e.g. with `_${PN}`).
- ▶ `DEPENDS = "package_b"`: the local `do_configure` task depends on the `do_populate_sysroot` task of package b.
- ▶ `RDEPENDS_${PN} = "package_b"`: the local `do_build` task depends on the `do_package_write_<archive-format>` task of package b.



Tasks

Default tasks already exists, they are defined in classes:

- ▶ do_fetch
- ▶ do_unpack
- ▶ do_patch
- ▶ do_configure
- ▶ do_compile
- ▶ do_install
- ▶ do_package
- ▶ do_rootfs

You can get a list of existing tasks for a recipe with:

```
bitbake <recipe> -c listtasks
```



Writing tasks 1/3

- ▶ Functions use the sh shell syntax, with available OpenEmbedded variables and internal functions available.
 - ◻ The destination directory (root directory of where the files are installed, before creating the image).
WORKDIR the package's working directory
- ▶ Syntax of a task:

```
do_task() {  
    action0  
    action1  
    ...  
}
```



Writing tasks 2/3

► Example:

```
do_compile() {
    ${CC} ${CFLAGS} ${LDFLAGS} -o hello ${WORKDIR}/hello.c
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 hello ${D}${bindir}
}
```



- ▶ Or using a Makefile:

```
do_compile() {
    oe_runmake
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 hello ${D}${bindir}
}
```



Adding new tasks

Tasks can be added with `addtask`

```
do_mkimage () {  
    uboot-mkimage ...  
}
```

```
addtask mkimage after do_compile before do_install
```




Hello world recipe

```
DESCRIPTION = "Hello world program"
HOMEPAGE = "http://example.net/helloworld/"
PRIORITY = "optional"
SECTION = "examples"
LICENSE = "GPLv2"

SRC_URI = "file://hello.c"

do_compile() {
    ${CC} ${CFLAGS} ${LDFLAGS} -o hello ${WORKDIR}/hello.c
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 hello ${D}${bindir}
}
```



Extending a recipe



Introduction to recipe extensions

- ▶ It is a good practice **not** to modify recipes available in Poky.
- ▶ But it is sometimes useful to modify an existing recipe, to apply a custom patch for example.
- ▶ The BitBake *build engine* allows to modify a recipe by extending it.
- ▶ Multiple extensions can be applied to a recipe.

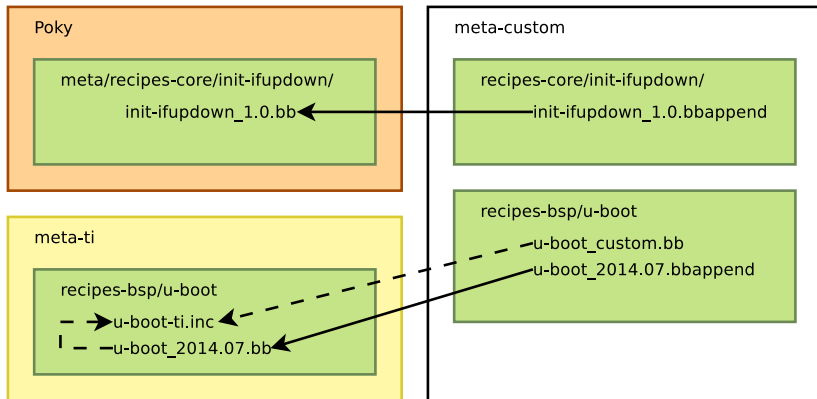


Introduction to recipe extensions

- ▶ Metadata can be changed, added or appended.
- ▶ Tasks can be added or appended.
- ▶ Operators are used extensively, to add, append, prepend or assign values.



Extend a recipe



- extend
- - include/require



Extend a recipe

- ▶ The recipe extensions end in `.bbappend`
- ▶ Append files must have the same root name as the recipe they extend.
 - ▶ `example_0.1.bbappend` applies to `example_0.1.bb`
- ▶ Append files are **version specific**. If the recipe is updated to a newer version, the append files must also be updated.
- ▶ If adding new files, the path to their directory must be prepended to the `FILESEXTRAPATHS` variable.
 - ▶ Files are looked up in paths referenced in `FILESEXTRAPATHS`, from left to right.
 - ▶ Prepending a path makes sure it has priority over the recipe's one. This allows to override recipes' files.



