Yocto Learning

Document Number: LPN-QMS-SHOW-00000

Revision: 0.0.0-75-g2094a49

VCS: 2094a49d

Li-Pro.Net

Li-Pro.Net — Documentation

Mar 03, 2021

Li-Pro.Net
Jena, Germany
Email: info@li-pro.net

Table of Contents:

1 Challenge **9**
- 1.1 Example: UI Toolkits 9
- 1.2 Collaboration 10
- 1.3 Market 10
- 1.4 Vendor and Contractor 11
- 1.5 Community 11
- 1.6 Business 12
- 1.7 Domains 12

2 Intentions **13**

3 Decision **15**
- 3.1 Android 15
- 3.2 Baserock 16
- 3.3 Buildroot 16
- 3.4 OpenEmbedded 16
- 3.5 OpenWrt 17
- 3.6 Reflection 17

4 About **19**
- 4.1 Governance Structure 20
- 4.2 Achievement and Standing 20
- 4.3 Offer and Support 21
- 4.4 Yocto Project Provides 22
- 4.5 Yocto Project Community 24

5 Terms, Concepts, Idioms **27**

6 Git Repo Manifest **31**
- 6.1 Platform Manifest 32
- 6.2 Platform Pipeline 35

7 Build System **37**
- 7.1 Configuration Area 38
- 7.2 Metadata Area 39

7.3 Source Area . . . 40
7.4 Package Area . . . 41
7.5 BitBake Tool Area . . . 42
7.6 Images Area . . . 43
7.7 SDK Area . . . 44

8 User Configuration **45**
8.1 `bblayers.conf` . . . 47
8.2 `local.conf` . . . 47
8.3 `setup-environment` . . . 48

9 Metadata Layers **49**
9.1 Meta Layer Types . . . 51
9.2 Meta Layer Stack . . . 52
9.3 Meta Layer Machines . . . 52
9.3.1 hydrogen.conf . . . 52
9.3.2 helium.conf . . . 53
9.4 Meta Layer Distros . . . 54
9.4.1 cedi.conf . . . 54
9.5 Meta Layer Images . . . 55
9.5.1 central-image.bb . . . 55
9.5.2 central-image.bbclass . . . 56
9.5.3 central-image-version.bbclass . . . 57
9.5.4 packagegroup-central-boot.bb . . . 58
9.5.5 packagegroup-central-tools-testapps.bb . . . 58

10 Source Files **61**
10.1 Upstream Project Releases . . . 63
10.2 Local Projects . . . 63
10.3 Source Code Manager (optional) . . . 63
10.4 Source Mirror(s) . . . 64

11 Package Feeds **65**

12 BitBake Tool **67**
12.1 Inherit . . . 67
12.2 Fetching . . . 68
12.2.1 Fetching in Recipe . . . 69
12.3 Patching . . . 69
12.3.1 Patching in Recipe . . . 70
12.4 Compilation . . . 70
12.4.1 Compilation in Recipe . . . 71
12.5 Packaging . . . 71
12.5.1 Packaging in Recipe . . . 72
12.6 Image Generation . . . 72
12.7 SDK Generation . . . 74

13 Images **75**

14 Application Development SDK **77**

15 Final Closer Look **81**

16 Read-The-Docs **83**
16.1 Documentations 83
16.2 Presentations 84
16.3 Books 85

17 Thank You **87**

Appendices **89**

A General Information **89**
A.1 License 89
A.2 Credits 97

B Glossary **99**
B.1 Common Terms 99
B.2 Programming Languages 101
B.3 Technologies 103

Acronyms **109**

Listings **111**

List of Tables **113**

List of Figures **115**

List of Equations **117**

List of Downloads **119**

List of Issues **121**

Bibliography **123**

Index **125**

SUMMARY

Abstract This document is an effort by Li-Pro.Net — Documentation and aims to improve technical understanding and usage of describe piece of tools and frameworks. *In short words:* Learning Yocto.

Involved Components

- **Git** – Repo and Manifest
- **Yocto** – Terms and Workflow
- **Bitbake** – Layer and Recipies

Audience

- System Architects
- Hard- and Software Developers
- Integrators and Testers

Status preliminary (*some mature, much in progress*)

Version 0.0.0

Release 0.0.0-75-g2094a49

Date Mar 03, 2021

Authors Li-Pro.Net — Documentation

- Stephan Linz <linz@li-pro.net>

Credits See the file CREDITS that comes with the documentation for a list of all well known contributors.

Organization Li-Pro.Net

Contact info@li-pro.net

Address Jena, Germany

PREAMBLE

Learning Yocto.

This document introduce to the Yocto framework, their intentions of use and their main focus and area of application. The format of the document is an online presentation, but is also prepared simultaneously as a minimalistic reference guide and covers the following topics:

- Domain and Tool Landscape
- Concepts, Workflow and Build System
- *Dive into the deep* – **coming soon**

For easier creation, editing and maintenance, this documentation is written using the *reStructuredText* syntax and processed using the *Sphinx* text processor. The presentation and distribution in *HTML* or *PDF* formats are only a result of processing with Sphinx from the underlying reStructuredText sources.

Thus, the methodology of "Programmable Documentation" is applied here. A Travis-CI server performs the Sphinx processing. Its results are considered to be official and can be accessed using the following links:

***Li-Pro.Net Learning Yocto* — LATEST**

- online presentation — **THE SHOW**
- online document
- printable document

Prerequisites to readers' skills

For a better understanding it is necessary that the following terms and methods are known and preferably practiced day by day:

Revision Controll Systems

- Git
- Subversion
- Mercurial
- Principals of work
- Collaboration work

Configuration and Build Tools

- Kconfig + GNU Make
- Autotools and CMake
- Qmake
- Ninja
- Source Code Patching
- Triad: `config` → `make` → `install`
- Canadian Cross: `build` → `host` → `target`

Distribution and Packaging

- Debian Packages: `*.deb`
- Red Hat Packages: `*.rpm`
- Open Packages: `*.opkg` / `*.ipkg`
- System Root: `SYSROOT`
- Destination Directory: `DESTDIR`
- Package Repository (Server)

Scripting

- Bash
- Python
- Perl (rarely)

LEGAL NOTICE

Learning Yocto.

VERSION HISTORY

Table 1: Document Revisions of “Learning Yocto”

Version	Change	Passed	Date
0.0	document skeleton, project created	Stephan Linz	2018-01-22

1 Challenge

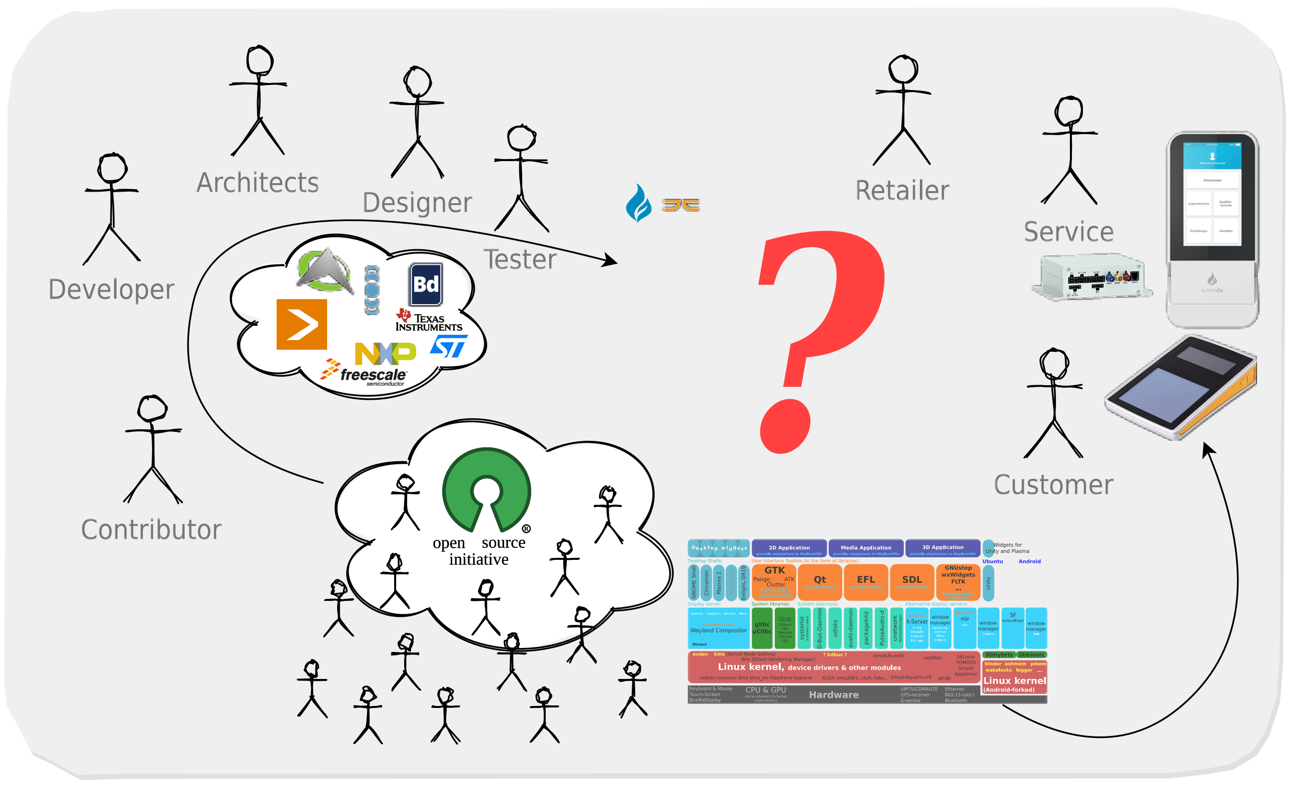


Figure 1.1: The Open Source Challenge

1.1 Example: UI Toolkits

–English Wikipedia: Linux#Desktop

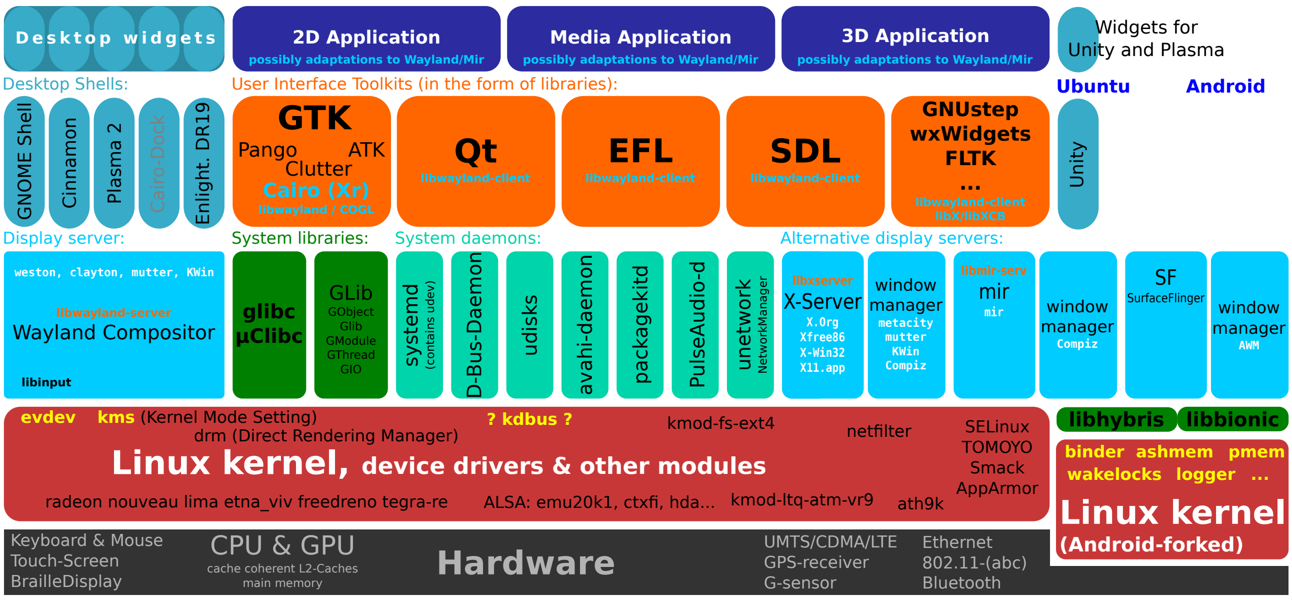


Figure 1.2: Example: UI Toolkits

1.2 Collaboration

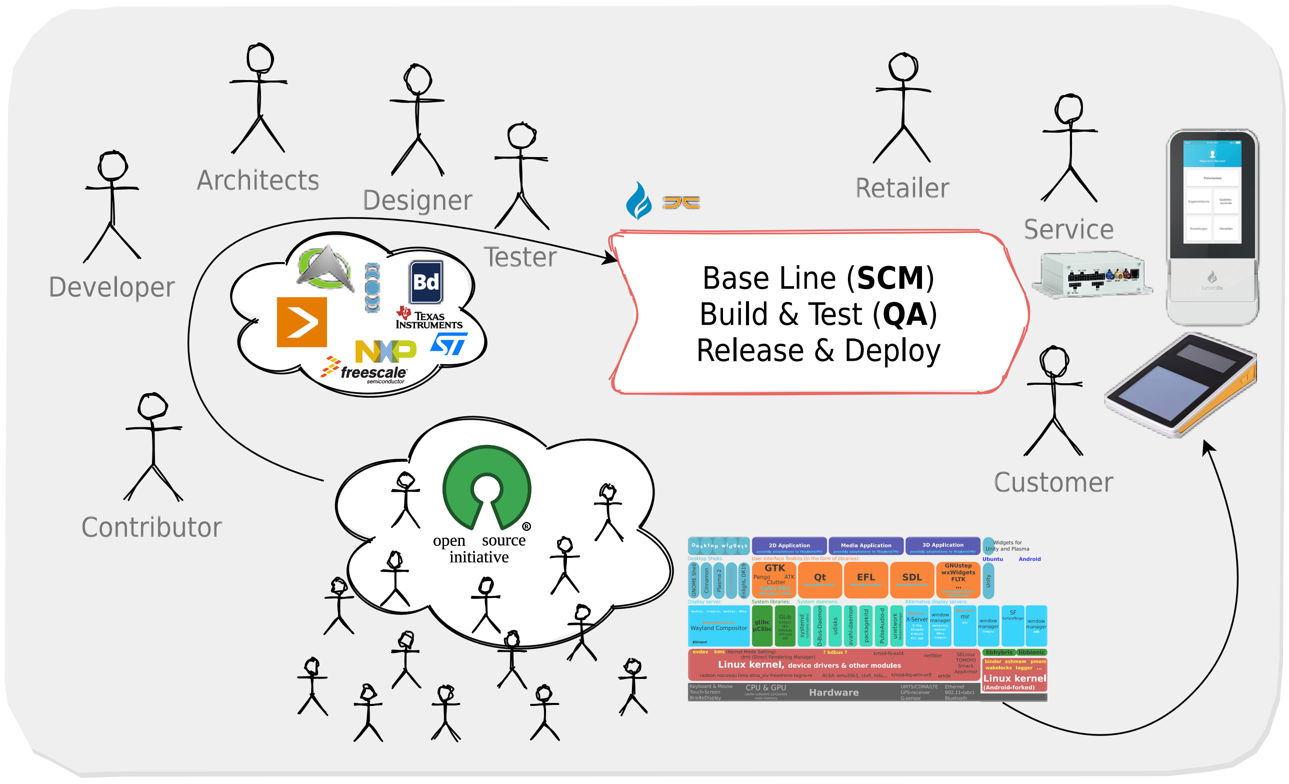


Figure 1.3: Collaboration as a Solution

1.3 Market

→ that are:

- Customer and User
- Retailer and Service

→ which are constantly looking for:

- Improvements and Bug Fixes (updates)
- Features and Extensions (back ports)
- Documentation and Tests

1.4 Vendor and Contractor

→ all participants and partners:

- Engineers
- Business Owners
- Compliance Offices

→ have to ensure results with:

- protected release base line: Software Configuration Management Plan (SCMP)
- build and test: Continuous Integration (CI)
- **DISTRIBUTION**: Continuous Deployment / Delivery (CD)

1.5 Community

→ what is their role ▯

- best efforts to **COOPERATE** with and **MUTUAL ASSISTANCE** the business
- ▯ *free of charge*: **SCALE**, **BE FAST**, pay later
- ▯ *open for business*: **SHARE WITH ALL**, no protection, survival of the fittest
- ▯ *passing grade*: **BEST FOR ALL**, acceptance
- ㊙ *secret*: kick out the Troll

1.6 Business

→ where and who am I ▯

- mostly I am a tiny candle in front of the biggest star of all
- leads to: relation of ▯ *candle* / ▯ *star*
- ask you: what is **MY BUSINESS** ▯ *candle*
- ask you: what is **THE COMMUNITY** ▯ *star*

my experience of the last 20 years:

10–40 % ▯ *candle* / **90–60 %** ▯ *star*

1.7 Domains

→ the COMMUNITY – there are we all ‼

- Market with its Suppliers and Vendors
- Contractors and their Principals

→ we *should* take over the WORKFLOWS ‼

- Infrastructure
- Frameworks
- Toolboxes

2 Intentions

Understanding the reasons behind the decision to use a specific tool for product development requires a closer look at the various perspectives of all the involved participants, from the perspective of:

1 – Product Management

which hard- and software features we have, and need for a specific product → feature sets

2 – Feature Management

how the distribution looks for a given feature set → composing

3 – Distribution Management

which packages will needed for a given distribution / composing → flavor

4 – Package Management

how to build and deploy the given list of packages and their dependencies

5 – Deployment Management

what have to share and how to distribute all the resulting artefacts

6 – QA Management (Test, Integration)

how to ensure accuracy of all the development, in parts and altogether

3 Decision

This section contains direct excerpts and quotes from publicly accessible presentations: [Str15][OMM15]. — © 2015 The Linux Foundation.

Embedded Linux Landscape

- Android
- Baserock
- Buildroot
- OpenEmbedded
- OpenWrt
- The Yocto Project
- Commercial Distributions

Reflection

- **Buildroot ↔ The Yocto Project**

3.1 Android

→ https://source.android.com/

- Great for systems with ARM-based SoCs and touch screens
- Build system and development tools by Google
- Java based system runtime environment on target
- Stable and widely accepted Java API for final applications
- Very limited adaptability for vendors, contributors or customers

3.2 Baserock

→ https://www.baserock.org/

- Targeted for embedded appliances
- Slightly pushed by the GENIVI Alliance
- Native Builds for x86, x86_64, ARMv7
- Based on Morph workflow tool
- Maintained by Codethink Ltd.

