Blog

Including Custom Executables and Libraries in Your Embedded Linux Image with Yocto Project

Posted on November 26, 2019 by Aaron Crump

Tags: **LINUX (HTTPS://DORNERWORKS.COM/TAG/LINUX/)**, **OS (HTTPS://DORNERWORKS.COM/TAG/OS/)**, **SOFTWARE (HTTPS://DORNERWORKS.COM/TAG/SOFTWARE/)**, **YOCTO (HTTPS://DORNERWORKS.COM/TAG/YOCTO/)**

Yocto customization is power

Yocto is incredibly flexible. It enables building an embedded Linux distribution with virtually any combination of packages in the root filesystem (rootfs). This flexibility also makes Yocto quite powerful. Understanding how to include custom files in the rootfs can

Most Used Tags (/blog/)

IOT (HTTPS://DORNERWORKS.COM/TAG/IOT/)

FPGA (HTTPS://DORNERWORKS.COM/TAG/FPGA/)

VIRTUALIZATION (HTTPS://DORNERWORKS.COM/TAG/VIRTUALIZATION/)

SEL4 (HTTPS://DORNERWORKS.COM/TAG/SEL4/)

SECURITY (HTTPS://DORNERWORKS.COM/TAG/SECURITY/)

PRODUCT DEVELOPMENT (HTTPS://DORNERWORKS.COM/TAG/PRODUCTDEVELOPM

SOFTWARE (HTTPS://DORNERWORKS.COM/TAG/SOFTWARE/)

MEDICAL (HTTPS://DORNERWORKS.COM/TAG/MEDICAL/)

LINUX (HTTPS://DORNERWORKS.COM/TAG/LINUX/)

XILINX (HTTPS://DORNERWORKS.COM/TAG/XILINX/)

CYBERSECURITY (HTTPS://DORNERWORKS.COM/TAG/CYBERSECURITY/)

MICROCHIP (HTTPS://DORNERWORKS.COM/TAG/MICROCHIP/)

AMD (HTTPS://DORNERWORKS.COM/TAG/AMD/)

CAREERS (HTTPS://DORNERWORKS.COM/TAG/CAREERS/)

SPACE (HTTPS://DORNERWORKS.COM/TAG/SPACE/)

VIEW ALL (/BLOG/)

simplify the process of creating custom Linux images. This blog outlines a collection of recipes that guided me through the sea of misinformation to the process of correctly including a large custom application.

As previously recommended, you can install Yocto on your system (/blog/heres-how-you-can-build-your-own-custom-linux-distro-with-yocto/) with at least **8GB of RAM** and **120GB of hard drive space**, then prepare yourself and expand your Yocto knowledge with the Mega-Manual (https://www.yoctoproject.org/docs/2.5/mega-manual/mega-manual.html) and these steps. Once you have Yocto set up, follow these steps to customize your Linux image with your own executables and shared libraries.

Adding a meta-layer and including a basic executable

Including a basic executable in a custom Yocto image can be done quite easily by following the steps below. All of the steps listed here were tested using Yocto Sumo version running in Centos 7. This example is based on an example that can be found in the Yocto Mega-Manual here (https://wiki.yoctoproject.org/wiki/Building_your_own_recipes_from_first_principles).

1. Add a new meta-layer and configuration files for that meta-layer:

Create the necessary directory structure for your meta-layer.

```
$ mkdir -p ~/poky/meta-test/conf
```

Add a layer.conf file to configure your meta-layer.

```
$ cd ~/poky/meta-test/conf && touch layer.conf
```

Edit your layer.conf file to define paths to your recipes, and your layer version.

```
$ vi layer.conf
```

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

# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
${LAYERDIR}/recipes-*/*/*.bbappend \
${LAYERDIR}/recipes-*/*.bb \
${LAYERDIR}/recipes-*/*.bbappend "

BBFILE_COLLECTIONS += "test"
BBFILE_PATTERN_test = "^${LAYERDIR}/"
BBFILE_PRIORITY_test = "1"
LAYERVERSION_test = "1"
```

2. Create a directory for your recipe, and add a helloworld recipe:

```
$ cd ~/poky/meta-test
```

```
$ mkdir -p recipes-test/helloworld && cd recipes-
test/helloworld

$ touch helloworld.bb
```


Use your newly created recipe to build the helloworld executable:

```
$ vi helloworld.bb
```

This is an example of a basic recipe. The do_compile function tells Yocto how to build the executable. The variables CC, CFLAGS, and LDFLAGS are set by Yocto to defaults for the machine that is specified. The do_install function lays out what directories and files are to be added to the rootfs. The S variable is the location where files are used at build time, and where output binaries will be placed. The SRC_URI variable is used to define the location of source files before build time.

```
DESCRIPTION = "Example Hello, World application for Yocto
build."
SECTION = "examples"
DEPENDS = ""
LICENSE = "CLOSED"

FILESEXTRAPATHS_prepend := "${THISDIR}/src:"

SRC_URI = "file://helloworld.c"

S = "${WORKDIR}"

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

do_install() {
# create the /usr/bin folder in the rootfs with default
permissions
install -d ${D}${bindir}

# install the application into the /usr/bin folder with
default permissions
install ${WORKDIR}/hello ${D}${bindir}
}
```

3. Add source code for the helloworld recipe to your newly created recipes-test folder:

```
$ cd ~/poky/meta-test/recipes-test/helloworld

$ mkdir src && cd src
```

```
$ touch helloworld.c && vi helloworld.c
```

```
# Include <stdio.h>
int main()
{
printf("Hello, World\n");
return 0;
};
```


4. Create a bbappend for the selected image, in order to include the executable in that image:

```
$ cd ~/poky/meta-test && mkdir recipes-images

$ cd recipes-images

$ touch core-image-minimal.bbappend && vi core-image-
minimal.bbappend
```

The IMAGE_INSTALL variable defines which recipes will be used to make up the rootfs of the final image. The helloworld recipe that was created will be appended to the primary list of packages.