Hello world append file

```
FILESETRAPATHS_prepend := "${THISDIR}/files:"
```

```
SRC_URI += "file://custom-modification-0.patch \  
           file://custom-modification-1.patch \  
           "
```



Warning

You can find it in some recipes but don't use the following construct in `pkg_postinst`:

```
pkg_postinst_wpa-supPLICant () {  
    # If we're offline, we don't need to do this.  
    if [ "x$D" != "x" ]; then  
        exit 0  
    fi  
  
    killall -q -HUP dbus-daemon || true  
}
```

It can't be extended using `.bbappend`



Debugging recipes

- ▶ For each task, logs are available in the `temp` directory in the work folder of a recipe.
- ▶ A development shell, exporting the full environment can be used to debug build failures:

```
$ bitbake -c devshell <recipe>
```

- ▶ To understand what a change in a recipe implies, you can activate build history in `local.conf`:

```
INHERIT += "buildhistory"  
BUILDHISTORY_COMMIT = "1"
```

Then use the `buildhistory-diff` tool to examine differences between two builds.

- ▶ `./scripts/buildhistory-diff`



Images



Image recipes

- ▶ an image recipe is used to define the content of the final image
- ▶ it is the entry point of the build and defines all the necessary packages through dependencies
- ▶ image recipes are usually in `recipes-*/images/`



```
DESCRIPTION = "Image for Crystalfontz boards"
LICENSE = "MIT"

IMAGE_INSTALL = "packagegroup-core-boot \
    ${ROOTFS_PKGMANAGE_BOOTSTRAP} \
    ${CORE_IMAGE_EXTRA_INSTALL}"

IMAGE_INSTALL += "init-ifupdown busybox-udhcpd iw"

IMAGE_INSTALL += "evtest tslib tslib-conf tslib-tests \
    tslib-calibrate"

IMAGE_LINGUAS = " "

inherit core-image
```



Image recipes

- ▶ use `IMAGE_INSTALL` to specify which packages you need on your target
- ▶ you can use packagegroups, they are useful when needing features with complex dependencies
- ▶ inherit the base image class `core-image`
- ▶ you can also include already existing image recipes



images/demo-image-cfa.bb (1)

```
include recipes-sato/images/core-image-sato.bb

IMAGE_FEATURES += "debug-tweaks"
WEB = "web-webkit"

IMAGE_INSTALL += " linux-firmware init-ifupdown busybox-udhcpd"

# we don't need the full tools-testapps
IMAGE_INSTALL += " evtest tslib tslib-conf tslib-tests tslib-calibrate xev"
IMAGE_INSTALL += " iw connman-client"

EXTRA_IMAGE_FEATURES += " \
    nfs-server \
    qt4-pkgs \
"

# more debugging and profiling
EXTRA_IMAGE_FEATURES += " \
    tools-debug \
    tools-profile \
"
```



```
IMAGE_INSTALL += " \  
    cpufrequtils \  
    nano \  
    packagegroup-qt-in-use-demos \  
    qt4-demos \  
    qt4-examples \  
    cfa-config-extra \  
    "  
  
export IMAGE_BASENAME = "demo-image-cfa"
```



IMAGE_FEATURES The primary list of features to include in an image.

EXTRA_IMAGE_FEATURES List of additional features to include in an image, typically to be put in your `local.conf` file.

Available features: `dbg-pkgs`, `dev-pkgs`, `doc-pkgs`, `nfs-server`, `read-only-rootfs`, `splash`, `ssh-server-dropbear`, `ssh-server-openssh`, `staticdev-pkgs`, `tools-debug`, `tools-profile`, `tools-sdk`, `tools-testapps`, `x11`, `x11-base`, `x11-sato`



Image tweaks

There is a mechanism to describe what functionalities are available on the target, the formfactor configuration file.

- ▶ extend it with a `.bbappend`:

```
recipes-bsp/formfactor/formfactor_0.0.bbappend
```

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

- ▶ it install a file named `machconfig`

```
$ tree recipes-bsp/formfactor/  
recipes-bsp/formfactor/  
|-- formfactor  
|   |-- cfa10057  
|   |   '-- machconfig  
|   '-- cfa10058  
|       '-- machconfig  
'-- formfactor_0.0.bbappend
```



Display options

```
HAVE_TOUCHSCREEN=1
```

Other available variables: HAVE_KEYBOARD,
HAVE_KEYBOARD_PORTRAIT, HAVE_KEYBOARD_LANDSCAPE,
DISPLAY_CAN_ROTATE, DISPLAY_ORIENTATION,
DISPLAY_WIDTH_PIXELS, DISPLAY_HEIGHT_PIXELS,
DISPLAY_BPP, DISPLAY_WIDTH_MM, DISPLAY_HEIGHT_MM,
DISPLAY_SUBPIXEL_ORDER.



