Yocto Project Quick Build

Welcome!

This short document steps you through the process for a typical image build using the Yocto Project. The document also introduces how to configure a build for specific hardware. You will use Yocto Project to build a reference embedded OS called Poky.

Compatible Linux Distribution

Make sure your Build Host meets the following requirements:

- At least 90 Gbytes of free disk space, though much more will help to run multiple builds and increase performance by reusing build artifacts.
- At least 8 Gbytes of RAM, though a modern modern build host with as much RAM and as many CPU cores as possible is strongly recommended to maximize build performance.
- Runs a supported Linux distribution (i.e. recent releases of Fedora, openSUSE, CentOS, Debian, or Ubuntu). For a list of Linux distributions that support the Yocto Project, see the Supported Linux Distributions section in the Yocto Project Reference Manual. For detailed information on preparing your build host, see the Preparing the Build Host section in the Yocto Project Development Tasks Manual.

- Git 1.8.3.1 or greater
- tar 1.28 or greater
- Python 3.8.0 or greater.
- gcc 8.0 or greater.
- GNU make 4.0 or greater

If your build host does not meet any of these three listed version requirements, you can take steps to prepare the system so that you can still use the Yocto Project. See the Required Git, tar, Python, make and gcc Versions section in the Yocto Project Reference Manual for information.

Build Host Packages

You must install essential host packages on your build host. The following command installs the host packages based on an Ubuntu distribution:

```
$ sudo apt install build-essential chrpath cpio debianutils diffstat file gawk gcc git iputils-ping libacl1 liblz4-tool locales python3 python3-git python3-jinja2 python3-pexpect python3-pip python3-subunit socat texinfo unzip wget xz-utils zstd
```

Use Git to Clone Poky

Once you complete the setup instructions for your machine, you need to get a copy of the Poky repository on your build host. Use the following commands to clone the Poky repository.

```
$ git clone git://git.yoctoproject.org/poky
```

```
Cloning into 'poky'...

remote: Counting

objects: 432160, done. remote: Compressing objects: 100%

(102056/102056), done. remote: Total 432160 (delta 323116), reused

432037 (delta 323000) Receiving objects: 100% (432160/432160), 153.81 MiB | 8.54 MiB/s, done.

Resolving deltas: 100% (323116/323116), done.

Checking connectivity... done.
```

Go to Releases wiki page, and choose a release codename (such as styhead), corresponding to either the latest stable release or a Long Term Support release.

Then move to the poky directory and take a look at existing branches:

```
$ cd poky

$ git branch -a
```

```
.

.

.

remotes/origin/HEAD -> origin/master

remotes/origin/dunfell

remotes/origin/dunfell-next

.

.

.

remotes/origin/gatesgarth

remotes/origin/gatesgarth-next

.

.
```

```
.

remotes/origin/master

remotes/origin/master-next

.

.

.
```

For this example, check out the styhead branch based on the Styhead release:

```
$ git checkout -t origin/styhead -b my-styhead

Branch 'my-styhead' set up to track remote branch 'styhead' from 'origin'.

Switched to a new branch 'my-styhead'
```

The previous Git checkout command creates a local branch named my-styhead. The files available to you in that branch exactly match the repository's files in the styhead release branch.

Note that you can regularly type the following command in the same directory to keep your local files in sync with the release branch:

```
$ git pull
```

For more options and information about accessing Yocto Project related repositories, see the Locating Yocto Project Source Files section in the Yocto Project Development Tasks Manual.

Building Your Image

Use the following steps to build your image. The build process creates an entire Linux distribution, including the toolchain, from source.

Note

- If you are working behind a firewall and your build host is not set up for proxies, you could encounter problems with the build process when fetching source code (e.g. fetcher failures or Git failures).
- If you do not know your proxy settings, consult your local network infrastructure resources and get that information. A good starting point could also be to check your web browser settings. Finally, you can find more information on the "Working Behind a Network Proxy" page of the Yocto Project Wiki.

Initialize the Build Environment: From within the poky directory, run the oe-init-build-env environment setup script to define Yocto Project's build environment on your build host.

$ cd poky

$ source oe-init-build-env

You had no conf/local.conf file. This configuration file has therefore been

created for you with some default values. You may wish to edit it to, for

example, select a different MACHINE (target hardware). See conf/local.conf

for more information as common configuration options are commented.

You had no conf/bblayers.conf file. This configuration file has therefore

been created for you with some default values. To add additional metadata

layers into your configuration please add entries to conf/bblayers.conf.

The Yocto Project has extensive documentation about OE including a reference

manual which can be found at:

    https://docs.yoctoproject.org

For more information about OpenEmbedded see their website:

    https://www.openembedded.org/

Shell environment set up for builds.

You can now run 'bitbake <target>'

Common targets are:

    core-image-minimal

    core-image-full-cmdline

core-image-sato

core-image-weston

meta-toolchain

meta-ide-support

You can also run generated QEMU images with a command like 'runqemu qemux86-64'

Other commonly useful commands are:

- 'devtool' and 'recipetool' handle common recipe tasks

- 'bitbake-layers' handles common layer tasks

- 'oe-pkgdata-util' handles common target package tasks

1. Among other things, the script creates the Build Directory, which is build in this case and is located in the Source Directory. After the script runs, your current working directory is set to the Build Directory. Later, when the build completes, the Build Directory contains all the files created during the build.