Last releases in 2016. → https://wiki.baserock.org/releases/

3.3 Buildroot

→ https://buildroot.org/

- Targeted for complete Linux root filesystem build
- Generate a cross-compilation toolchain
- Simplifies and automates the building
- uClibc or musl target library
- BusyBox command line utility applications
- Based on Kconfig and GNU Make build tool
- Promoted by french non-profit Buildroot Association

3.4 OpenEmbedded

→ https://www.openembedded.org/

- Created by merging OpenZaurus with contributions from Familiar Linux, OpenSIMPad into a common code base
- Focusing on broad hardware and architectures
- Based on BitBake build engine
- Technology code base for the Yocto Project and the Ångström Distribution

The Ångström Distribution was widely used on TI-based embedded boards like the BeagleBoard and PandaBoard and uses the Yocto Project build environment. Last releases in 2017 (2020). → https://github.com/Angstrom-distribution

3.5 OpenWrt

→ https://www.openwrt.org/

- Debuted as open source OS for embedded devices routing network traffic
- Originally created from Linksys GPL sources for their WRT54G residential gateway
- Targeted thousands of different hardware
- Buildroot-based build environment
- Headless operation with web UI
- Member of Software Freedom Conservancy (like Git, Inkscape, or Qemu)

3.6 Reflection

→ https://lwn.net/Articles/682540/ (ELCE-2016, ELCE-2018, Youtube)

	Buildroot	Yocto Project
Minimal Size:	2.2 MB	4.9 MB
Build Time:	15-45 minutes	1-2 hours (at least)
Concept:	single Makefile	layered classes
Complexity:	being simple by design, only patchable	multiple aspects of composing and QA
Composing:	multiple Kconfig file	feature driven
State Cache:	compiler only	on task level (states)

	Buildroot	Yocto Project
Rebuild:	full (will be improved)	partial per task / on changes
Meta/Configure:	fixed	sharable
Output:	no packages or SDK	images, packages, SDK, licenses, manifests
Reliance:	no QA workflow	documented QA and RM
3rd Party:	rare, some few	many (growing)
Decision:		**Yocto Project**

4 About

This section contains direct excerpts and quotes from publicly accessible presentations: [Str15][OMM15]. — © 2015 The Linux Foundation.

The Yocto Project® is an open source collaboration project that helps developers create custom Linux-based systems for embedded products, regardless of the hardware architecture. For additional information about the project, please visit: About the Yocto Project.

What is the Yocto Project?

- Open source project with a strong community
- A collection of embedded projects and tooling
  - Place for Industry to publish BSPs
  - Application Development Tools including Eclipse plug-ins and emulators
- Key project is the reference distribution build environment (Poky)
  - Complete Build System for Linux OS
  - Releases every 6 months with latest (but stable) kernel (LTSI), toolchain, and package versions
  - Full documentation representative of a consistent system

Yocto Project

1. It's not an embedded Linux distribution – it creates a custom one for you.
2. It is an Ecosystem.

The Yocto Project combines the convenience of a ready-to-run Linux Distribution with the flexibility of a custom Linux operating system stack. [Str15][OMM15] — © 2015 The Linux Foundation and Yocto Project.

4.1 Governance Structure

→ Ecosystem and Showcase

- based on meritocracy (leadership by demonstrated achievement)
- managed by its chief architect, **Richard Purdie**, a Linux Foundation fellow
- remain independent of any one of its member organizations

Who is the Yocto Project?

Advisory Board and Technical Leadership

- Organized under the Linux Foundation
- Individual Developers
- Embedded Hardware Companies
- Semiconductor Manufacturers
- Embedded Operating System Vendors
- OpenEmbedded (OE) / LTSI Community

Linux Foundation

The Yocto Project is a lab workgroup of the Linux Foundation, who owns its trademark.

4.2 Achievement and Standing

- open source collaboration project
- founded in 2010 as a collaboration among many hardware manufacturers, open-source operating systems vendors and electronics companies
- help you create custom Linux-based systems for embedded products
- regardless of the hardware architecture
- provides templates, tools and methods (Best-Practice)

What the Yocto Project Provides

- The industry needed a common build system and core technology
  - BitBake and OpenEmbedded Core = OE (OpenEmbedded) build system
- The benefit of doing so is:
  - Designed for the long term
  - Designed for embedded
  - Transparent Upstream changes
  - Vibrant Developer Community

Best Practices

Less time spent on things which don't make money (build system, core Linux components)

More time spent on things which do make money (app & product development, ...)

4.3 Offer and Support

- resources and information for new and experienced users
- core system component recipes by OE
- example code to demonstrate its capabilities
- include the Yocto Project Linux kernel
- covers several build profiles across multiple architectures: ARM, PPC, x86, and more
- BSP (Board Support Package) layers for customer or vendor specific platform support
- BSP layers follow a predetermined and standardized format

Why Should a Developer Care?

- Build a complete Linux system – from source – in about an hour (about 90 minutes with X)
  - Multiple cores (i.e. quad i7)
  - Lots of RAM (i.e. 16 GB of ram or more)
  - Fast disk (RAID, SSD, etc...)
- Start with a validated collection of software (toolchain, kernel, user space)
- Blueprints to get you started quickly and that you can customize for your own needs
- We distinguish app developers from system developers and we support both

- Access to a great collection of app developer tools (performance, debug, power analysis, Eclipse)
- Supports all major embedded architectures
  - x86, x86-64, ARM, PPC, MIPS
  - MIPS64, ARM Arch 64, PPC64
  - and more exotic like MicroBlaze or Nios
- Advanced kernel development tools
- Layer model encourages modular development, reuse, and easy customizations
- Compatibility program that is used to encourage interoperability and best practices

4.4 Yocto Project Provides

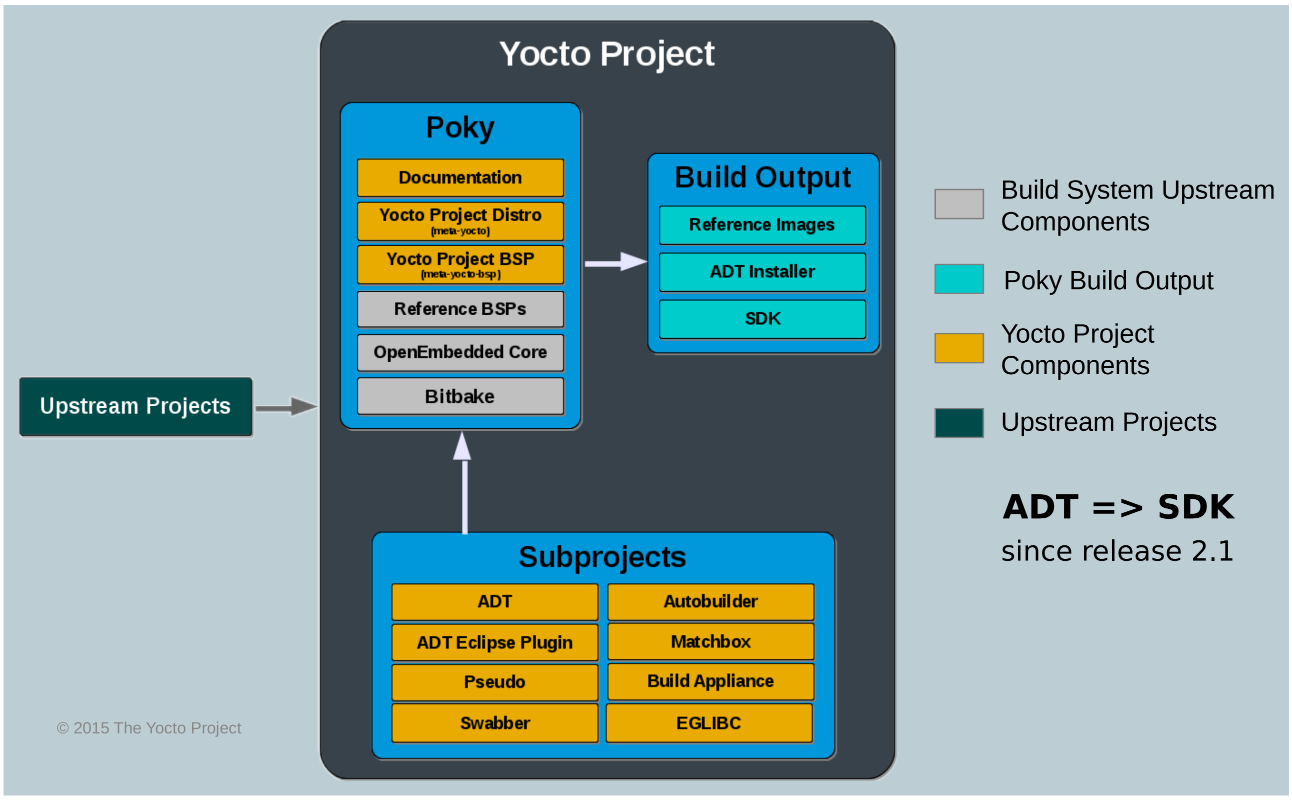


Figure 4.1: Yocto Project Deliveries [OMM15]

- embedded tools
- best practices
- reference implementation

Yocto Project and OpenEmbedded

OpenEmbedded

- Created by merging the work of the OpenZaurus project with contributions from other projects such as Familiar Linux and OpenSIMpad into a common code base
- Community project focusing on broad hardware and architectures
- Large library of recipes to cross-compile over 1000 packages
- Switched from flat meta-data architecture (OpenEmbedded Classic) to layered architecture based on OpenEmbedded Core layer, which is in common with the Yocto Project and the Ångström Distribution

Yocto Project

- Family of projects for developing Linux-based devices
- Self-contained build environment providing tools and blueprints for building Linux OS stacks
- Supported by silicon vendors, OSVs (also providing commercial support), open source projects for hardware and software, electronics companies
- Standardized components with compliance program
- Focused on tooling and maintenance, major release every 6 months

Why not just use OpenEmbedded?

OpenEmbedded

- OpenEmbedded is an Open Source Project providing a Build Framework for Embedded Linux Systems
  - Not a reference distribution
  - Designed to be the foundation for others
  - Cutting-edge technologies and software packages

Yocto Project

- The Yocto Project is focused on enabling Commercial Product Development
  - Provides a reference distribution policy and root file system blueprints
  - Co-maintains OpenEmbedded components and improves their quality
  - Provides additional tooling such as Autobuilder, QA Tests
  - Provides tools for application development such as ADT and Eclipse Plugin

4.5 Yocto Project Community

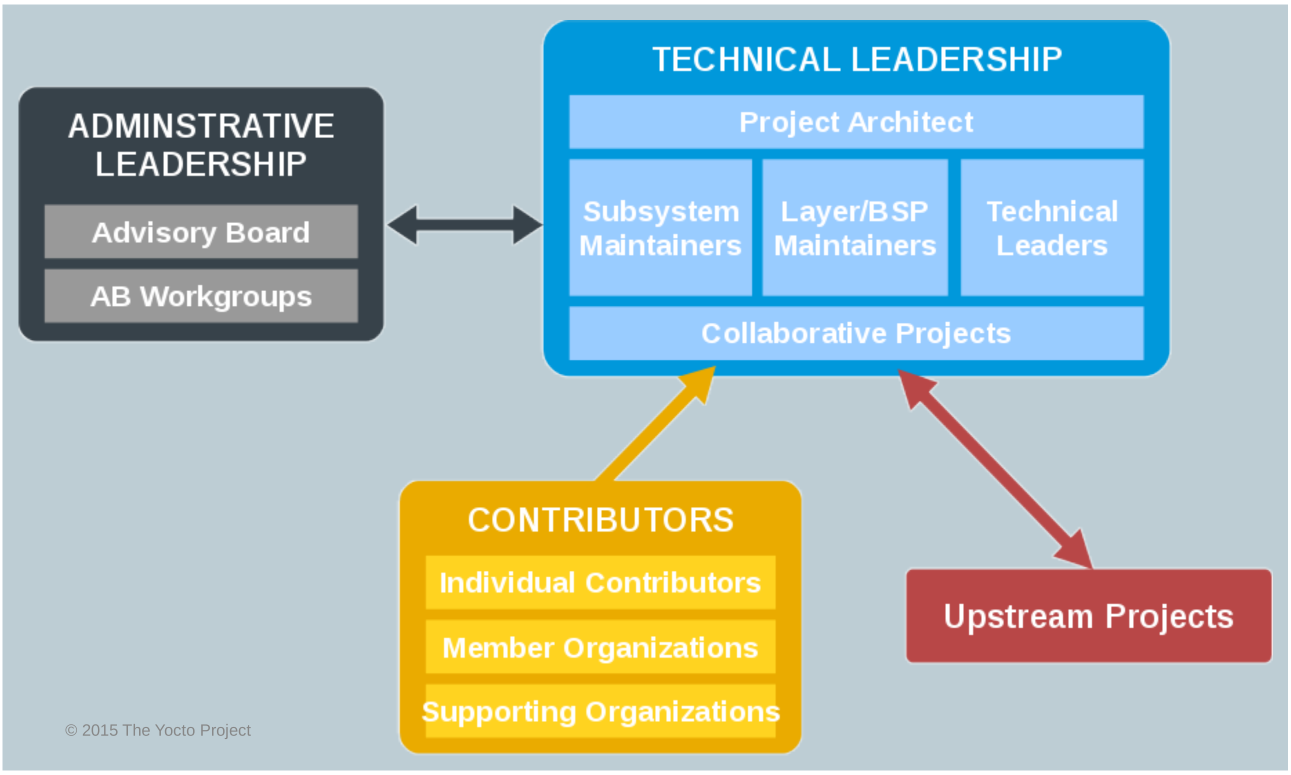


Figure 4.2: The Yocto Project Community [OMM15]

- Linux Foundation: chief architect
- OpenEmbedded: technologies and software
- silicon vendors and electronics companies

The Yocto Project Ecosystem

Product Showcase

- Hardware Platforms
- Distributions – Open Source and Commercial
- Projects – Open Source Project using the Yocto Project

Participants

- Organizations who participate in the Yocto Project Compliance Program
- They also support the project through contributions and engineering resources

Member Organizations

- Organizations who provide the administrative leadership of the Yocto Project
- Their support includes membership dues for infrastructure etc. and engineering resources

- Members of the Yocto Project Advisory Board

Supporting Organizations

- Organizations who support the Yocto Project through contributions, product development, etc.

Yocto Project Branding and Compliance Program

Goals

- Strengthen the Yocto Project through a consistent branding.
- Provide recognition to participating organizations.
- Reduce fragmentation in the embedded Linux market by encouraging collaborative development of a common set of tools, standards, and practices and ensure that these tools, standards, and practices are architecturally independent as much as possible.

Yocto Project Participant

- Organizations and entities who use and support the Yocto Project publicly.
- Open to open source projects, non-profit organizations, small companies, and Yocto Project member organizations.

Yocto Project Compatible

- Products, BSPs, OpenEmbedded-compatible layers and other open source software projects that are built and work with the Yocto Project.
- These products and components must be submitted by open source projects, non-profit entities, or Yocto Project member organizations.

5 Terms, Concepts, Idioms

Following is a slightly extended list of terms, concepts and definitions users new to the Yocto Project development environment might find helpful. While some of these terms are universal, the list includes them just in such case. A more or less completed version of the list can be found on the following website: Yocto Project Terms

Terms overall

1. **Git Repo** (`repo`): The collection manager that Google has built on top of Git to help manage multiple Git repositories, does the uploads to Gerrit and automates the Android system image development workflow. The **`repo`** command is an executable *Python* script that you can put anywhere in your path. → https://gerrit.googlesource.com/git-repo/
2. **BitBake**: The **Task** executor and scheduler used by the **Build System** to build **Images**. The **`bitbake`** command is part of an fancy *Python* and Bash environment around the **BitBake** project. → BitBake, or Yocto Components and Tools → BitBake, or BitBake User Manual
3. **Metadata**: The files that **BitBake** parses when building an **Images**. In general, **Metadata** provides **Recipes**, **Classes** and **Configuration Files**. → Metadata, or BitBake User Manual → Overview → Introduction
4. **OE-Core**: A core set of **Metadata** originating with OpenEmbedded that is shared between OE and the Yocto Project. → OpenEmbedded-Core (OE-Core), or Yocto Components and Tools → OE-Core, or OE Wikipedia
5. **Poky**: The reference distribution of the Yocto Project. It contains the **Build System** (**BitBake** and **OE-Core**) as well as a set of **Metadata** to get you started building your own distro (distribution). → Poky, or Yocto Components and Tools → Poky, or Yocto Reference Distribution

Terms internal

1. **Layer**: A collection of **Recipes** representing the core, a Board Support Package (BSP), an application stack, or any other 3rd party contributions. **Layers** are hierarchical in their ability to override previous specifications. → Layer, or in Concept
   - list of Yocto Compatible Layers
   - list of OpenEmbedded Contributed Layers
2. **Image**: An artifact of the **BitBake** build process given a collection of **Recipes** and related **Metadata**. **Images** are the binary output that run on specific target architecture. → Image, or in Concept, or in Reference
3. **Package**: Refers to a **Recipe**'s packaged output produced by **BitBake** (i.e. a *"baked recipe"*). A **Package** is generally the compiled binaries produced from the **Recipe**'s sources. You *"bake"* any *"bin"* by running it through **BitBake**. → Package
4. **Package Groups**: Arbitrary group of software **Recipes**. You use **Package Groups** to hold **Recipes** that, when built, usually accomplish a single **Task**. For example, **Package Groups** could contain the **Recipes** for a company's proprietary or value-add software. → Package Groups
5. **Task**: A unit of execution for **BitBake** (e.g. do_fetch, do_patch, do_compile, and so forth). → Task, or in Reference

Metadata

1. **Classes**: Files with the `.bbclass` file extension that provided for logic encapsulation and inheritance so that commonly used patterns can be defined once and then easily used in multiple **Recipes**. → Classes, or in Concept, or in Reference
2. **Recipes**: Files with the `.bb` file extension have a set of instructions for building **Packages** or **Package Groups** or **Images** and describes where you get source code, which patches to apply, how to configure the source, how to compile it and so on. It also describe dependencies for libraries or for other **Recipes**. **Recipes** represent the logical unit of execution. → Recipe, or in Concept
3. **Append Files**: Files with the `.bbappend` file extension that append build information to a **Recipe**. The **Build System** respects every **Append file** to have a corresponding **Recipe** (`.bb`) file. → Append Files
4. **Configuration Files**: Configuration information in various → Configuration File, or in Concept

Features

Features provide a mechanism for:

1. working out which **Packages** or **Package Groups** should be included in the generated **Image**, typically configured in **Image Recipes** and their related **Classes** → Image Features
2. distributions **Configuration Files** can select which features they want to support → Distro Features
3. machine **Configuration Files** specifies the hardware functions that will be available for a given machine → Machine Features

4. backward and inter-layer compatibility for special (historical) cases → Feature Backfilling

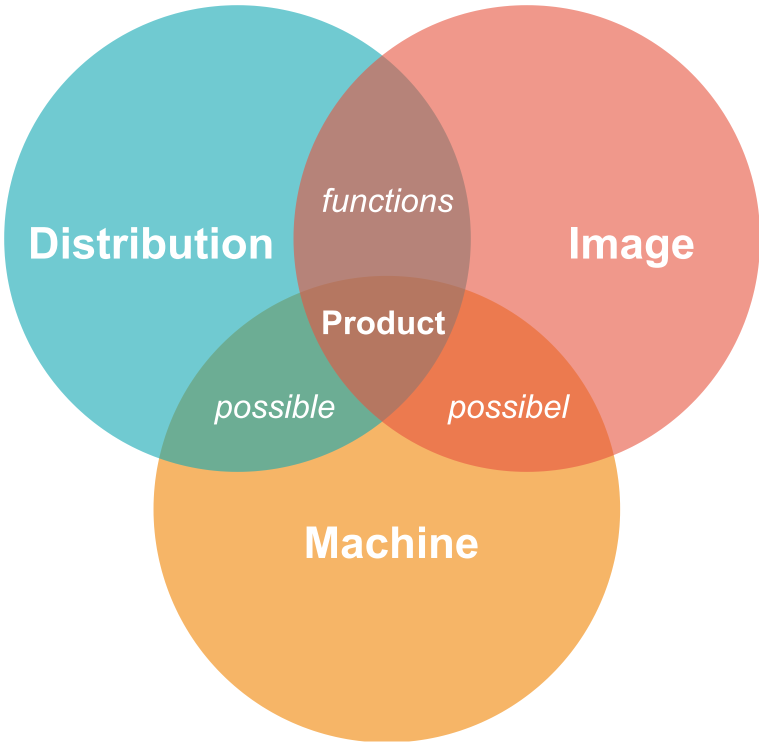


Figure 5.1: Yocto Build System Features and their interaction

Idioms

1. **Build System**: The build system specific to the Yocto Project based on the project **Poky**, which uses **BitBake** as the **Task** executor.
2. **Source Directory**: Refers to the directory structure created as a result of creating a local copy of the **Poky** and all other Git repositories as referred by the **Git Repo** manifest. It contains **BitBake**, Documentation, **Metadata** and other files that all support the Yocto Project. Consequently, you must have the **Source Directory** in place on your development system in order to do any development using the Yocto Project. → Source Directory
3. **Build Directory**: This term refers to the area used by the **Build System** for downloads, builds and deployments. → Build Directory
4. **Cross-Development Toolchain**: A collection of cross-compilers, linkers, debuggers, libraries and utilities that run on one architecture and allow you to develop software for a different, or targeted, architecture. → Cross-Development Toolchain
5. **Board Support Package (BSP)**: A group of drivers, definitions, and other components that provide support for a specific hardware configuration. → Board Support Package (BSP), or Yocto Project Board Support Package Developer's Guide and Yocto Project Linux Kernel Development Manual

Conclusion

---

the following applies and is unavoidable

Git Repo: determines the code base line

Layers: are essential and able to overlay

Classes: inheritable but not extensible

Recipes: extensible but not inheritable

Features: determines the final product

---

6 Git Repo Manifest

A product firmware stack will developed locally and pushed to different in-house or external repositories at irregular intervals. A tool to unifies all that repositories as necessary, performs uploads and automates the pull and sync process will be needed.