Adding extra configuration

You can create a recipe to simply install a few configuration files in your final filesystem. This is what `cfa-config-extra` is doing:
`recipes/cfa-config-extra/cfa-config-extra.bb`

```
DESCRIPTION = "Extra files for demo-image-cfa"
LICENSE = "GPLv2"
PR = "r1"
S="${WORKDIR}"
LIC_FILES_CHKSUM = "file://LICENSE;md5=c746876a5e2eaefef09efb9d7c1c463d"

SRC_URI += "file://qtbrowser.desktop \
            file://webkit.png \
            file://qtmediaplayer.desktop \
            file://qtmediaplayer.png \
            file://qtdemo.desktop \
            file://qtdemo.png \
            file://LICENSE"

inherit allarch

do_install () {
    install -d ${D}/${datadir}/pixmaps
    install -d ${D}/${datadir}/applications
    install -m 0644 ${WORKDIR}/webkit.png ${D}/${datadir}/pixmaps
    install -m 0644 ${WORKDIR}/qtbrowser.desktop ${D}/${datadir}/applications
    install -m 0644 ${WORKDIR}/qtmediaplayer.png ${D}/${datadir}/pixmaps
    install -m 0644 ${WORKDIR}/qtmediaplayer.desktop ${D}/${datadir}/applications
    install -m 0644 ${WORKDIR}/qtdemo.png ${D}/${datadir}/pixmaps
    install -m 0644 ${WORKDIR}/qtdemo.desktop ${D}/${datadir}/applications
}
```



Useful package tweaks

recipes-connectivity/connman, to be extended to install and connman.defaults file, especially to prevent connman from configuring some interfaces.

recipes-connectivity/connman/connman_1.17.bbappend

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"  
SRC_URI += " file://connman.defaults"  
  
do_install_append() {  
    if {@base_contains('DISTRO_FEATURES', 'sysvinit', 'true', 'false', d)}; then  
        install -d ${D}${sysconfdir}/default  
        install -m 0755 ${WORKDIR}/connman.defaults \  
            ${D}${sysconfdir}/default/connman  
    fi  
}
```

recipes-connectivity/connman/connman/connman.defaults

```
EXCLUDED_INTERFACES="usb0"
```



Useful package tweaks

recipes-core/busybox, to be extended to install various configuration files for the busybox applets
recipes-core/busybox/busybox_1.21.1.bbappend:

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

SRC_URI_append_cfa10036 = " \
    file://udhcpd.conf \
"

do_install_append_cfa10036 () {
    install -m 0755 ${WORKDIR}/udhcpd.conf ${D}${sysconfdir}/
}
```

This recipe is better than the previous one as it restrict changes to a particular machine.



Useful package tweaks

recipes-core/psplash, can be extended to change the splash screen, needs more to change the color of the progress bar:
recipes-core/psplash/psplash_git.bbappend:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
DEPENDS += "gdk-pixbuf-native"
PRINC = "g"
SRC_URI += "file://psplash-colors.h \
            file://psplash-bar-img.png"

# NB: this is only for the main logo image; if you add multiple images here,
# poky will build multiple psplash packages with 'outsuffix' in name for
# each of these ...
SPLASH_IMAGES = "file://psplash-poky-img.png;outsuffix=default"

# The core psplash recipe is only designed to deal with modifications to the
# 'logo' image; we need to change the bar image too, since we are changing
# colors
do_configure_append () {
    cd ${S}
    cp ../psplash-colors.h ./
    # strip the -img suffix from the bar png -- we could just store the
    # file under that suffix-less name, but that would make it confusing
    # for anyone updating the assets
    cp ../psplash-bar-img.png ./psplash-bar.png
    ./make-image-header.sh ./psplash-bar.png BAR
}
```



BSP

BSP



Machine configuration

Create a `<machine>.conf` file in `conf/machine/`. As we want to support multiple similar boards (all based on `cfa10036`), an include was created in `conf/machine/include/`.