Examine Your Local Configuration File: When you set up the build environment, a local configuration file named local.conf becomes available in a conf subdirectory of the Build Directory. For this example, the defaults are set to build for a qemux86 target, which is suitable for emulation. The package manager used is set to the RPM package manager.

Tip

You can significantly speed up your build and guard against fetcher failures by using Shared State Cache mirrors and enabling Hash Equivalence. This way, you can use pre-built artifacts rather than building them. This is relevant only when your network and the server that you use can download these artifacts faster than you would be able to build them.

To use such mirrors, uncomment the below lines in your conf/local.conf file in the Build Directory:

```
BB_HASHSERVE_UPSTREAM = "wss://hashserv.yoctoproject.org/ws"

SSTATE_MIRRORS ?= "file://.* http://cdn.jsdelivr.net/yocto/sstate/all/PATH;downloadfilename=PATH"

BB_HASHSERVE = "auto"

BB_SIGNATURE_HANDLER = "OEEquivHash"
```

2. The hash equivalence server needs the websockets python module version 9.1 or later. Debian GNU/Linux 12 (Bookworm) and later, Fedora, CentOS Stream 9 and later, and Ubuntu 22.04 (LTS) and later, all have a recent enough package. Other supported distributions need to get the module some other place than their package feed, e.g. via pip.

Start the Build: Continue with the following command to build an OS image for the target, which is core-image-sato in this example:

```
$ bitbake core-image-sato
```

3. For information on using the bitbake command, see the BitBake section in the Yocto Project Overview and Concepts Manual, or see The BitBake Command in the BitBake User Manual.

Simulate Your Image Using QEMU: Once this particular image is built, you can start QEMU, which is a Quick EMUlator that ships with the Yocto Project:

```
$ runqemu qemux86-64
```

4. If you want to learn more about running QEMU, see the Using the Quick EMUlator (QEMU) chapter in the Yocto Project Development Tasks Manual.
5. Exit QEMU: Exit QEMU by either clicking on the shutdown icon or by typing Ctrl-C in the QEMU transcript window from which you evoked QEMU.

Customizing Your Build for Specific Hardware

So far, all you have done is quickly built an image suitable for emulation only. This section shows you how to customize your build for specific hardware by adding a hardware layer into the Yocto Project development environment.

In general, layers are repositories that contain related sets of instructions and configurations that tell the Yocto Project what to do. Isolating related metadata into functionally specific layers facilitates modular development and makes it easier to reuse the layer metadata.

Note

By convention, layer names start with the string “meta-“.

Follow these steps to add a hardware layer:

1. Find a Layer: Many hardware layers are available. The Yocto Project Source Repositories has many hardware layers. This example adds the meta-altera hardware layer.

Clone the Layer: Use Git to make a local copy of the layer on your machine. You can put the copy in the top level of the copy of the Poky repository created earlier:

```
$ cd poky

$ git clone https://github.com/kraj/meta-altera.git

Cloning into 'meta-altera'...

remote: Counting objects: 25170, done.

remote: Compressing objects: 100% (350/350), done.
```

```
remote: Total 25170 (delta 645), reused 719 (delta 538), pack-reused 24219

Receiving objects: 100% (25170/25170), 41.02 MiB | 1.64 MiB/s, done.

Resolving deltas: 100% (13385/13385), done.

Checking connectivity... done.
```

2. The hardware layer is now available next to other layers inside the Poky reference repository on your build host as meta-altera and contains all the metadata needed to support hardware from Altera, which is owned by Intel.

   Note

   It is recommended for layers to have a branch per Yocto Project release. Please make sure to checkout the layer branch supporting the Yocto Project release you're using.

3. Change the Configuration to Build for a Specific Machine: The MACHINE variable in the local.conf file specifies the machine for the build. For this example, set the MACHINE variable to cyclone5. These configurations are used:
   https://github.com/kraj/meta-altera/blob/master/conf/machine/cyclone5.conf.

   Note

   See the "Examine Your Local Configuration File" step earlier for more information on configuring the build.

Add Your Layer to the Layer Configuration File: Before you can use a layer during a build, you must add it to your bblayers.conf file, which is found in the Build Directory conf directory.

Use the bitbake-layers add-layer command to add the layer to the configuration file:

```
$ cd poky/build

$ bitbake-layers add-layer ../meta-altera

NOTE: Starting bitbake server...

Parsing recipes: 100% |##################################################################| Time: 0:00:32

Parsing of 918 .bb files complete (0 cached, 918 parsed). 1401 targets,

123 skipped, 0 masked, 0 errors.
```

4. You can find more information on adding layers in the Adding a Layer Using the bitbake-layers Script section.

Completing these steps has added the meta-altera layer to your Yocto Project development environment and configured it to build for the cyclone5 machine.