The **`repo`** tool and a meta Git repository called *manifest* or such as in our case *platform repo* make it possible to get all the code needed to build your own Yocto *source* and *build directory*.

excerpts and quotes from https://source.android.com/setup/develop#repo — © 2008–2021 Google LLC.

Git

handle projects that are distributed over multiple Git repositories

Repo

unifies Git repositories, performs uploads to Gerrit, and automates the workflow

Gerrit

code review system for projects that use Git

6.1 Platform Manifest

This section contains direct excerpts and quotes from publicly accessible documentation: Manifest Format — © 2008–2021 Google LLC.

https://github.com/lipro-yocto/lpn-central-repo

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>

  <default sync-j="4" revision="master"/>

  <remote fetch="https://git.yoctoproject.org/git" name="yocto"/>
  <remote fetch="https://github.com/openembedded" name="oe"/>
  <remote fetch="https://github.com/meta-qt5" name="qt5"/>
  <remote fetch="https://github.com/lipro-yocto" name="lpy"/>

  <project remote="yocto" name="poky" path="sources/poky"/>
  <project remote="yocto" name="meta-mingw" path="sources/yocto/meta-mingw"/>
  <project remote="oe" name="meta-openembedded" path="sources/openembedded"/>
  <project remote="qt5" name="meta-qt5" path="sources/qt/meta-qt5" />

  <project remote="lpy" name="lpn-central" path="sources/central">
    <copyfile dest="setup-environment" src="scripts/setup-environment"/>
  </project>

</manifest>
```

→ https://gerrit.googlesource.com/git-repo/+/master/docs

Manifest Format, Configuration, Hooks

Element *manifest*

The root element of the file.

Element *include*

This element provides the capability of including another manifest file into the originating manifest. Normal rules apply for the target manifest to include – it must be a usable manifest on its own.

Attribute *name* The name of the manifest file to include, specified relative to the manifest repository's root.

Element *default*

At most one default element may be specified. Its remote and revision attributes are used when a project element does not specify its own remote or revision attribute.

Attribute *remote* See *project* → *remote*. Name of a previously defined remote element. Project elements lacking a remote attribute of their own will use this remote.

Attribute *revision* See *remote* → *revision*. Project elements lacking their own revision attribute will use this revision.

Attribute *sync-j* Number of parallel jobs to use when synching.

Attribute *sync-s* Set to true to also sync sub-projects (Git submodules).

Element *remote*

One or more remote elements may be specified. Each remote element specifies a Git URL shared by one or more projects and (optionally) the Gerrit review server those projects upload changes through.

Attribute *name* A short name unique to this manifest file. The name specified here is used as the remote name in each project's .git/config, and is therefore automatically available to commands like **`git fetch`**, **`git remote`**, **`git pull`** and **`git push`**.

Attribute *alias* The alias, if specified, is used to override *name* to be set as the remote name in each project's `.git/config`. Its value can be duplicated while attribute name has to be unique in the manifest file. This helps each project to be able to have same remote name which actually points to different remote URL.

Attribute *fetch* The Git URL prefix for all projects which use this remote. Each project's name is appended to this prefix to form the actual URL used to clone the project.

Attribute *pushurl* The Git "push" URL prefix for all projects which use this remote. Each project's name is appended to this prefix to form the actual URL used to **`git push`** the project. This attribute is optional; if not specified then **`git push`** will use the same URL as the *fetch* attribute.

Attribute *review* Hostname of the Gerrit server where reviews are uploaded to by **`repo upload`**. This attribute is optional; if not specified then **`repo upload`** will not function.

Attribute *revision* Name of a Git branch or specific reference (e.g. `master` or `refs/heads/master`, and also possible `refs/tags/1.0.0`). Tags and/or explicit SHA-1s should work in theory, but have not been extensively tested. Remotes with their own revision will override the *default* → *revision*.

Element *project*

One or more project elements may be specified. Each element describes a single Git repository to be cloned into the repo client workspace. You may specify Git submodules by creating a nested project. Git submodules will be automatically recognized and inherit their parent's attributes, but those may be overridden by an explicitly specified project element.

Attribute *name* A unique name for this project. The project's name is appended onto its remote's *fetch* URL to generate the actual URL to configure the Git remote with. The URL gets formed as:

```
${remote_fetch}/${project_name}.git
```

Attribute *path* An optional path relative to the top directory of the repo client where the Git working directory for this project should be placed. If not supplied the project *name* is used. If the project has a parent element, its path will be prefixed by the parent's.

Attribute *remote* Name of a previously defined *remote* element. If not supplied the remote given by the *default* element is used.

Attribute *revision* Name of the Git branch the manifest wants to track for this project. See *remote* → *revision*. If not supplied the revision given by the *remote* element is used if applicable, else the *default* element is used.

Attribute *sync-s* Set to true to also sync sub-projects (Git submodules).

Element *copyfile*

Zero or more copyfile elements may be specified as children of a *project* element. Each element describes a source-destination-pair of files; the *src* file will be copied to the *dest* place during **`repo sync`** command.

Copying from paths outside of the project or to paths outside of the repo client is not allowed. Directories or symlinks are not allowed. Intermediate paths must not be symlinks either.

Attribute *src* The source file; is project relative and must be a file.

Attribute *dest* The destination file; is relative to the top of the tree and must be a file. Parent directories will be automatically created if missing.

Element *linkfile*

It's just like *copyfile* and runs at the same time as *copyfile* but instead of copying it creates a symlink.

The symlink is created at *dest* (relative to the top of the tree) and points to the path specified by *src* which is a path in the project. Parent directories of *dest* will be automatically created if missing. The symlink target may be a file or directory, but it may not point outside of the repo client.

6.2 Platform Pipeline

This section contains direct excerpts and quotes from publicly accessible documentation: Revision Control of your Embedded Linux System — © 2019 Ville Baillie.

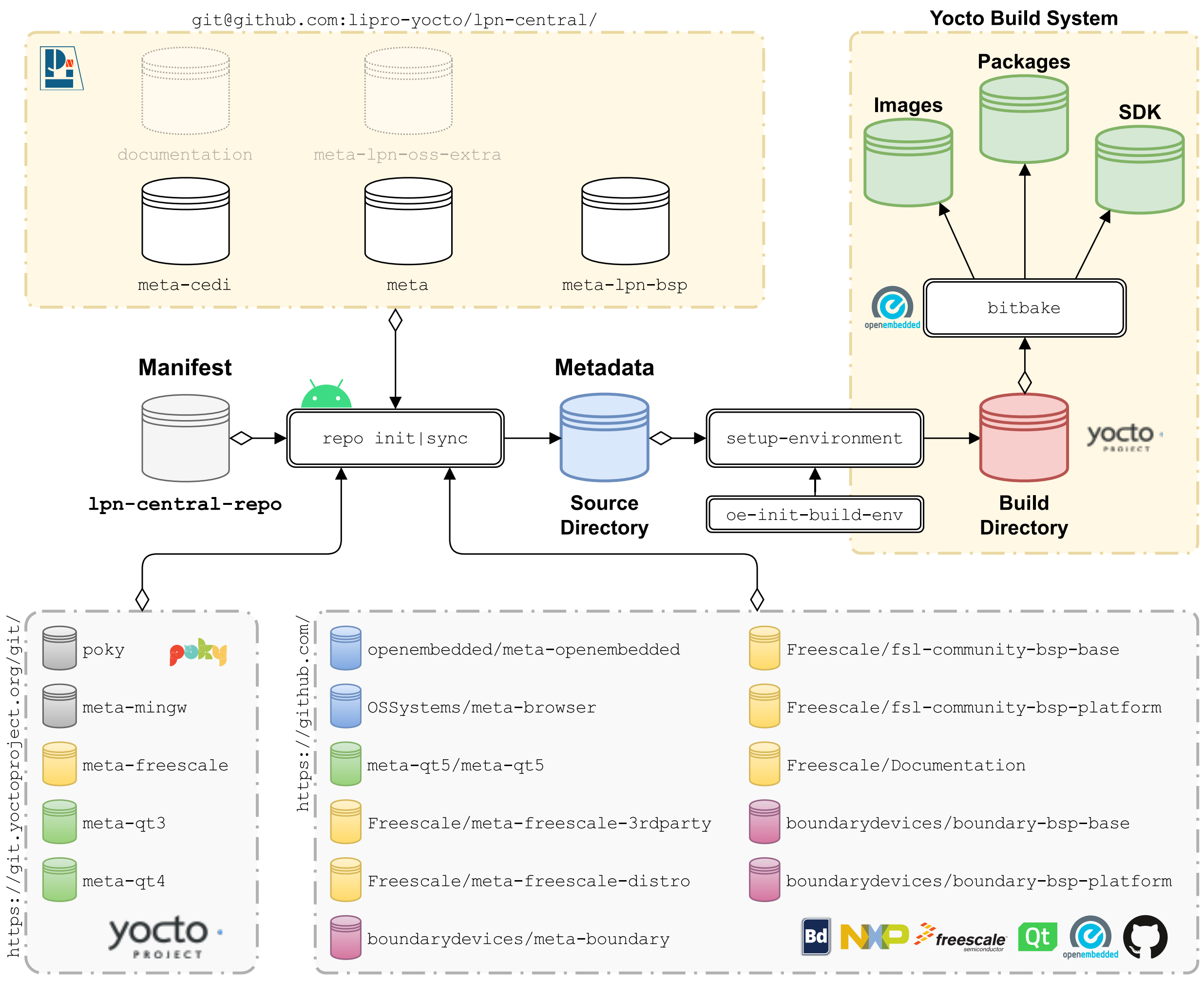


Figure 6.1: Revision Control of your Embedded Linux System using Git repo

Revision Control of your Embedded Linux System

Using Yocto you'll need to juggle several repositories at once. You'll have your "poky" repository (typically you'll get this from https://git.yoctoproject.org/git/poky), then you'll have your OpenEmbedded layer (usually retrieved from https://github.com/openembedded/meta-openembedded.git).

In addition, you'll also have your SoC and SoM specific layers, and then you'll probably have your own project specific layer. So you may have anywhere up to 10 or possibly more repositories to handle.

Earlier in this series, a short introduction to the tool named **repo** was given. This tool allows multiple git repositories to be handled together as a cohesive whole. At its heart is the idea of a manifest, which is simply a list of git repositories and commit IDs (or tags or branches), and how to arrange them on your local filesystem. The **repo** comes in two parts: One is the **repo** launcher you download and install. It's a Python script that communicates with the second part and knows how to initialize a checkout and can download the second part, the full **repo** tool included in a source code checkout. To install **repo**, e.g. from the `lipro-yocto` setup:

```
mkdir -p ~/.local/bin
curl https://raw.githubusercontent.com/lipro-yocto/git-repo/lpn-launcher/repo > \
     ~/.local/bin/repo
chmod a+x ~/.local/bin/repo
sha1sum ~/.local/bin/repo
```

For **repo launcher version 2.12.2.2021.2.20**, the SHA-1 checksum for **repo** is **93cab6406f072e78874dac1d891427c84189f0b6**.

Optionally verify the launcher matches the correct signature:

```
gpg --recv-key 5BA1FE49FB5F4F60C974D991579B34AFDE6AB439
curl https://raw.githubusercontent.com/lipro-yocto/git-repo/lpn-launcher/repo.asc | \
     gpg --verify - ~/.local/bin/repo
```

Alternatively, of course, the original can also be downloaded and installed directly from Google.

Basic repo usage

You need to create a repository to hold your manifest file (I know another repository) is the last thing you need. But this can be on your local machine if you wish, at least to begin with.

Once this is done, you should find a folder in which you wish to keep all your sources, and run the following, e.g. from the `lipro-yocto` setup:

```
mkdir lpn-central-bsp
cd lpn-central-bsp
repo init --manifest-url=https://github.com/lipro-yocto/lpn-central-repo \
          --manifest-branch=master --manifest-name=default.xml
```

This sets up the current directory as the working directory for this repo manifest. Now you can run:

```
repo sync
```

...which will fetch all the repositories into the local directory. If you want to see if everything is as it should be (no changes in the repositories and branches with respect to their upstream brothers) you can run:

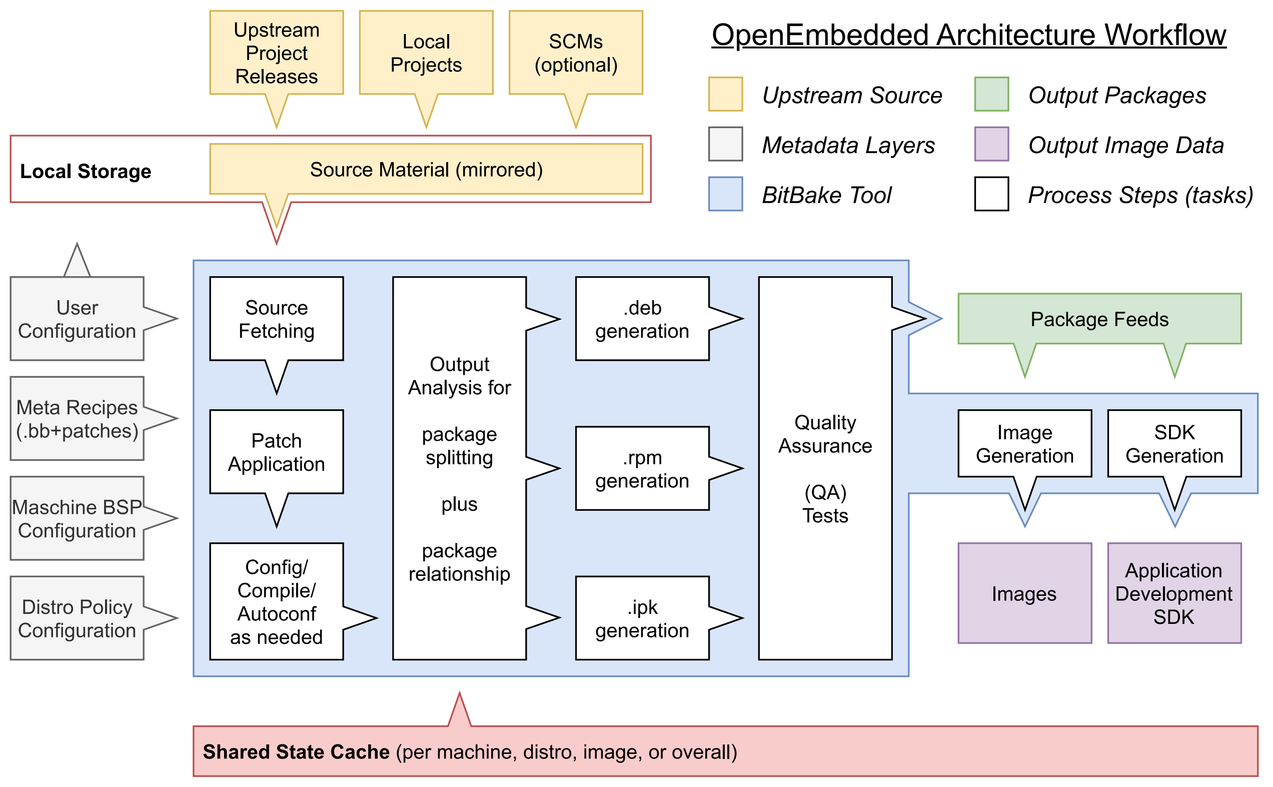
```
repo status
```

...and if you want to update everything to the manifest discarding all changes you can run