```
# Common definitions for cfa-10036 boards
include conf/machine/include/mxs-base.inc

SOC_FAMILY = "mxs:mx28:cfa10036"

PREFERRED_PROVIDER_virtual/kernel ?= "linux-cfa"
IMAGE_BOOTLOADER = "barebox"
BAREBOX_BINARY = "barebox"
IMXBOOTLETS_MACHINE = "cfa10036"
KERNEL_IMAGETYPE = "zImage"
KERNEL_DEVICETREE = "imx28-cfa10036.dtb"

# we need the kernel to be installed in the final image
IMAGE_INSTALL_append = " kernel-image kernel-devicetree"

SDCARD_ROOTFS ?= "${DEPLOY_DIR_IMAGE}/${IMAGE_NAME}.rootfs.ext3"
IMAGE_FSTYPES ?= "tar.bz2 ext3 barebox.mxsboot-sdcard sdcard"

SERIAL_CONSOLE = "115200 ttyAMA0"
MACHINE_FEATURES = "usb gadget usbhost vfat"
```



The machine configuration for the module is simple:

```
#@TYPE: Machine
#@NAME: Crystalfontz CFA-10036
#@SOC: i.MX28
#@DESCRIPTION: Machine configuration for CFA-10036
#@MAINTAINER: Alexandre Belloni <alexandre.belloni@bootlin.com>

include conf/machine/include/cfa10036.inc
```

It is always a good idea to put a contact as maintainer.



For a carrier board, add the corresponding device tree and the supported features.

```
#@TYPE: Machine
#@NAME: Crystalfontz CFA-10057
#@SOC: i.MX28
#@DESCRIPTION: Machine configuration for CFA-10057, also ca
#@MAINTAINER: Alexandre Belloni <alexandre.belloni@bootlin

include conf/machine/include/cfa10036.inc

KERNEL_DEVICETREE += "imx28-cfa10057.dtb"

MACHINE_FEATURES += "touchscreen"
```



For the kernel, you have multiple choices:

- ▶ patches over silicon vendor kernel tree
 - ▶ available as an include
 - ▶ using a `.bbappend`
- ▶ custom git tree
- ▶ mainline git

You also probably have to provide a configuration file.



Patches, include

The compilation logic is provided by your silicon vendor as an include file:

- ▶ create a `recipes-kernel/linux/`
- ▶ write a new recipe `linux-<vendor>_<version>.bb`
- ▶ copy your patches to
`recipes-kernel/linux/linux-<vendor>-<version>`
- ▶ Example: for `linux-congatec`:

```
$ ls recipes-kernel/linux/linux-congatec*
recipes-kernel/linux/linux-congatec_3.0.35.bb

recipes-kernel/linux/linux-congatec-3.0.35:
0001-Add-linux-support-for-congatec-evaluation-board-qmx6q.patch
0001-perf-tools-Fix-getrusage-related-build-failure-on-gl.patch
0002-ARM-7668-1-fix-memset-related-crashes-caused-by-rece.patch
0003-ARM-7670-1-fix-the-memset-fix.patch
[...]
defconfig
```



Patches, include: recipe

```
SUMMARY = "Linux Kernel based on Freescale Linux kernel to add support for Cong  
include recipes-kernel/linux/linux-imx.inc
```

```
SRCREV = "bdde708ebfde4a8c1d3829578d3f6481a343533a"
```

```
LOCALVERSION = "-4.1.0+yocto"
```

```
SRCBRANCH = "imx_3.0.35_4.1.0"
```

```
SRC_URI += "file://drm-vivante-Add-00-suffix-in-retuned-bus-Id.patch \  
file://epdc-Rename-mxcfb_epdc_kernel.h-to-mxc_epdc.h.patch \  
file://0001-perf-tools-Fix-getrusage-related-build-failure-on-gl.pa  
file://0002-ARM-7668-1-fix-memset-related-crashes-caused-by-rece.pa  
file://0003-ARM-7670-1-fix-the-memset-fix.patch \  
file://0004-ENGR00271136-Fix-build-break-when-CONFIG_CLK_DEBUG-i.pa  
file://0005-ENGR00271359-Add-Multi-touch-support.patch \  
file://0006-Add-support-for-DVI-monitors.patch \  
file://0001-Add-linux-support-for-congatec-evaluation-board-qmx6q.p  
file://ENGR00278350-gpu-viante-4.6.9p13-kernel-part-integra.patch \  
"
```