```
IMAGE_INSTALL_append = " helloworld "
```

5. Add your custom meta-layer to the bblayers.conf file, so that Yocto knows to include it:

```
$ cd ~/poky
```

Sourcing the oe-init-build-env script will setup a new build directory and place default configuration files in the build/conf directory.

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

The default layer configuration file must be edited to add the newly created meta-layer to the resources for this build.

```
$ vi conf/bblayers.conf
```

```
BBLAYERS = " \
…
~/poky/meta-test \
"
```

6. Build the chosen image by executing the bitbake command in the shell from the build directory.

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

The helloworld executable will now be included in the rootfs of the image.

Including a shared header file

There are many instances where you may wish to include a shared header file in the rootfs of an image. If a library is being written that will be utilized by more than one application it can easily be included alongside other standard Linux headers using the following steps.

1. Add the source code for the header file to your meta-layer:

```
$ cd ~/poky/meta-test/recipes-test/helloworld/src && touch
helloworld.h

$ vi helloworld.h
```

```
#Include <stdio.h>

int helloworld(void)
{
printf("Hello, World!\n");
return 0;
}
```

2. Edit your recipe to include the header file in the rootfs:

```
$ vi ~/poky/meta-test/recipes-test/helloworld/helloworld.bb
```

```
DESCRIPTION = "Example Hello, World application for Yocto
build."
SECTION = "examples"
DEPENDS = ""
LICENSE = "CLOSED"

FILESEXTRAPATHS_prepend := "${THISDIR}/src:"

SRC_URI = "file://helloworld.c \
helloworld.h"

S = "${WORKDIR}"

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

do_install() {
# create the /usr/bin folder in the rootfs give it default
permissions
install -d ${D}${bindir}
```

```
# add the /usr/include folder to the sysroot for this
recipe, to be
# added to the final rootfs
install -d ${D}${includedir}

# install the application into the /usr/bin folder
install ${S}/hello ${D}${bindir}

# install the header file in /usr/include with default
permissions
install ${S}hello.h ${D}${includedir}
}
```


3. The image append must also be edited. Header files are not considered normal packages; Yocto defines them as development packages. We must also include the development version of our recipe.

```
$ vi ~/poky/meta-test/recipes-images/core-image-
minimal.bbappend“`
```

```
IMAGE_INSTALL_append = ” helloworld \
helloworld-dev”
```

The header file should now be included in the /usr/include directory with the rest of the Linux headers.

Include a prebuilt shared library

It is frequently helpful to include prebuilt libraries in the rootfs of an image. These prebuilt libraries take the form of .so files located in /usr/lib, they allow for other applications to easily link against them. The helloworld application that we have been using can be repurposed to be included as a library, using the process laid out in the Yocto Project Wiki (https://wiki.yoctoproject.org/wiki/TipsAndTricks/Packaging_Prebuilt_Libraries).

1. Edit the recipe to include the application as a library instead of an executable. There are a few things to make note of in this recipe.
    - Bitbake provides a function to include libraries called “oe_soinstall.” It has been used here, in favor of the normal install function.
    - A QA error will be thrown if you attempt to include a non-versioned library, therefore this library has been built with the version of the recipe attached, as libhelloworld.so.${PV}.
    - The PV variable contains the package version. If it is not defined in the recipe, it defaults to 1.0.
    - Yocto requires that the ELF tag SONAME must be included in the library when it is built, thus, the flag “-Wl,-soname,libhelloworld.so.${PV}” has been added to the do_compile step.

```
DESCRIPTION = “Example Hello, World application for Yocto
build.”
SECTION = “examples”
```

```
DEPENDS = ""
LICENSE = "CLOSED"

FILESEXTRAPATHS_prepend := "${THISDIR}/src:"

SRC_URI = "file://helloworld.c"

S = "${WORKDIR}"

do_compile() {
${CC} ${CFLAGS} ${LDFLAGS} -shared -fPIC -Wl,-
soname,libhelloworld.so.${PV} \
${WORKDIR}/helloworld.c -o libhelloworld.so.${PV}
}

do_install() {

# add the /usr/lib folder to the sysroot for this recipe,
to be
# added to the final rootfs
install -d ${D}${libdir}

# install the prebuilt library in /usr/lib with default
permissions
oe_soinstall ${S}/libhelloworld.so.${PV} ${D}${libdir}
}
```

2. Bitbake the recipe. Shared libraries are also considered development packages,so ensure that "helloworld-dev" remains in the IMAGE_INSTALL variable in your bbappend.

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

Conclusion

The recipe examples here should allow any user to include custom applications in multiple ways. These instructions will allow you to improve your Yocto recipes by giving you the tools to include custom applications and libraries easily.

<< PREVIOUS POST (HTTPS://DORNERWORKS.COM/BLOG/BUILD-YOUR-OWN-FILESYSTEM/)

NEXT POST >> (HTTPS://DORNERWORKS.COM/BLOG/RENODE-RISCV-SEL4/)

(/)

by Aaron Crump

EMBEDDED ENGINEER WHAT WE DO (HTTPS://DORNERWORKS.COM/SOLUTIONS/) WHO WE ARE (HTTPS://DORNERW

(mailto:aaron.crump@dornerworks.com)

GET A JOB (HTTPS://DORNERWORKS.COM/CAREERS/) **SCHEDULE A MEETING (HTTPS://DORNERWORKS.COM/CC**

Aaron Crump is an embedded engineer at DornerWorks.