```
repo sync --detach
```

These tools should give you enough get starting and going with **repo** and Embedded Linux development. The original project page has a more comprehensive command line documentation and should always be consulted if there are any questions.

7 Build System



Upstream Project Releases
Local Projects
SCMs (optional)
OpenEmbedded Architecture Workflow
Upstream Source
Output Packages
Metadata Layers
Output Image Data
BitBake Tool
Process Steps (tasks)
Local Storage
Source Material (mirrored)
User Configuration
Meta Recipes (.bb+patches)
Maschine BSP Configuration
Distro Policy Configuration
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationship
.deb generation
.rpm generation
.ipk generation
Quality Assurance (QA) Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK
Shared State Cache (per machine, distro, image, or overall)

Figure 7.1: OpenEmbedded Build System Concepts as part of Yocto Build System Workflow

In general, the build's workflow consists of several functional areas:

User Configuration Setup values you can use to control the build process.

Metadata Layers Various layers that provide software, machine, and distro metadata.

Source Files Upstream releases, local projects, and SCMS (Software Configuration Management System).

Package Feeds Directories containing output packages (RPM, DEB or IPK), which are subsequently used in the construction of an image or Software Development Kit (SDK), produced by the build system. These feeds can also be copied and shared using a web server or other means to facilitate extending or updating existing images on devices at runtime if runtime package management is enabled.

BitBake Tool Processes under the control of BitBake. This block expands on how BitBake fetches source, applies patches, completes compilation, analyzes output for package generation, creates and tests packages, generates images, and generates cross-development tools.

Images Images produced by the build system workflow.

Application Development SDK Cross-development tools that are produced along with an image or separately with BitBake.

7.1 Configuration Area

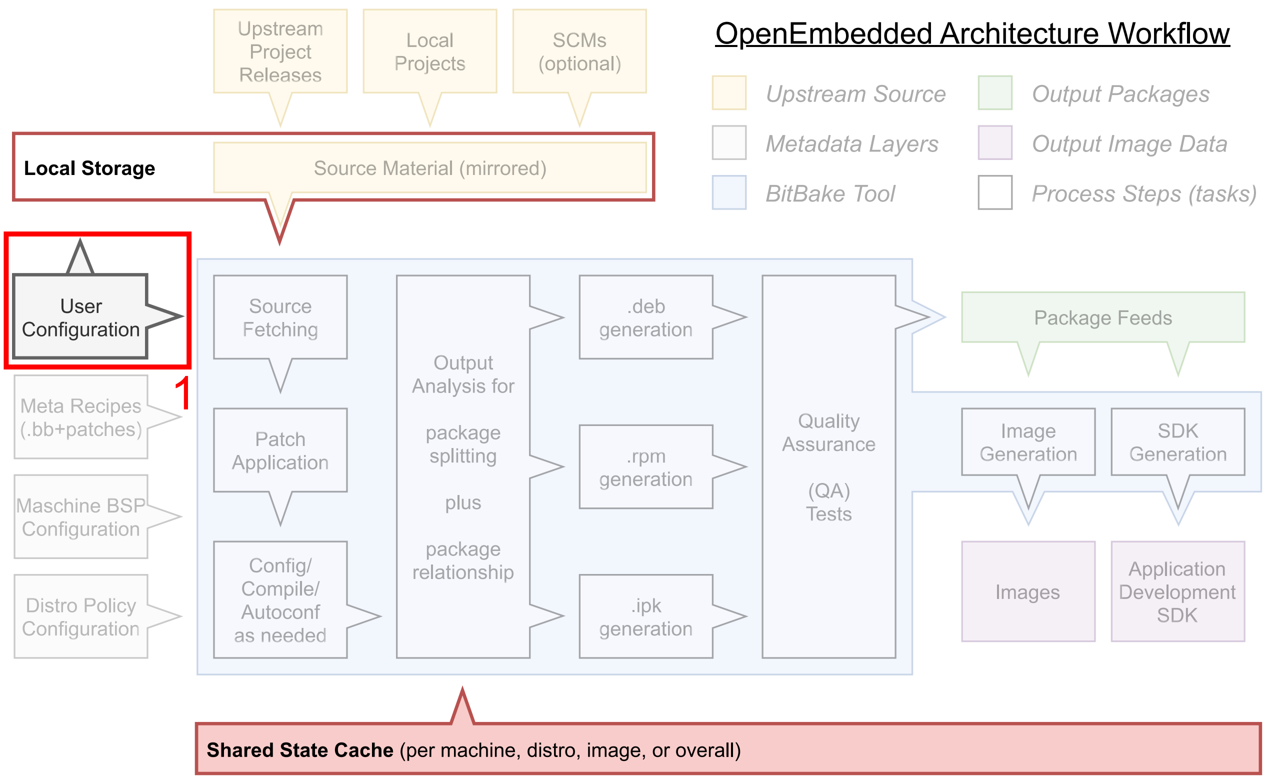

Upstream Project Releases
Local Projects
SCMs (optional)
OpenEmbedded Architecture Workflow
Upstream Source
Output Packages
Metadata Layers
Output Image Data
BitBake Tool
Process Steps (tasks)
Local Storage
Source Material (mirrored)
User Configuration
1
Meta Recipes (.bb+patches)
Maschine BSP Configuration
Distro Policy Configuration
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationship
.deb generation
.rpm generation
.ipk generation
Quality Assurance (QA) Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK
Shared State Cache (per machine, distro, image, or overall)

Figure 7.2: User Configuration as part of Yocto Build System Workflow

User configuration helps define the build. Through user configuration, you can tell BitBake the target architecture for which you are building the image, where to store downloaded source, and other build properties. See *User Configuration* (page 45) for more details.

7.2 Metadata Area

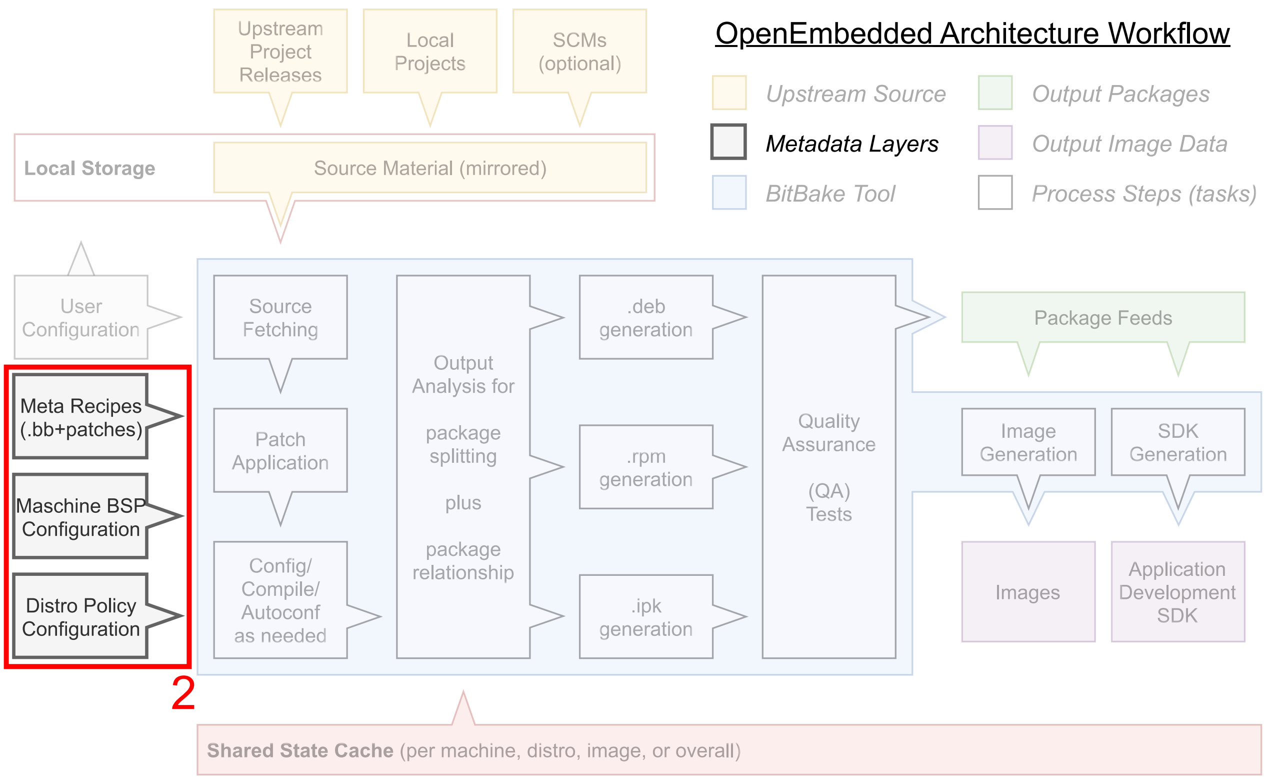

OpenEmbedded Architecture Workflow
Upstream Source
Metadata Layers
BitBake Tool
Output Packages
Output Image Data
Process Steps (tasks)
Upstream Project Releases
Local Projects
SCMs (optional)
Local Storage
Source Material (mirrored)
User Configuration
Meta Recipes (.bb+patches)
Maschine BSP Configuration
Distro Policy Configuration
2
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationship
.deb generation
.rpm generation
.ipk generation
Quality Assurance (QA) Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK
Shared State Cache (per machine, distro, image, or overall)

Figure 7.3: Metadata Layers as part of Yocto Build System Workflow

The previous area will be used to define BitBake's global behavior. This area takes a closer look at the layers the build system uses to further control the build. These layers provide Metadata for the software, machine, and policies. See *Metadata Layers* (page 49) for more details.

7.3 Source Area

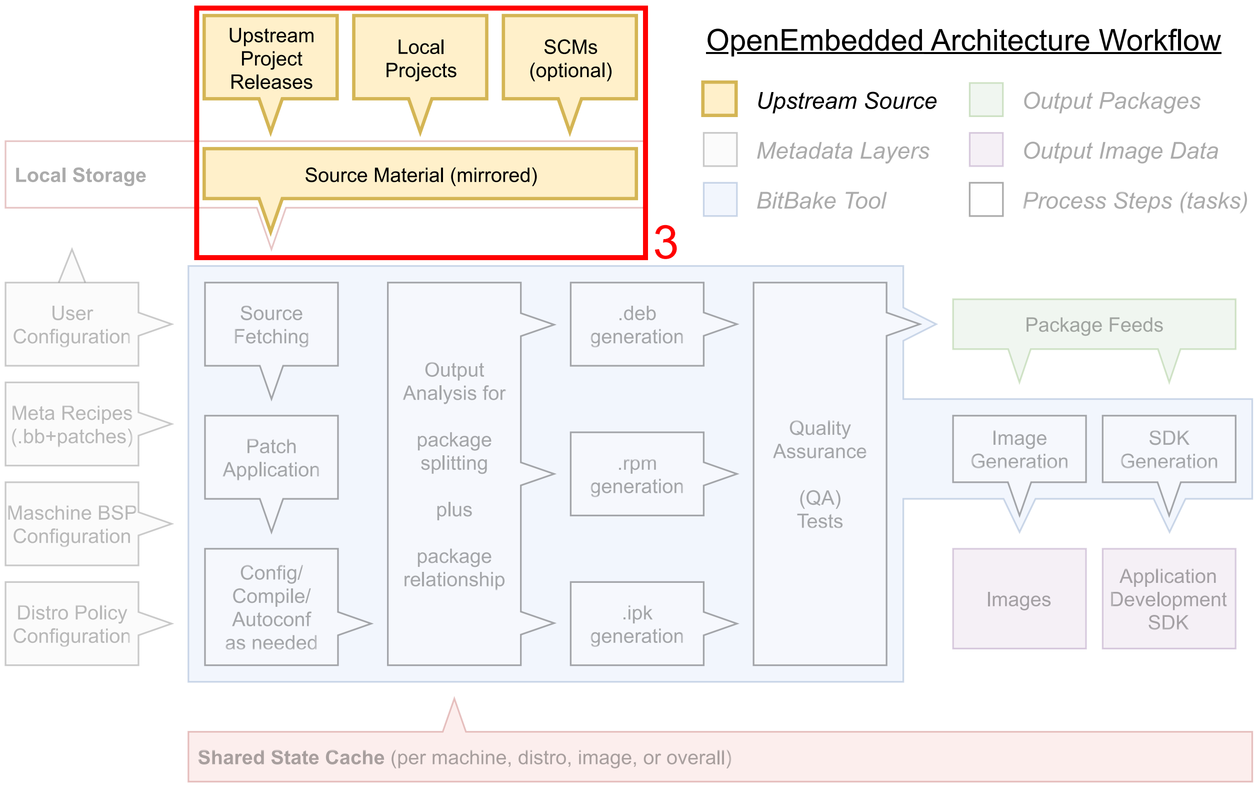

Upstream Project Releases
Local Projects
SCMs (optional)
Local Storage
Source Material (mirrored)
3
OpenEmbedded Architecture Workflow
Upstream Source
Output Packages
Metadata Layers
Output Image Data
BitBake Tool
Process Steps (tasks)
User Configuration
Meta Recipes (.bb+patches)
Maschine BSP Configuration
Distro Policy Configuration
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationship
.deb generation
.rpm generation
.ipk generation
Quality Assurance (QA) Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK
Shared State Cache (per machine, distro, image, or overall)

Figure 7.4: Sources as part of Yocto Build System Workflow

In order for the OpenEmbedded build system to create an image or any target, it must be able to access source files. The general workflow figure represents source files using the "Upstream Project Releases", "Local Projects", and "SCMS (optional)" boxes. The figure represents mirrors, which also play a role in locating source files, with the "Source Materials" box. See *Source Files* (page 61) for more details.

7.4 Package Area

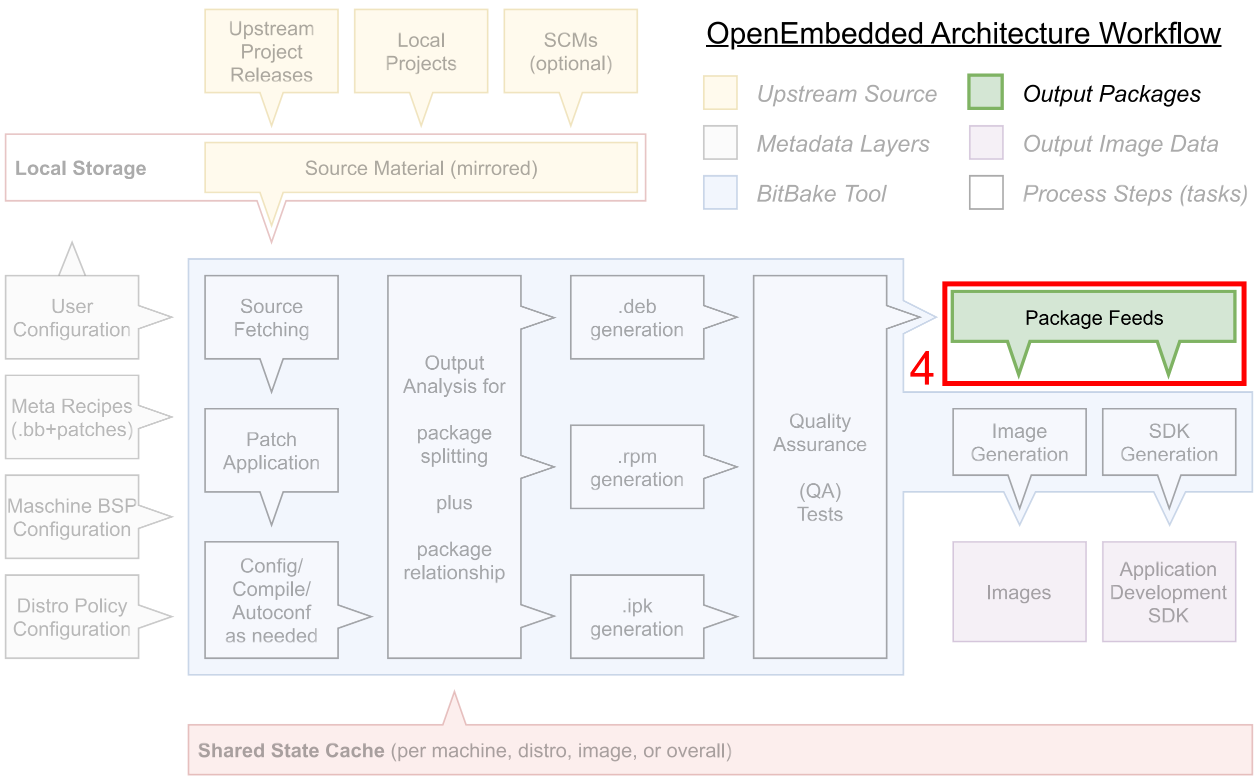

OpenEmbedded Architecture Workflow
Upstream Source
Output Packages
Metadata Layers
Output Image Data
BitBake Tool
Process Steps (tasks)
Upstream Project Releases
Local Projects
SCMs (optional)
Local Storage
Source Material (mirrored)
User Configuration
Meta Recipes (.bb+patches)
Maschine BSP Configuration
Distro Policy Configuration
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationship
.deb generation
.rpm generation
.ipk generation
Quality Assurance (QA) Tests
4
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK
Shared State Cache (per machine, distro, image, or overall)

Figure 7.5: Package Feeds as part of Yocto Build System Workflow

When the OpenEmbedded build system generates an image or an SDK, it gets the packages from a package feed area located in the Build Directory. The general workflow figure shows this package feeds area in the upper-right corner. See *Package Feeds* (page 65) for more details.

7.5 BitBake Tool Area

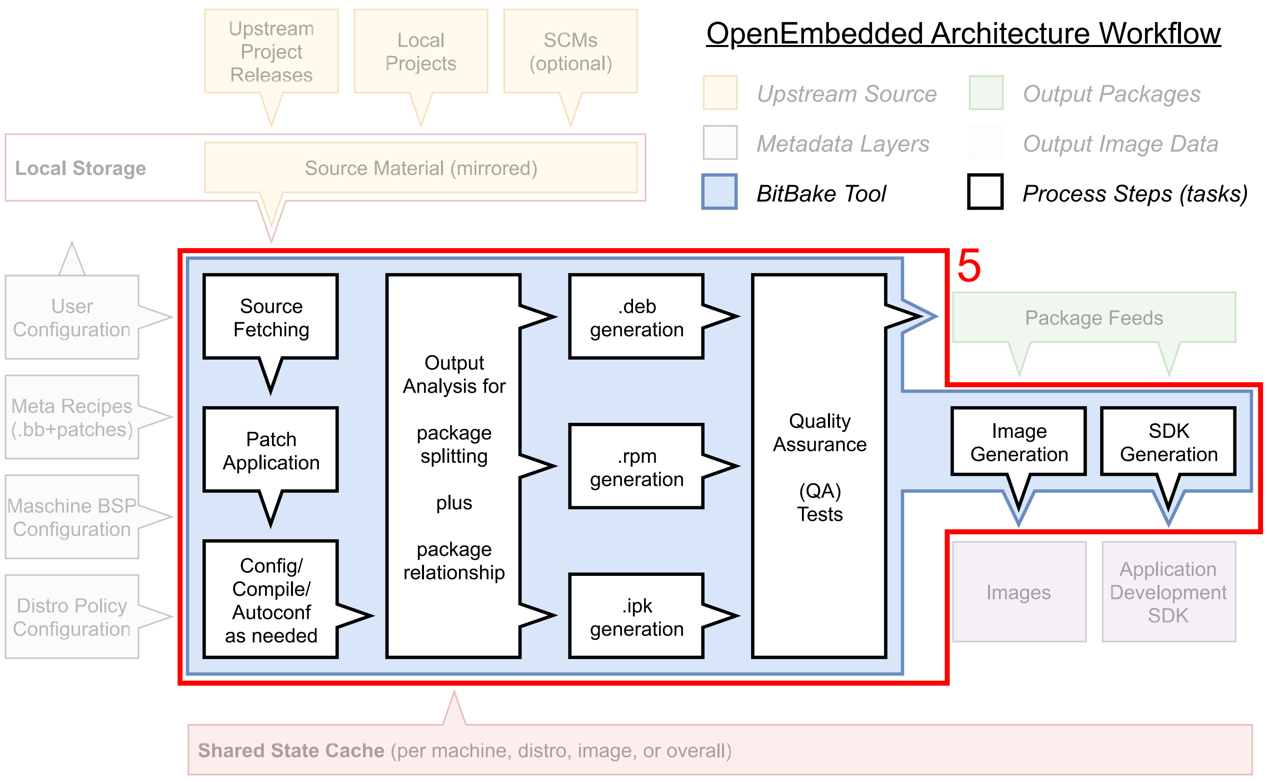

Upstream Project Releases
Local Projects
SCMs (optional)
OpenEmbedded Architecture Workflow
Upstream Source
Output Packages
Metadata Layers
Output Image Data
BitBake Tool
Process Steps (tasks)
Local Storage
Source Material (mirrored)
5
User Configuration
Meta Recipes (.bb+patches)
Maschine BSP Configuration
Distro Policy Configuration
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationship
.deb generation
.rpm generation
.ipk generation
Quality Assurance (QA) Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK
Shared State Cache (per machine, distro, image, or overall)

Figure 7.6: BitBake Tool as part of Yocto WBuild System orkflow

The OpenEmbedded build system uses BitBake to produce images and Software Development Kit (SDK). You can see from the general workflow figure, the BitBake area consists of several functional areas:

1. Source Fetching
2. Patching
3. Configuration, Compilation, and Staging
4. Package Splitting
5. Image Generation
6. SDK Generation

See *BitBake Tool* (page 67) for more details.

7.6 Images Area

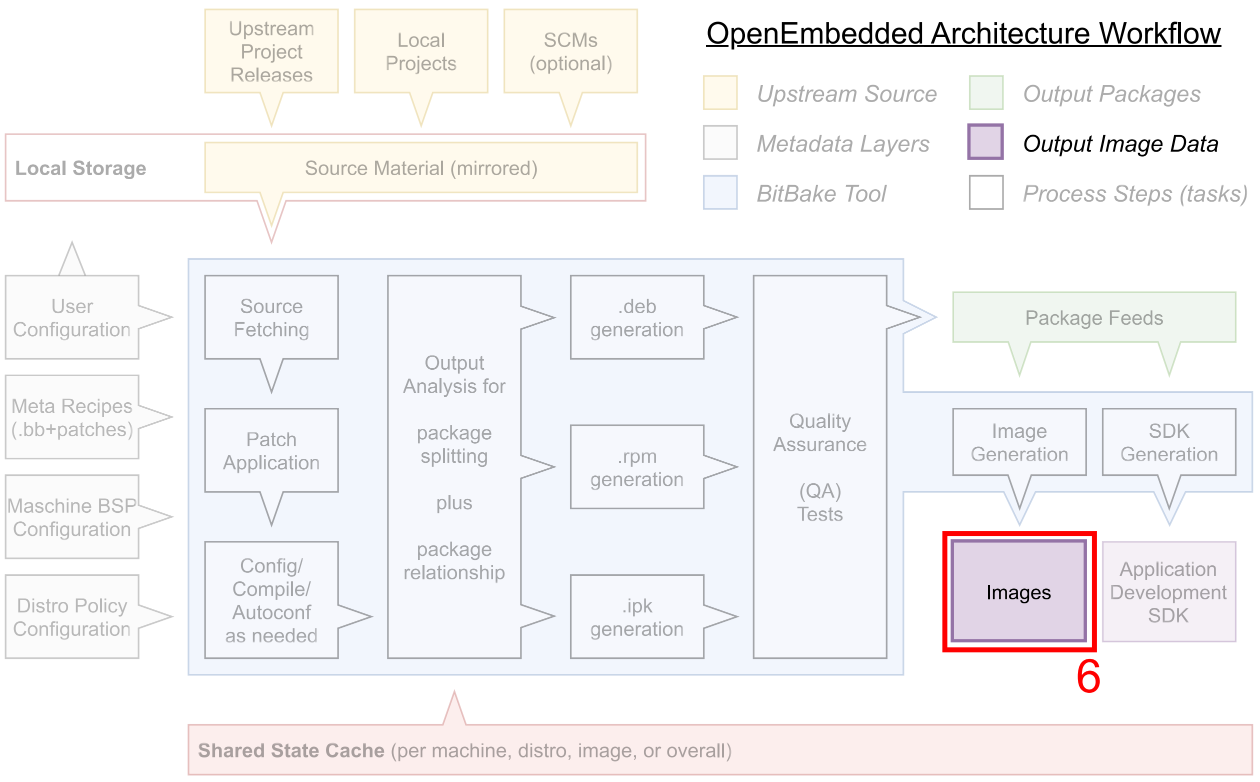

OpenEmbedded Architecture Workflow
Upstream Source
Output Packages
Metadata Layers
Output Image Data
BitBake Tool
Process Steps (tasks)
Upstream Project Releases
Local Projects
SCMs (optional)
Local Storage
Source Material (mirrored)
User Configuration
Meta Recipes (.bb+patches)
Maschine BSP Configuration
Distro Policy Configuration
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationship
.deb generation
.rpm generation
.ipk generation
Quality Assurance (QA) Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK
6
Shared State Cache (per machine, distro, image, or overall)

Figure 7.7: Images as part of Yocto Build System Workflow

The images produced by the build system are compressed forms of the root filesystem and are ready to boot on a target device. You can see from the general workflow figure that BitBake output, in part, consists of images. See *Images* (page 75) for more details.

7.7 SDK Area

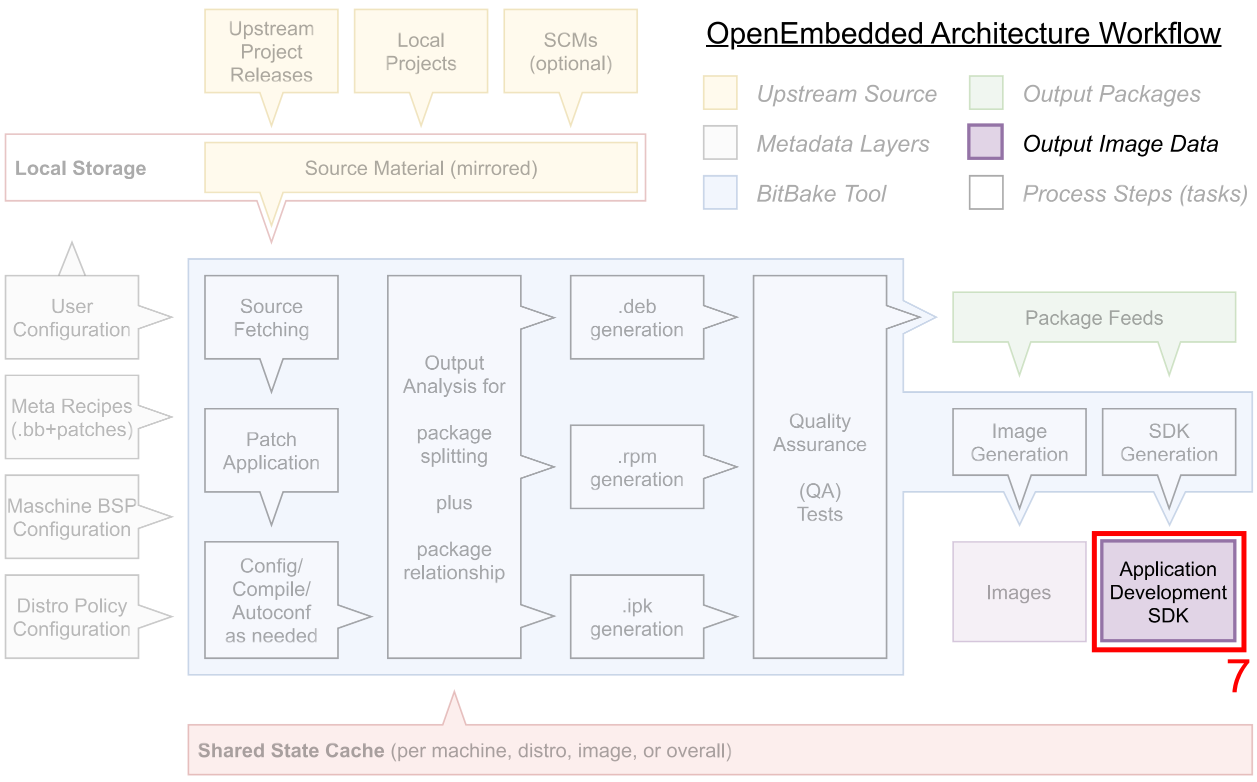

Upstream Project Releases
Local Projects
SCMs (optional)
OpenEmbedded Architecture Workflow
Upstream Source
Output Packages
Metadata Layers
Output Image Data
BitBake Tool
Process Steps (tasks)
Local Storage
Source Material (mirrored)
User Configuration
Meta Recipes (.bb+patches)
Maschine BSP Configuration
Distro Policy Configuration
Source Fetching
Patch Application
Config/ Compile/ Autoconf as needed
Output Analysis for package splitting plus package relationship
.deb generation
.rpm generation
.ipk generation
Quality Assurance (QA) Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK
7
Shared State Cache (per machine, distro, image, or overall)

Figure 7.8: Application Development SDK as part of Yocto Build System Workflow

In the general workflow figure, the output labeled "Application Development SDK" represents an SDK (Software Development Kit). The SDK generation process differs depending on whether you build an extensible SDK (e.g. `bitbake -c populate_sdk_ext imagename`) or a standard SDK (e.g. `bitbake -c populate_sdk imagename`). See *Application Development SDK* (page 77) for more details.

8

User Configuration

This section contains direct excerpts and quotes from publicly accessible documentation: User Configuration — © 2010–2021 The Yocto Project.

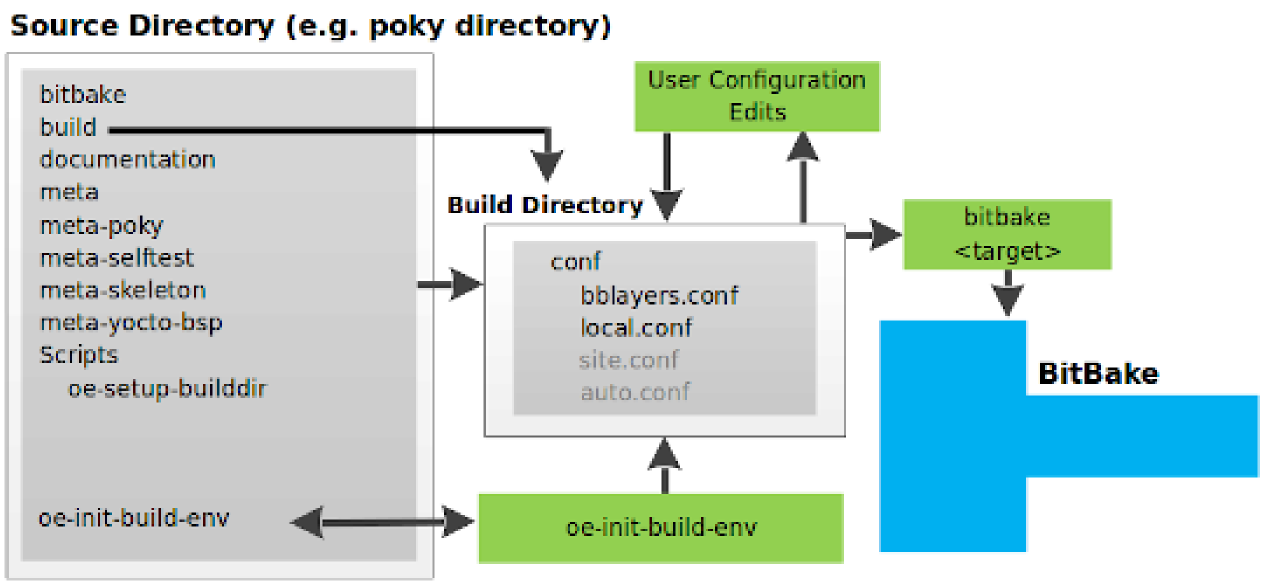


Figure 8.1: User Configuration with Details

BitBake needs some basic configuration files in order to complete a build. These files are `*.conf` files. The minimally necessary ones reside as example files in the `build/conf` directory of the *Source Directory*. For simplicity, this section and Figure 8.1 refers to the *Source Directory* as the *Poky Directory*.

The `meta-poky` layer inside Poky contains a `conf` directory that has example configuration files. These example files are used as a basis for creating actual configuration files when you source oe-init-build-env, which is the build environment script.

Sourcing the build environment script creates a *Build Directory* if one does not already exist. BitBake uses the *Build Directory* for all its work during builds. The *Build Directory* has a `conf` directory that contains default versions of your `local.conf` and `bblayers.conf` configuration files. These default configuration files are created only if versions do not already exist in the *Build Directory* at the time you source the build environment setup script.

Attention: Because the Poky repository is fundamentally an aggregation of existing repositories, some users might be familiar with running the oe-init-build-env script in the context of separate OpenEmbedded Core and BitBake repositories rather than a single Poky repository. This discussion **assumes the script is executed from within a cloned or unpacked version of Poky**.

The `conf/local.conf` file provides many basic variables that define a build environment. Here is a list of a few:

Target Machine Selection Controlled by the MACHINE variable.

Download Directory Controlled by the DL_DIR variable. Mostly set in `conf/site.conf`. → **Local Storage** → **Source Material (mirrored)**

Shared State Directory Controlled by the SSTATE_DIR variable. Mostly set in `conf/site.conf`. → **Shared State Cache**

Build Output Controlled by the TMPDIR variable.

Distribution Policy Controlled by the DISTRO variable.

Packaging Format Controlled by the PACKAGE_CLASSES variable.

SDK Target Architecture Controlled by the SDKMACHINE variable.

Extra Image Packages Controlled by the EXTRA_IMAGE_FEATURES variable.

Note: Configurations set in the `conf/local.conf` file can also be set in the `conf/site.conf` and `conf/auto.conf` configuration files. Both files are not created by the environment initialization script (oe-init-build-env). If you want the `conf/site.conf` file, you need to create that yourself. The `conf/auto.conf` file is typically created by an autobuilder running on a CI (Continuous Integration)/CD (Continuous Deployment / Delivery) environment.

The `conf/bblayers.conf` file tells BitBake what layers you want considered during the build. By default, the layers listed in this file include layers minimally needed by the build system. However, **you must manually add any custom layers you have created**. You can find more information on working with the `bblayers.conf` file in the Enabling Your Layer section in the Yocto Project Development Tasks Manual.

When you launch your build with the **`bitbake target`** command, BitBake sorts out the configurations to ultimately define your build environment. It is important to understand that the *BitBake Tool* (page 67) reads the configuration files in a specific order: `conf/site.conf`, `conf/auto.conf`, and `conf/local.conf`. And, the build system applies the normal assignment statement rules as described in the Syntax and Operators chapter of the BitBake User Manual. Because the files are parsed in a specific order, variable assignments for the same variable could be affected. For example, if the `conf/auto.conf` file and the `conf/local.conf` set `VARIABLE_1` to different values, because the build system parses `conf/local.conf` after `conf/auto.conf`, `VARIABLE_1` is assigned the value from the `conf/local.conf` file.

8.1 bblayers.conf

https://github.com/lipro-yocto/lpn-central → meta-cedi/conf