```
COMPATIBLE_MACHINE = "(cgtqmx6)"
```



Patches, include: recipe

SRCREV The revision of the source code used to build the package.

SRCBRANCH New in `daisy`, when using `git` it is required to specify in which branch the commit resides.

`SRCBRANCH` is used in `SRC_URI`, in `linux-imx.inc`

SRC_URI The list of source files. Here patches are added in the original `SRC_URI`

COMPATIBLE_MACHINE A regular expression used to match against the `MACHINEOVERRIDES` variable which in turn includes `MACHINE`. Used to ensure the recipe won't build for other machines.



- ▶ When using `file://` in `SRC_URI`, OpenEmbedded will search files relative to the subdirectories listed in `FILESPATH`
- ▶ By default, this is:
 - ▶ `${BPN}`, the base recipe name
 - ▶ `${BP}`, which is `${BPN}-${PV}`, `${PV}` being the package version
 - ▶ `files`
- ▶ also looks in a subdirectory named `${MACHINE}` inside those directories
- ▶ if set, also looks for subdirectories named from `${MACHINEOVERRIDES}` and `${DISTROOVERRIDES}`
- ▶ Don't modify `FILESPATH` directly, use `FILESEXTRAPATHS`



When using a custom git tree, you'll have to write your own recipe. But this doesn't have to be difficult:

- ▶ inherit the kernel class, it already takes care of downloading, unpacking, configuring and compiling your kernel.
- ▶ if using device trees, include `recipes-kernel/linux/linux-dtb.inc`
- ▶ define `SRC_URI`
- ▶ define `S`
- ▶ define `COMPATIBLE_MACHINE`



```
DESCRIPTION = "Linux kernel for Crystalfontz boards"
SECTION = "kernel"
LICENSE = "GPLv2"

LIC_FILES_CHKSUM = "file://COPYING;md5=d7810fab7487fb0aad327b76f1be7cd7"

inherit kernel
require recipes-kernel/linux/linux-dtb.inc

SRCBRANCH = "cfa-3.10.25"
SRC_URI = "git://github.com/crystalfontz/cfa_10036_kernel;branch=${SRCBRANCH} \
          file://defconfig"

SRCREV = "61dbe8ef338ce4cc1c10d5a6cdd418c047fb136d"

S = "${WORKDIR}/git"

COMPATIBLE_MACHINE = "cfa10036"
```



Bootloader support

- ▶ using `imxbootlets` to start Barebox
- ▶ the recipes are going in the `recipes-bsp` folder
- ▶ those recipes are extended using `.bbappend`



- ▶ extends
recipes-bsp/imx-bootlets/imx-bootlets_10.12.01.bb
from meta-fsl-arm

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

```
SRC_URI_append_cfa10036 = " file://cfa10036-support.patch"
```

- ▶ use immediate expansion :=
- ▶ conditionally adds the cfa10036 support patch when
MACHINEOVERRIDES matches
- ▶ don't forget the space at the beginning of the string when
using `_append`



- ▶ extends `recipes-bsp/barebox/barebox_2013.08.0.bb` from `meta-fsl-arm`

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}-${PV}:"
```

```
COMPATIBLE_MACHINE_cfa10036 = "cfa10036"
```

- ▶ simply adds the subdirectory, it contains the configuration

```
$ tree recipes-bsp/barebox/barebox-2013.08.0/  
recipes-bsp/barebox/barebox-2013.08.0/  
  '-- cfa10036  
    '-- defconfig
```



- ▶ <https://www.yoctoproject.org/documentation>
- ▶ in particular the variable glossary:
<http://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#ref-variables-glossary>
- ▶ and the BSP developer's guide:
<http://www.yoctoproject.org/docs/current/bsp-guide/bsp-guide.html>
- ▶ Freescale BSP: <http://freescale.github.io>



Training

- ▶ We teach a 3 day course on Yocto Project and OpenEmbedded development
- ▶ Either on your company site
- ▶ Or in a public session, next ones: March 9-11, 2015 in Lyon, in French or May 20-22, 2015 in Paris, in English
- ▶ Info and materials available at:
<http://bootlin.com/training/yocto/>
- ▶ training@bootlin.com

Questions?

Alexandre Belloni

`alexandre.belloni@bootlin.com`

Slides under CC-BY-SA 3.0

<http://bootlin.com/pub/conferences/2015/fosdem/belloni-starting-with-YP/>