```
CEDI_BBLAYERS_CONF_VERSION = ”1”

BBPATH = ”${TOPDIR}”
BSPDIR := ”${@os.path.abspath('##OEROOT##/../..')}”
BBFILES ?= ””

BBLAYERS = ” \
  ${BSPDIR}/sources/poky/meta \
  ${BSPDIR}/sources/poky/meta-poky \
  ${BSPDIR}/sources/poky/meta-yocto-bsp \
  \
  ${BSPDIR}/sources/yocto/meta-mingw \
  \
  ${BSPDIR}/sources/openembedded/meta-oe \
  ${BSPDIR}/sources/openembedded/meta-python \
  ${BSPDIR}/sources/openembedded/meta-initramfs \
  ${BSPDIR}/sources/openembedded/meta-filesystems \
  ${BSPDIR}/sources/openembedded/meta-networking \
  ${BSPDIR}/sources/openembedded/meta-multimedia \
  \
  ${BSPDIR}/sources/qt/meta-qt5 \
  \
  ${BSPDIR}/sources/central/meta \
  ${BSPDIR}/sources/central/meta-cedi \
  ${BSPDIR}/sources/central/meta-lpn-bsp \
”
```

The `bblayers.conf` file tells BitBake what layers you want considered during the build.

8.2 local.conf

https://github.com/lipro-yocto/lpn-central → meta-cedi/conf

The `local.conf` file provides many basic variables that define a build environment. Here is a list of a few:

Target Machine Selection: `MACHINE ??= ”qemux86-64”`

Target Distribution Selection: `DISTRO ?= ”cedi”`

Target Packaging Format: `PACKAGE_CLASSES ?= ”package_rpm”`

Download Directory: `DL_DIR ?= ”${TOPDIR}/downloads”`

Shared State Cache Directory: `SSTATE_DIR ?= ”${TOPDIR}/sstate-cache”`

Build Output: `TMPDIR = ”${TOPDIR}/tmp”`

Parallelism Options: `BB_NUMBER_THREADS ?= ”4”`

Host SDK Machine Selection: `SDKMACHINE ?= ”x86_64”`

The **`setup-environment`** script have to use to generate the content of the `local.conf` file.

8.3 `setup-environment`

https://github.com/lipro-yocto/lpn-central → scripts

The **`setup-environment`** script provides many basic options that setup a build environment with the correct *User Configuration*. Here is a short synopsis:

```
DISTRO=”cedi” \
MACHINE=”helium” \
SDKMACHINE=”x86_64-mingw32” \
BUILD_DIR=”build-cedi-helium” \
source setup-environment
```

The **`setup-environment`** calls the **`oe-init-build-env`** script implicitly from the correct place in the Poky *Source Directory*.

To get a detailed online help use:

```
source setup-environment --help
```

9

Metadata Layers

This section contains direct excerpts and quotes from publicly accessible documentation: Metadata, Machine Configuration, and Policy Configuration — © 2010–2021 The Yocto Project.

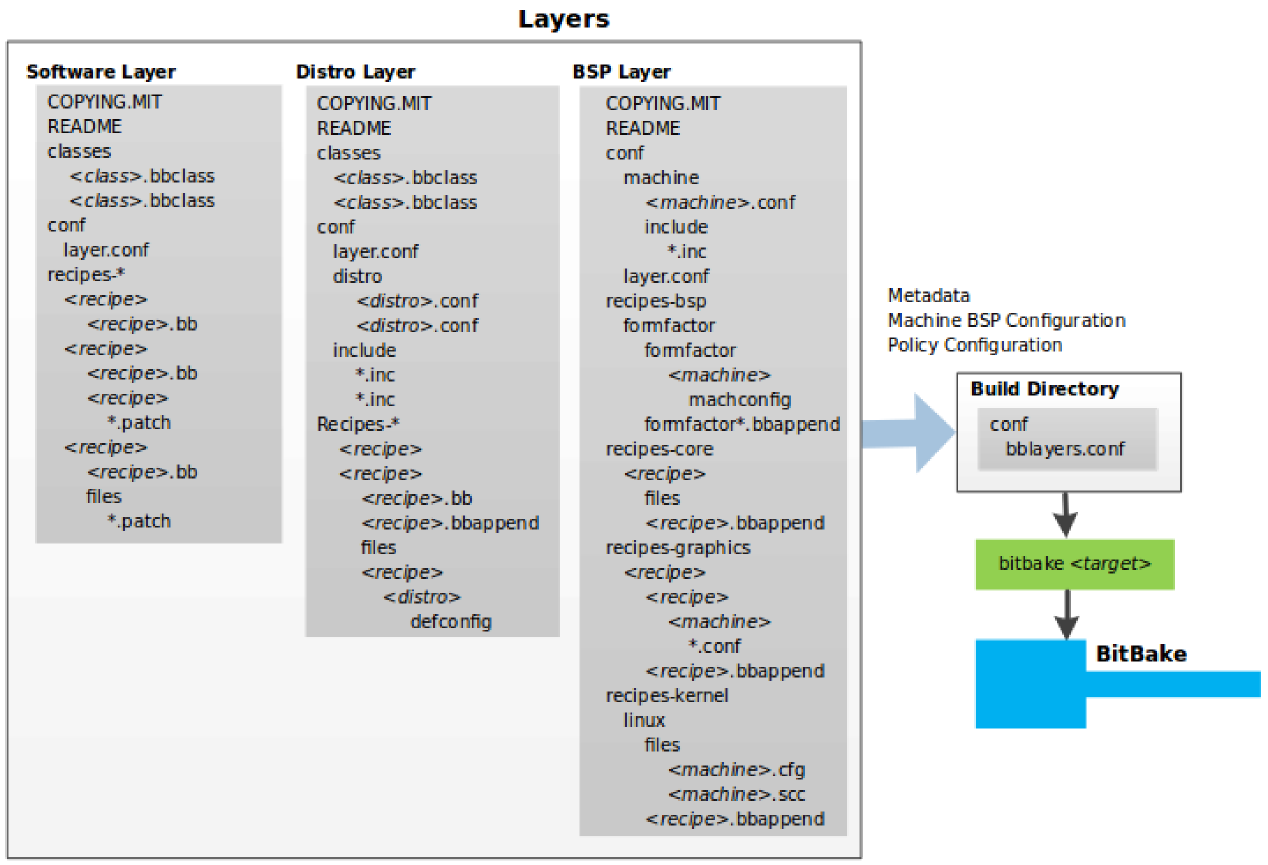


Figure 9.1: Metadata Layers with Details

In general, three types of layer input exists. You can see them below *User Configuration* box in Figure 7.1:

Metadata (.bb + Patches) Software Layers containing user-supplied recipe files, patches, and append files. A good example of a software layer might be the meta-qt5 layer. This layer is for version 5.0 of the popular Qt cross-platform application development framework for desktop, embedded and mobile.

Policy Configuration Distribution Layers (i.e. **Distro Layer** in the following Figure 9.1) providing top-level or general policies for the images or SDKs being built for a particular distribution. For example, in the Reference Embedded Distribution (Poky) the distro layer is the meta-poky layer. Within the distro layer is a `conf/distro` directory that contains distro configuration files (e.g. poky.conf that contain many policy configurations for the Poky distribution).

Machine BSP Configuration Board Support Package Layers (i.e. **BSP Layer** in the following Figure 9.1) providing machine-specific configurations. This type of information is specific to a particular target architecture. A good example of a BSP layer from the Reference Embedded Distribution (Poky) is the meta-yocto-bsp layer.

In general, all layers have a similar structure. They all contain a licensing file (e.g. `COPYING.MIT`) if the layer is to be distributed, a `README` file as good practice and especially if the layer is to be distributed, a configuration directory, and recipe directories. You can learn about the general structure for layers used with the Yocto Project in the Creating Your Own Layer section in the Yocto Project Development Tasks Manual.

Hint: BitBake uses the `conf/bblayers.conf` file, which is part of the *User Configuration*, to find what layers it should be using as part of the build.

Software Layer

The software layer provides the Metadata for additional software packages used during the build. This layer does not include Metadata that is specific to the distribution or the machine, which are found in their respective layers.

This layer contains any recipes (`.bb`), append files (`.bbappend`), and patches, that your project needs.

Distro Layer

The distribution layer provides policy configurations for your distribution. Best practices dictate that you isolate these types of configurations into their own layer. Settings you provide in `conf/distro/distro.conf` override similar settings that BitBake finds in your `conf/local.conf` file in the Build Directory.

The following list provides some explanation and references for what you typically find in the distribution layer:

classes Class files (`.bbclass`) hold common functionality that can be shared among recipes in the distribution. When your recipes inherit a class, they take on the settings and functions for that class. You can read more about class files in the Classes chapter of the Yocto Project Reference Manual.

conf This area holds configuration files for the layer (`conf/layer.conf`), the distribution (`conf/distro/distro.conf`), and any distribution-wide include files.

recipes-* Recipes (`.bb`) and append files (`.bbappend`) that affect common functionality across the distribution. This area could include recipes and append files to add distribution-specific configuration, initialization scripts, custom image recipes, and so forth. Examples of `recipes-*` directories are `recipes-core` and `recipes-extra`. Hierarchy and contents within a `recipes-*` directory can vary. Generally, these directories contain recipe files (`*.bb`), recipe append files (`*.bbappend`), directories that are distro-specific for configuration files, and so forth.

BSP Layer

The BSP layer provides machine configurations that target specific hardware. Everything in this layer is specific to the machine for which you are building the image or the SDK. A common structure or form is defined for BSP layers. You can learn more about this structure in the Yocto Project Board Support Package Developer's Guide.

Attention: In order for a BSP layer to be considered compliant with the Yocto Project, it must meet some structural requirements.

The BSP layer's configuration directory contains configuration files for the machine (`conf/machine/machine.conf`) and, of course, the layer (`conf/layer.conf`). The remainder of the layer is dedicated to specific recipes by function: `recipes-bsp`, `recipes-core`, `recipes-graphics`, `recipes-kernel`, and so forth. Metadata can exist for multiple form factors, graphics support systems, and so forth.

Note: While Figure 9.1 shows several `recipes-*` directories, not all these directories appear in all BSP layers.

9.1 Meta Layer Types

Software Layers: Metadata .bb + Patches

user/vendor/community supplied recipe files, patches and append files

Distribution Layers: Policy Configuration

top-level or general policies for *Images* or *SDKs* being built

BSP Layers: Machine Configuration

specific to a particular target architecture

9.2 Meta Layer Stack

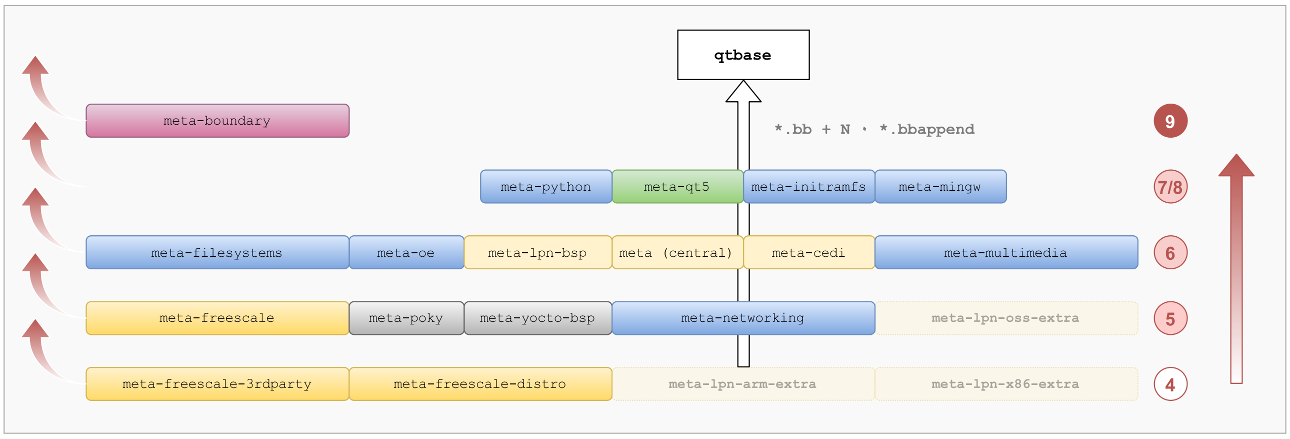


Figure 9.2: Metadata Layer Priorities (`BBFILE_PRIORITY_LayerName`)

See Figure 6.1 for reflection to related origin.

9.3 Meta Layer Machines

https://github.com/lipro-yocto/lpn-central → meta-lpn-bsp/conf/machine

The `meta-lpn-bsp` layer provides machine configurations for all the specific evaluation hardware based on QEMU or real target architecture. The following target machines are supported:

hydrogen: QEMU x86 machine based on `qemux86`

helium: QEMU x86-64 machine based on `qemux86-64`

9.3.1 hydrogen.conf

```
#@TYPE: Machine
#@NAME: QEMU x86 machine
#@DESCRIPTION: Machine configuration for running an x86 system on QEMU (based on␣
↪qemux86).
#@MAINTAINER: Stephan Linz <linz@li-pro.net>

require conf/machine/include/qemux86.inc
require conf/machine/include/lpn-emu.inc
```

qemux86.inc

```
# derived from qemux86
require conf/machine/qemux86.conf
MACHINEOVERRIDES_prepend = "qemux86:"

# derived this machine back to qemux86 for the Linux kernel
KMACHINE = "qemux86"
```

lpn-emu.inc

```
# is a Li-Pro.Net emulated system
require conf/machine/include/lpn.inc
MACHINEOVERRIDES_append = ":lpnemu"
```

lpn.inc

```
# is a Li-Pro.Net system
MACHINEOVERRIDES_append = ":lpn"
```

9.3.2 helium.conf

```
#@TYPE: Machine
#@NAME: QEMU x86-64 machine
#@DESCRIPTION: Machine configuration for running an x86-64 system on QEMU (based on␣
↪qemux86-64).
#@MAINTAINER: Stephan Linz <linz@li-pro.net>

require conf/machine/include/qemux86-64.inc
require conf/machine/include/lpn-emu.inc
```

qemux86-64.inc

```
# derived from qemux86-64
require conf/machine/qemux86-64.conf
MACHINEOVERRIDES_prepend = "qemux86-64:"

# remap derived machine back to qemux86-64 for the Linux kernel
KMACHINE = "qemux86-64"
```

lpn-emu.inc

```
# is a Li-Pro.Net emulated system
require conf/machine/include/lpn.inc
MACHINEOVERRIDES_append = ":lpnemu"
```

lpn.inc

```
# is a Li-Pro.Net system
MACHINEOVERRIDES_append = ":lpn"
```

9.4 Meta Layer Distros

https://github.com/lipro-yocto/lpn-central → meta-cedi/conf/distro

The `meta-cedi` layer provides policy configurations for all the specific evaluation distributions based on Poky. The following distributions are supported:

cedi: Central Distro (Li-Pro.Net Demonstration) based on poky

9.4.1 cedi.conf

```
require ${COREBASE}/meta-poky/conf/distro/poky.conf

DISTROOVERRIDES = "poky:${@d.getVar('DISTRO') or ''}"

DISTRO = "cedi"
DISTRO_NAME = "Central Distro (Li-Pro.Net Demonstration)"
DISTRO_VERSION_append = "+r1.0"
DISTRO_CODENAME_append = "-cedi"
TARGET_VENDOR = "-cedi"
SDK_VENDOR = "-cedisdk"

MAINTAINER = "Cedi <cedi@li-pro.net>"

LOCALCONF_VERSION = "1"
```

```
# Override poky pre-settings (remove).
POKY_DEFAULT_DISTRO_FEATURES = ""
POKY_DEFAULT_EXTRA_RDEPENDS = ""
POKY_DEFAULT_EXTRA_RRECOMMENDS = ""

# Setup Cedi defaults.
CEDI_DEFAULT_DISTRO_FEATURES = "largefile opengl ptest multiarch wayland vulkan"
CEDI_DEFAULT_EXTRA_RDEPENDS = "packagegroup-core-boot"
```

(continues on next page)

(continued from previous page)

```
CEDI_DEFAULT_EXTRA_RRECOMMENDS = ”kernel-module-af-packet”

DISTRO_FEATURES = ”${DISTRO_FEATURES_DEFAULT} ${CEDI_DEFAULT_DISTRO_FEATURES}”
DISTRO_EXTRA_RDEPENDS += ” ${CEDI_DEFAULT_EXTRA_RDEPENDS}”
DISTRO_EXTRA_RRECOMMENDS += ” ${CEDI_DEFAULT_EXTRA_RRECOMMENDS}”
```

```
# Use systemd for system initialization
DISTRO_FEATURES_append = ” systemd”
VIRTUAL-RUNTIME_init_manager = ”systemd”
VIRTUAL-RUNTIME_login_manager = ”shadow-base”
DISTRO_FEATURES_BACKFILL_CONSIDERED += ”sysvinit”
VIRTUAL-RUNTIME_initscripts = ”systemd-compat-units”

# Add Cedi sanity bbclass.
INHERIT += ”cedi-sanity”
```

9.5 Meta Layer Images

https://github.com/lipro-yocto/lpn-central → meta/recipes-core/images

The meta (central core) layer provides recipes and classes for all the specific evaluation and demonstration images and package groups based on the customized image class central-image.bbclass. The following images are supported:

central-image-minimal: A small image just capable of allowing an Central device to boot.

central-dev-image: A developer image just capable of allowing an Central device to boot.

central-debug-image: A debugging image just capable of allowing an Central device to boot.

central-image: A product image capable of allowing an Central device to boot and provides full feature support.

9.5.1 central-image.bb

```
require central-image-base.bb

SUMMARY = ”A product image capable of allowing an Central \
device to boot and provides full feature support.”

DESCRIPTION = ”A product image capable of allowing an Central \
device to boot in graphical mode and provides full feature support.”

LICENSE = ”MIT”
PR = ”r0”

IMAGE_FEATURES += ” \
    ${@bb.utils.contains('DISTRO_FEATURES', 'x11', 'x11-base', '', d)} \
```

(continues on next page)

(continued from previous page)

```
"

inherit central-image
```

central-image-base.bb

```
SUMMARY = "A console-only image that fully supports the target device \
hardware provided by Central."

LICENSE = "MIT"
PR = "r0"

IMAGE_FEATURES += " \
    package-management \
    splash \
"

inherit central-image
```

9.5.2 central-image.bbclass

```
# Common code for generating Central reference images

LIC_FILES_CHKSUM = "\
    file://${CENTRALCOREBASE}/LICENSE;md5=4d183b8707e22082e5a8c5ad268e5149 \
"
```

```
# - qt5-sdk           - Qt5/X11 SDK and demo applications
# - tools-cross       - tools usable for basic cross development
FEATURE_PACKAGES_qt5-sdk = "\
    packagegroup-qt5-toolchain-target \
    packagegroup-qt5-qtcreator-debug \
"
FEATURE_PACKAGES_tools-cross = ""
# Provides the Central specific features 'qt5-sdk' and 'tools-cross'.
```

```
CENTRAL_IMAGE_BASE_INSTALL = '\
    packagegroup-central-boot \
    packagegroup-base-central \
    \
    ${@bb.utils.contains("IMAGE_FEATURES", "tools-cross", \
                         "packagegroup-central-tools-cross", "", d)} \
    ${@bb.utils.contains("IMAGE_FEATURES", "tools-debug", \
                         "packagegroup-central-tools-debug", "", d)} \
    ${@bb.utils.contains("IMAGE_FEATURES", "tools-profile", \
                         "packagegroup-central-tools-profile", "", d)} \
```

(continues on next page)

(continued from previous page)

```
    ${@bb.utils.contains("IMAGE_FEATURES", "tools-testapps", \
                         "packagegroup-central-tools-testapps", "", d)} \
    ${@bb.utils.contains("IMAGE_FEATURES", "tools-sdk", \
                         "packagegroup-central-sdk", "", d)} \
    \
    ${CENTRAL_IMAGE_EXTRA_INSTALL} \
    '

CENTRAL_IMAGE_EXTRA_INSTALL ?= ""

CORE_IMAGE_EXTRA_INSTALL += "${CENTRAL_IMAGE_BASE_INSTALL}"

inherit core-image central-image-version
```

9.5.3 central-image-version.bbclass

```
# Common code for generating Central version file

LIC_FILES_CHKSUM = "\
    file://${CENTRALCOREBASE}/LICENSE;md5=4d183b8707e22082e5a8c5ad268e5149 \
"
```

```
CENTRAL_VERSION_FILE = "${IMAGE_ROOTFS}${sysconfdir}/central_version"

write_central_version() {
        cat > ${CENTRAL_VERSION_FILE} <<EOF
[build information]
vendor-id=LPN
manufacturer-name=Li-Pro.Net
device-variant=${MACHINE}
purpose=${IMAGE_BASENAME}
feature=${IMAGE_FEATURES}
build-number=${BUILD_NUMBER}
EOF
}
```

```
ROOTFS_POSTPROCESS_COMMAND += "write_central_version;"
```

9.5.4 packagegroup-central-boot.bb

```
SUMMMARY = ”Central Minimal Boot Requirements”
DESCRIPTION = ”The minimal set of packages required to boot a Central system”
LICENSE = ”MIT”
PR = ”r0”

# For backwards compatibility after rename
RPROVIDES_packagegroup-central-boot = ”packagegroup-boot-central”
RREPLACES_packagegroup-central-boot = ”packagegroup-boot-central”
RCONFLICTS_packagegroup-central-boot = ”packagegroup-boot-central”
```

```
# packages which content depend on MACHINE_FEATURES need to be MACHINE_ARCH
PACKAGE_ARCH = ”${MACHINE_ARCH}”

inherit packagegroup

# Set by the machine configuration with packages essential
# for Central device bootup.
MACHINE_ESSENTIAL_EXTRA_CENTRAL_RDEPENDS ?= ””
MACHINE_ESSENTIAL_EXTRA_CENTRAL_RRECOMMENDS ?= ””
```

```
RDEPENDS_${PN} = ” \
    ${MACHINE_ESSENTIAL_EXTRA_CENTRAL_RDEPENDS} \
”

RRECOMMENDS_${PN} = ” \
    ${MACHINE_ESSENTIAL_EXTRA_CENTRAL_RRECOMMENDS} \
”
```

9.5.5 packagegroup-central-tools-testapps.bb

```
SUMMMARY = ”Central Testing tools/applications”
DESCRIPTION = ”The testing set of packages required for a Central system”
LICENSE = ”MIT”
PR = ”r0”
```

```
inherit packagegroup
```

```
# Event device test tools
EVTOOLS = ”evtest”

# Simple memmory access tools
MEMTOOLS = ”devmem2 libuio”

# Simple serial bus tools
SERBUSTOOLS = ”i2c-tools spitools”
```

(continues on next page)

(continued from previous page)

```

# Requires Serial to work
USE_SERIAL = ”picocom serial-forward”
```

```
RDEPENDS_${PN} = ” \
    ${EVTOOLS} \
    ${MEMTOOLS} \
    ${SERBUSTOOLS} \
    ${@bb.utils.contains('MACHINE_FEATURES', 'serial', '${USE_SERIAL}', '',d)} \
”

RRECOMMENDS_${PN} = ” \
”
```

10

Source Files

This section contains direct excerpts and quotes from publicly accessible documentation: Sources — © 2010–2021 The Yocto Project.

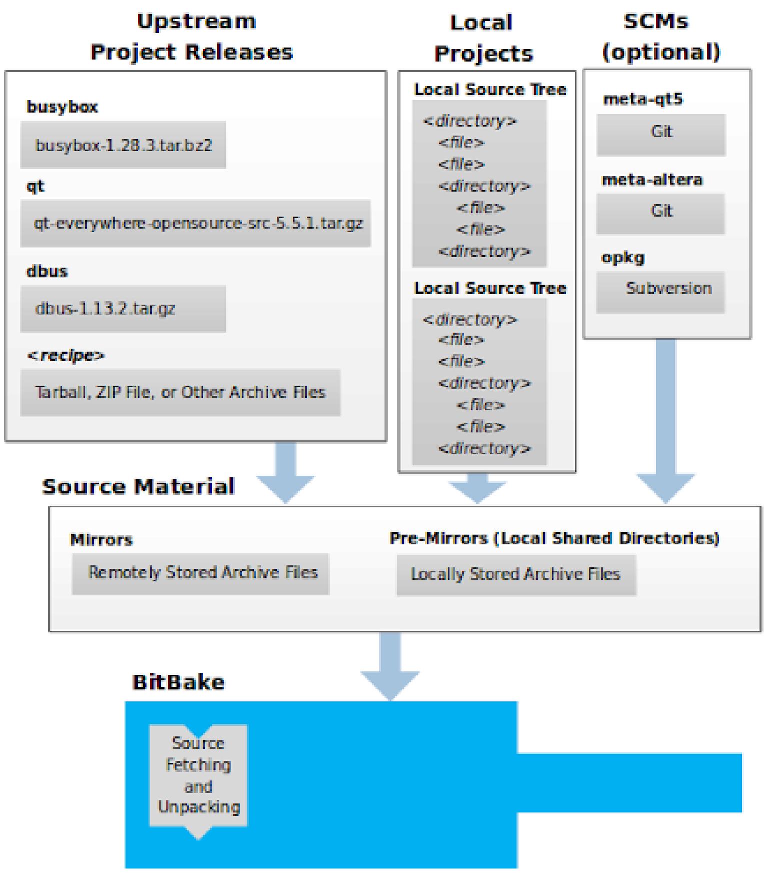


Figure 10.1: Sources with Details

The general workflow in Figure 7.1 represents source files using the *Upstream Project Releases*, *Local Projects*, and *SCMs (optional)* boxes. The figure represents mirrors, which also play a role in locating source files, with the *Source Material* box.

BitBake uses the SRC_URI variable to point to source files regardless of their location. Each recipe must have a SRC_URI variable that points to the source.

Another area that plays a significant role in where source files come from is pointed to by the DL_DIR variable. This area is a cache that can hold previously downloaded source. Judicious use of a `DL_DIR` directory can save the build system a trip across the Internet when looking for files. A good method for using a download directory is to have `DL_DIR` point to an area outside of your Build Directory. Doing so allows you to safely delete the Build Directory if needed without fear of removing any downloaded source file.

Tip: The method by which source files are ultimately organized is a function of the project. For example, for released software, projects tend to use tarballs or other archived files that can capture the state of a release guaranteeing that it is statically represented. On the other hand, for a project that is more dynamic or experimental in nature, a project might keep source files in a repository controlled by a Software Configuration Management System (SCMS) such as Git. Pulling source from a repository allows you to control the point in the repository (the revision) from which you want to build software. Finally, a combination of the two might exist, which would give the consumer a choice when deciding where to get source files.

Upstream Project Releases

Upstream Project Releases exist anywhere in the form of an archived file (e.g. tarball or zip file). These files correspond to individual recipes. For example, the figure uses specific releases each for BusyBox, Qt, and D-Bus. An archive file can be for any released product that can be built using a recipe.

Local Projects

Local Projects are custom bits of software the user provides. These bits reside somewhere local to a project – perhaps a directory into which the user checks in items (e.g. a local directory containing a development source tree used by the group).

The canonical method through which to include a local project is to use the externalsrc.bbclass class to include that local project. You use either the `local.conf` or a recipe's append file to override or set the recipe to point to the local directory on your disk to pull in the whole source tree.

Source Control Managers (Optional)

Another place from which the build system can get source files is with fetchers employing various Software Configuration Management System such as Git or Subversion. In such cases, a repository is cloned or checked out. The do_fetch task inside BitBake uses the SRC_URI variable and the argument's prefix to determine the correct fetcher module.

When fetching a repository, BitBake uses the SRCREV variable to determine the specific revision from which to build.

Source Mirror(s)

Two kinds of mirrors exist:

pre-mirrors The PREMIRRORS variables point to this. BitBake checks pre-mirrors before looking upstream for any source files. Pre-mirrors are appropriate when you have a shared directory that is not a directory defined by the DL_DIR variable. A Pre-mirror typically points to a shared directory that is local to your organization.

regular mirrors The MIRRORS variables point to this. Regular mirrors can be any site across the Internet that is used as an alternative location for source code should the primary site not be functioning for some reason or another.

10.1 Upstream Project Releases

(almost all recipes)

- exist anywhere in the form of an archived file (e.g. tarball or ZIP file)
- correspond to individual recipes, PN and PV match with a upstream archive file
- archive file can be for any released product that can be built using a recipe

Example

The `tslib_1.11.bb` recipe points to the following XZ compressed tarball file: https://github.com/kergoth/tslib/releases/download/1.11/tslib-1.11.tar.xz

10.2 Local Projects

(no preference for this)

- custom bits of software the user provides
- these bits reside somewhere local to a project; perhaps a directory into which the user checks in items
- use the externalsrc.bbclass class in recipes to include a local projects

Note

Community does not use local projects! Never seen!

10.3 Source Code Manager (optional)

(almost vendor or product recipes)

- get source files through an SCMS such as Git or Subversion or Mercurial
- repository is cloned and/or checked out

Example

The `meta/conf/layer.conf` file may tells BitBake how to reach special (perhaps hidden) Git, Subversion, or Mercurial servers:

```
CENTRAL_GIT ?= ”git://10.20.30.40/services/git”
CENTRAL_SVN ?= ”svn://10.20.30.40/services/svn”
CENTRAL_HG ?= ”hg://10.20.30.40/services/hg”
```

10.4 Source Mirror(s)

(almost all recipes by policy)

The `meta-poky/conf/distro/poky.conf` file as required by the `meta-cedi/conf/distro/cedi.conf` file provides the two kinds of mirrors:

`PREMIRRORS`: the pre-mirrors

check before looking *Upstream Project Releases*

`MIRRORS`: the regular mirrors

alternative location should the *Upstream Project Releases* not available

11

Package Feeds

This section contains direct excerpts and quotes from publicly accessible documentation: Package Feeds — © 2010–2021 The Yocto Project.

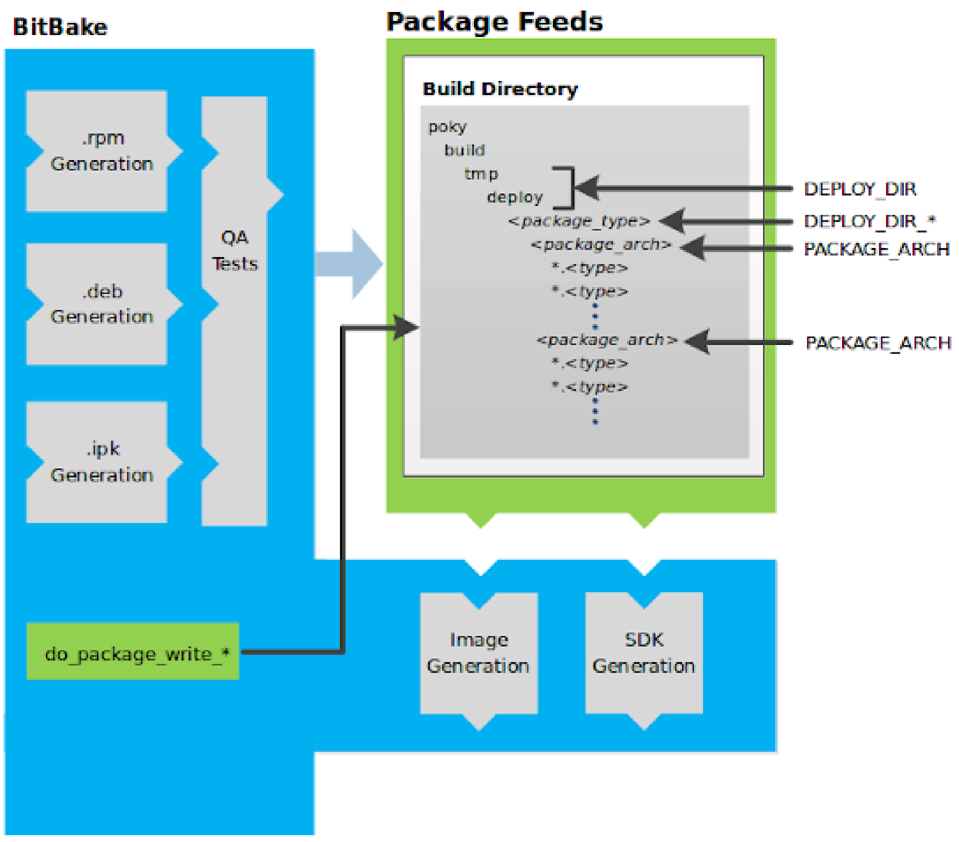


Figure 11.1: Package Feeds with Details

Package feeds are an intermediary step in the build process. The OpenEmbedded build system provides classes to generate different package types, and you specify which classes to enable through the PACKAGE_CLASSES variable. Before placing the packages into package feeds, the build process validates them with generated output quality assurance (QA) checks through the insane.bbclass class.

The package feed area resides in the *Build Directory*. The directory the build system uses to temporarily store packages is determined by a combination of variables and the particular package manager in use. See the *Package Feeds* box in Figure 11.1 and note the information to the right of that area. In particular, the following defines where package files are kept:

- DEPLOY_DIR: Defined as `tmp/deploy` in the *Build Directory* – the *Deploy Directory*.
- `DEPLOY_DIR_*`: Depending on the package manager used, the package type sub-folder. Given RPM, IPK, or DEB packaging and tarball creation, the DEPLOY_DIR_RPM, DEPLOY_DIR_IPK, DEPLOY_DIR_DEB, or DEPLOY_DIR_TAR, variables are used, respectively.
- PACKAGE_ARCH: Defines architecture-specific sub-folders. For example, packages could exist for the `i586` or qemux86 architectures.

Hint: Mostly all that variables set to default values in the BitBake default configuration file `conf/bitbake.conf` that residence inside the *Source Directory*.

BitBake uses the do_package_write_* tasks to generate packages and place them into the package holding area (e.g. do_package_write_ipk for IPK packages). See the do_package_write_deb, do_package_write_ipk, do_package_write_rpm, and do_package_write_tar sections in the Yocto Project Reference Manual for additional information. As an example, consider a scenario where an IPK packaging manager is being used and package architecture support for both `i586` and `qemux86` exist. Packages for the `i586` architecture are placed in `build/tmp/deploy/ipk/i586`, while packages for the `qemux86` architecture are placed in `build/tmp/deploy/ipk/qemux86`.

12 BitBake Tool

This section contains direct excerpts and quotes from publicly accessible documentation: BitBake Tool — © 2010–2021 The Yocto Project.

The OpenEmbedded build system uses BitBake to produce images and Software Development Kits (SDKs). You can see from Figure 7.1, the box *BitBake Tool Area* in the middle consists of several functional stages:

http://git.yoctoproject.org/cgit/cgit.cgi/poky → `meta/classes/*.bbclass`

1. **Source Fetching** tasks: do_fetch → do_unpack
2. **Patch Application** tasks: do_patch
3. **Configure / Compile / Install** tasks: do_configure → do_compile → do_install
4. **Analysis / Splitting Packages** tasks: do_package/data → do_package_write_* → do_populate_sysroot
5. **Image Generation** tasks: do_rootfs → do_image/_complete
6. **SDK Generation** tasks: do_populate_sdk/_ext

Hint: Separate documentation exists for the BitBake tool. See the BitBake User Manual for reference material on BitBake.

12.1 Inherit

tasks by classes:	cmake	base	package/group	image	populate_sdk_base
do_fetch, do_unpack	X	X			
do_patch	X	X			
do_configure, do_compile, do_install	X	X			
do_package/data, do_package_write_*, do_populate_sysroot	X		X		

continues on next page

Table 12.1 – continued from previous page

tasks by classes:	cmake	base	package/group	image	populate_sdk_base
do_rootfs, do_image/_complete				X	
populate_sdk/_ext					X

12.2 Fetching

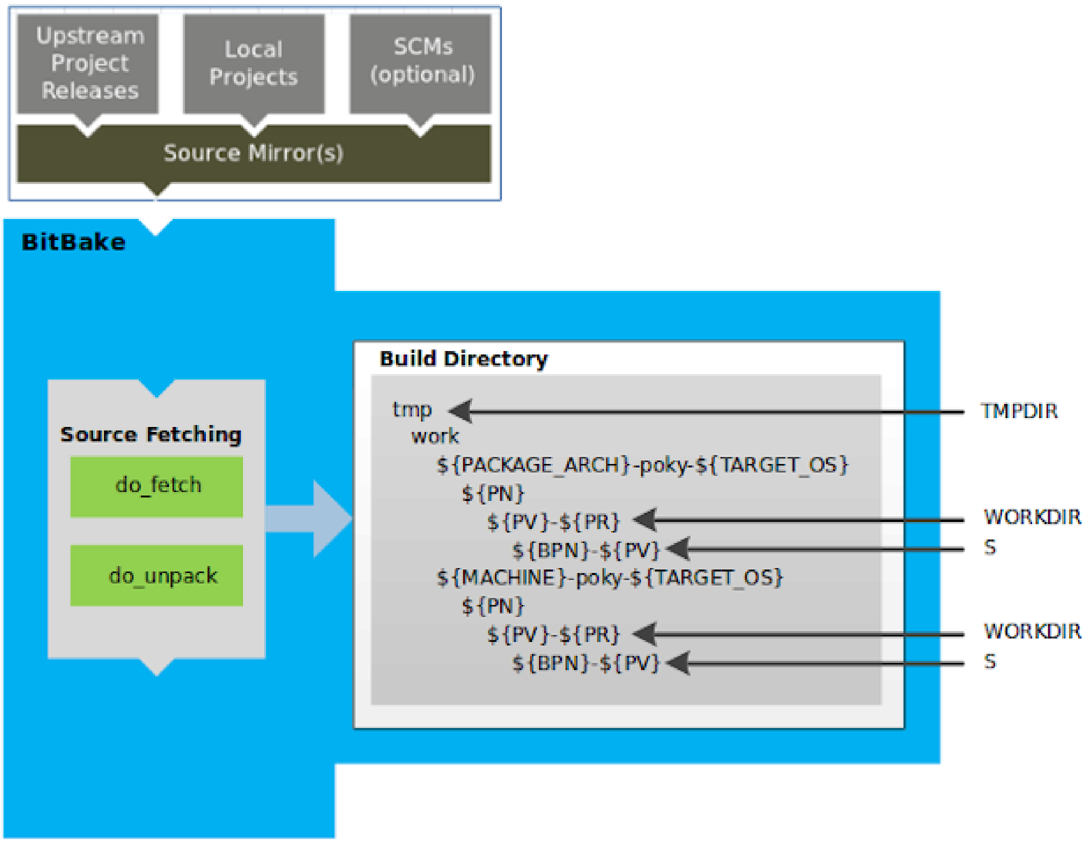


Figure 12.1: Source Fetching with Details

The first stages of building a recipe are to fetch and unpack the source code. The do_fetch and do_unpack tasks fetch the source files and unpack them into the *Build Directory*. BitBake uses the SRC_URI variable to point to source files regardless of their location.

12.2.1 Fetching in Recipe

https://github.com/lipro-yocto/meta-lpn-apps (dummy, not yet)

The `meta-lpn-apps` layer may provides private (closed) recipes. The following related SRC_URI may being used:

lib-utility_hg.bb: `"${CENTRAL_HG};module=lib_utility;branch=default;rev=${SRCREV};protocol=ssh"`

lib-crypto_hg.bb: `"${CENTRAL_HG};module=lib_crypto;branch=default;rev=${SRCREV};protocol=ssh"`

app-mcu-ctrl_hg.bb: `"${CENTRAL_HG};module=app_mcu_ctrl;branch=default;rev=${SRCREV};protocol=ssh"`

boost_1.75.0.bb: `"https://dl.bintray.com/boostorg/release/${PV}/source/${BOOST_P}.tar.bz2"`

For private recipes: SRCREV is set to SRCREV = `"default"` and CENTRAL_HG should comes from e.g. `meta/conf/layer.conf`.

12.3 Patching

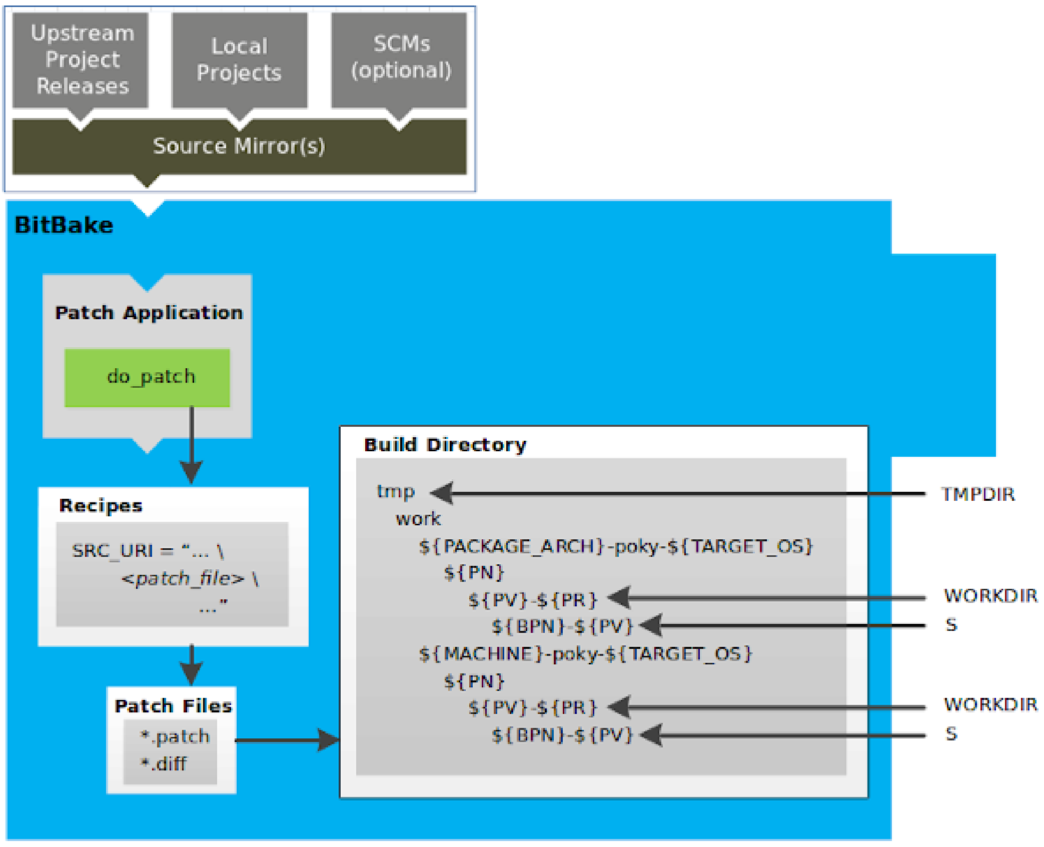


Figure 12.2: Patching with Details

Once source code is fetched and unpacked, BitBake locates patch files and applies them to the source files. The do_patch task processes recipes by using the SRC_URI variable to locate applicable patch files. BitBake finds and applies multiple patches for a single recipe.

12.3.1 Patching in Recipe

http://git.yoctoproject.org/cgit/cgit.cgi/poky

http://cgit.openembedded.org/meta-openembedded

The `poky/meta` and `meta-openembedded/meta-oe` layer provides support recipes. The following related SRC_URI being used:

boost_1.75.0.bb: SRC_URI `+= "file://boost-CVE-2012-2677.patch ..."`

opencv_4.5.0.bb: SRC_URI `= "git://github.com/opencv/opencv.git;name=opencv ... file://download.patch ..."`

Files in the SRC_URI list variable with the `.patch` or `.diff` extension are automatically identified as a patch and applied if present beside the recipe.

12.4 Compilation

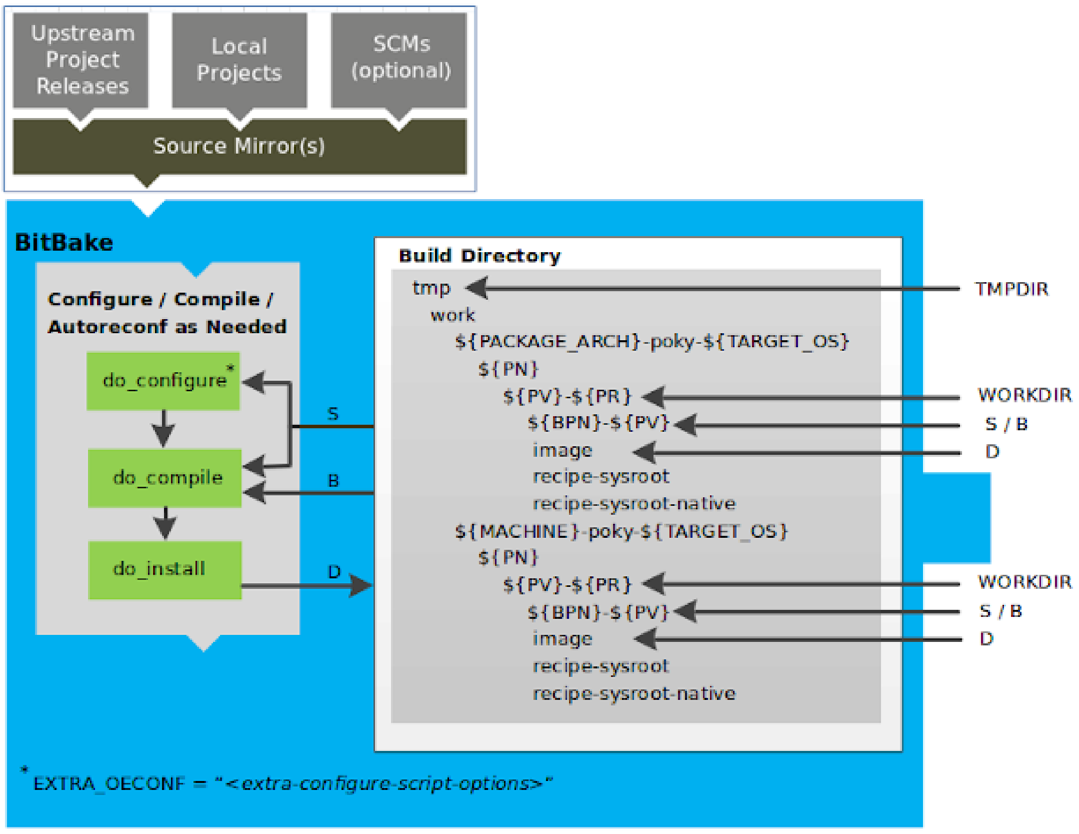


Figure 12.3: Configuration, Compilation, and Staging with Details

After source code is patched, BitBake executes tasks that configure and compile the source code. This step consists of three tasks: do_configure, do_compile and do_install. It starts at the earliest after all dependencies from the DEPENDS variable in the recipe have been resolved.

12.4.1 Compilation in Recipe

https://github.com/lipro-yocto/meta-lpn-apps (dummy, not yet)

The `meta-lpn-apps` layer may provides private (closed) recipes. The following related DEPENDS may being used:

lib-utility_hg.bb: `"boost"`

lib-crypto_hg.bb: `"boost opencv"`

app-mcu-ctrl_hg.bb: `"boost opencv qtbase qtsvg lib-utility lib-crypto"`

boost_1.75.0.bb: `"boost-build-native zlib bzip2"`

opencv_4.5.0.bb: `"libtool swig-native bzip2 zlib glib-2.0 libwebp"`

For the application: RDEPENDS may be set to additional dependent package names that have to be installed later into targets' runtime environment, e.g. RDEPENDS `= "app-observer-daemon"`.

12.5 Packaging

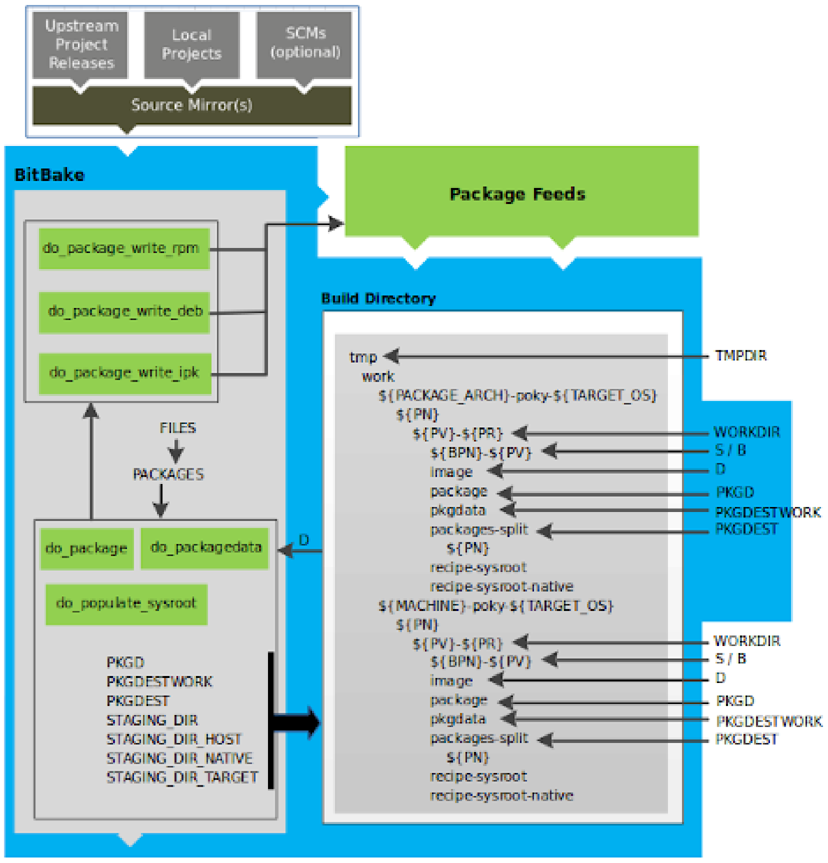


Figure 12.4: Package Splitting with Details

After compilation, the build system analyzes the results and splits the output into packages. The do_package and do_packagedata tasks combine to analyze the files found in the installation directory: debugging symbols, looking at shared library dependencies and relationships of RDEPENDS in the recipe.

12.5.1 Packaging in Recipe

https://github.com/lipro-yocto/meta-lpn-apps (dummy, not yet)

The `meta-lpn-apps` layer may provides private (closed) recipes. The following related RDEPENDS may being used:

app-mcu-ctrl: ”`i2c-tools`”

packagegroup-lpn-apps-toolchain-target: ”`boost-staticdev opencv-staticdev lib-utility-staticdev lib-crypto-staticdev`”

The following related RDEPENDS may being used as policy; mostly controlled by common *Image Features*, e.g. `IMAGE_FEATURES += ”tools-testapps”`:

packagegroup-central-tools-testapps: ”`devregs spitools v4l-utils`”

hostapd_%: ”`lpn-mcu-ssid`”

12.6 Image Generation

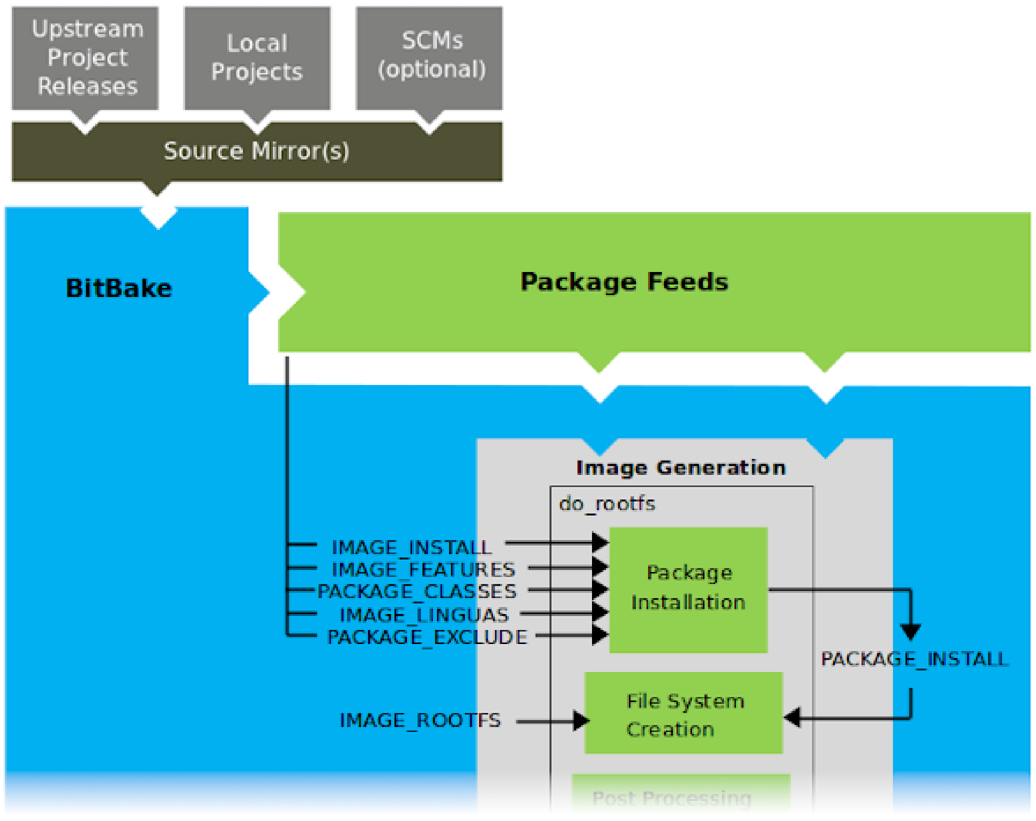


Figure 12.5: Image Generation with Details 1/2

Once packages are split and stored in the *Package Feeds*, the build system uses BitBake to generate the root filesystem image. The do_rootfs task creates the root filesystem for an image.

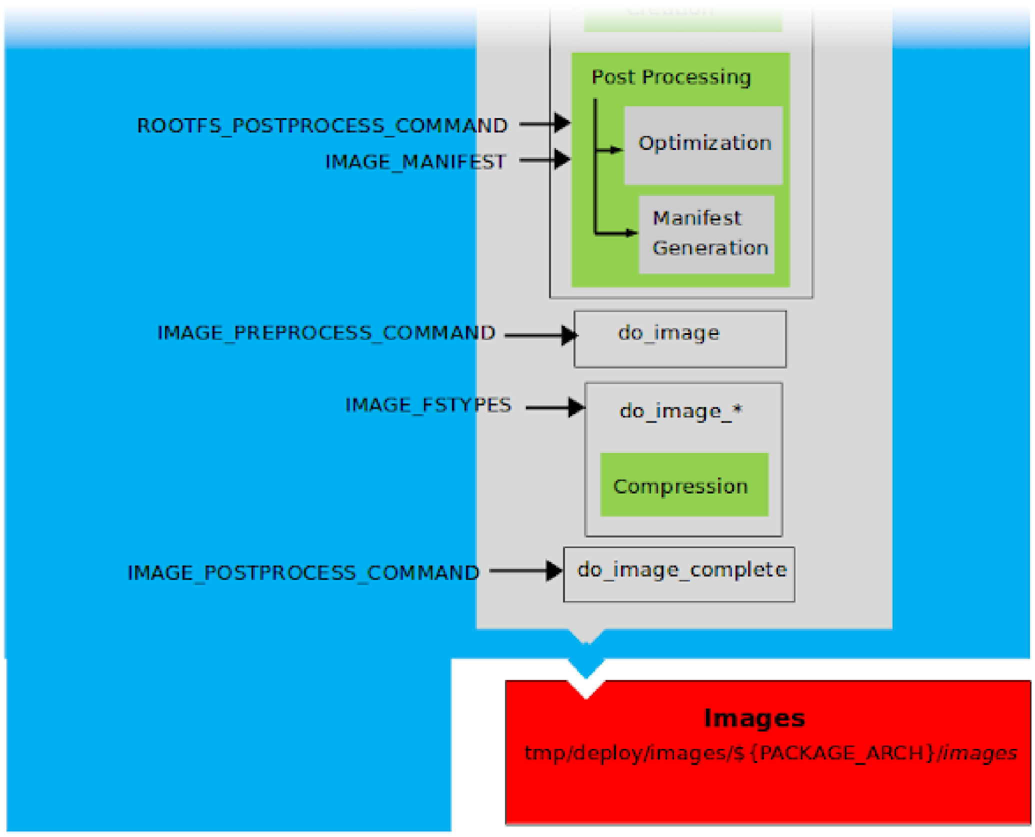


Figure 12.6: Image Generation with Details 2/2

The final stages of the do_rootfs task handle post processing, includes creation of a manifest file and optimizations. After this, processing begins on the image through the do_image task and dynamically created tasks as needed by the image types; ends with do_image_complete.

12.7 SDK Generation

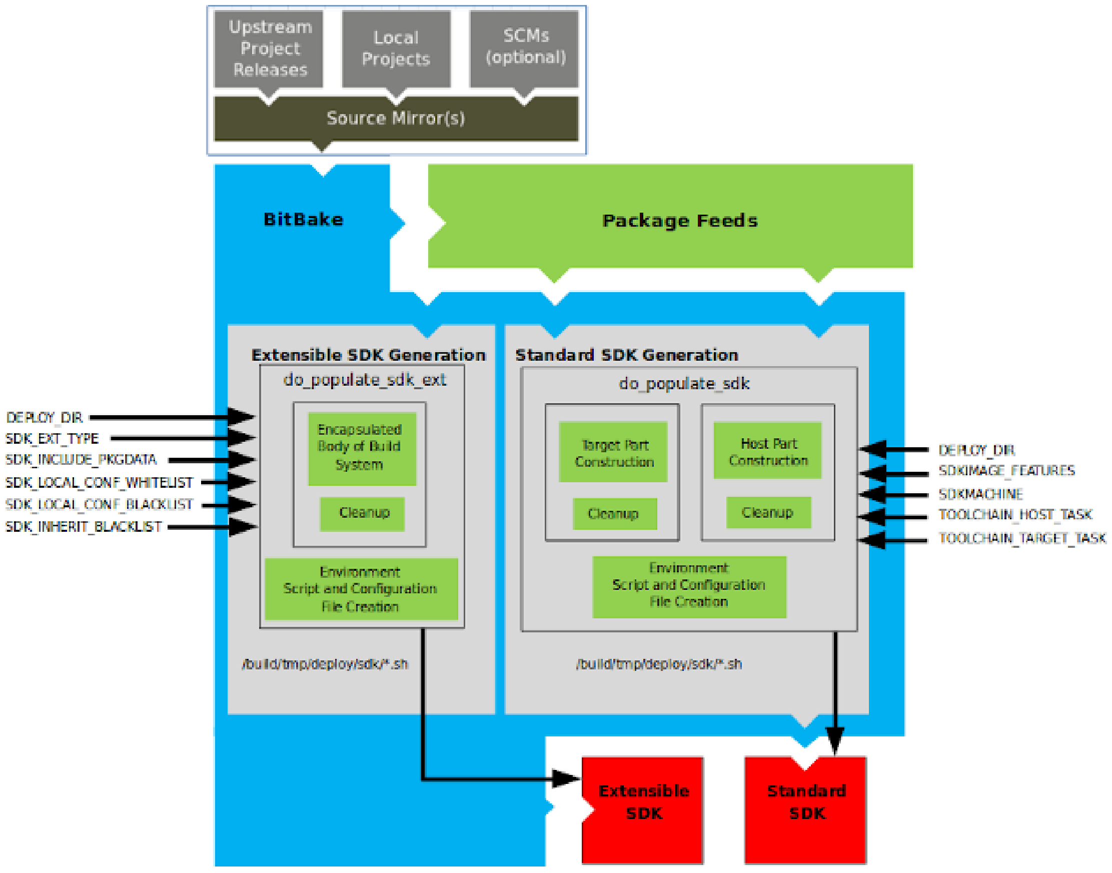


Figure 12.7: SDK Generation with Details

The build system uses BitBake to generate the Software Development Kit (SDK) installer script for both the *Standard SDK* (do_populate_sdk) and *Extensible SDK* (do_populate_sdk_ext). Like image generation, the SDK script process consists of several stages and depends on many variables.

13 Images

This section contains direct excerpts and quotes from publicly accessible documentation: Images — © 2010–2021 The Yocto Project.

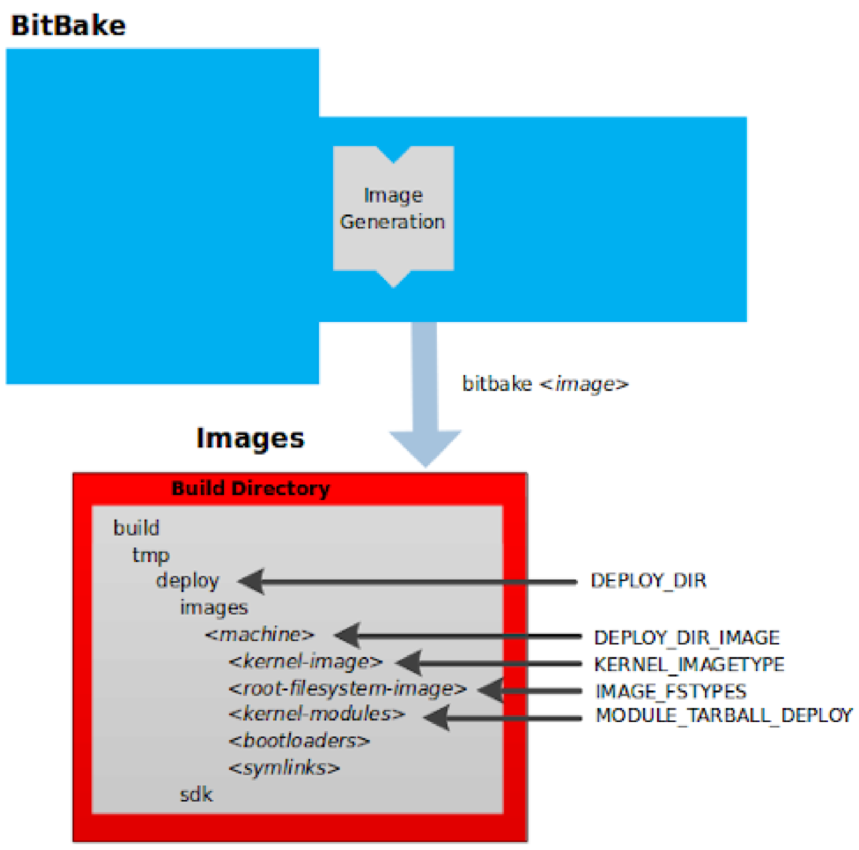


Figure 13.1: Images with Details

The images produced by the build system are compressed forms of the root filesystem and are ready to boot on a target device. You can see from Figure 7.1 that BitBake output, in part, consists of image artifacts.

Hint: For a list of example images that the Yocto Project provides, see the Images chapter in the Yocto Project Reference Manual.

The build process writes images out to the *Build Directory* inside the `tmp/deploy/images/machine/` folder as shown in Figure 13.1. This folder contains any files expected to be loaded on the target device. The DEPLOY_DIR variable points to the *Deploy Directory*, while the DEPLOY_DIR_IMAGE variable points to the appropriate directory containing images for the current configuration.

kernel-image A kernel binary file. The KERNEL_IMAGETYPE variable determines the naming scheme for the kernel image file. Depending on this variable, the file could begin with a variety of naming strings. The `deploy/images/machine` directory can contain multiple image files for the machine.

root-filesystem-image Root filesystems for the target device (e.g. `*.ext3` or `*.bz2` files). The IMAGE_FSTYPES variable determines the root filesystem image type. The `deploy/images/machine` directory can contain multiple root filesystems for the machine.

kernel-modules Tarballs that contain all the modules built for the kernel. Kernel module tarballs exist for legacy purposes and can be suppressed by setting the MODULE_TARBALL_DEPLOY variable to `0`. The `deploy/images/machine` directory can contain multiple kernel module tarballs for the machine.

bootloaders If applicable to the target machine, bootloaders supporting the image. The `deploy/images/machine` directory can contain multiple bootloaders for the machine.

symlinks The `deploy/images/machine` folder contains a symbolic link that points to the most recently built file for each machine. These links might be useful for external scripts that need to obtain the latest version of each file.

14

Application Development SDK

This section contains direct excerpts and quotes from publicly accessible documentation: Application Development SDK — © 2010–2021 The Yocto Project.

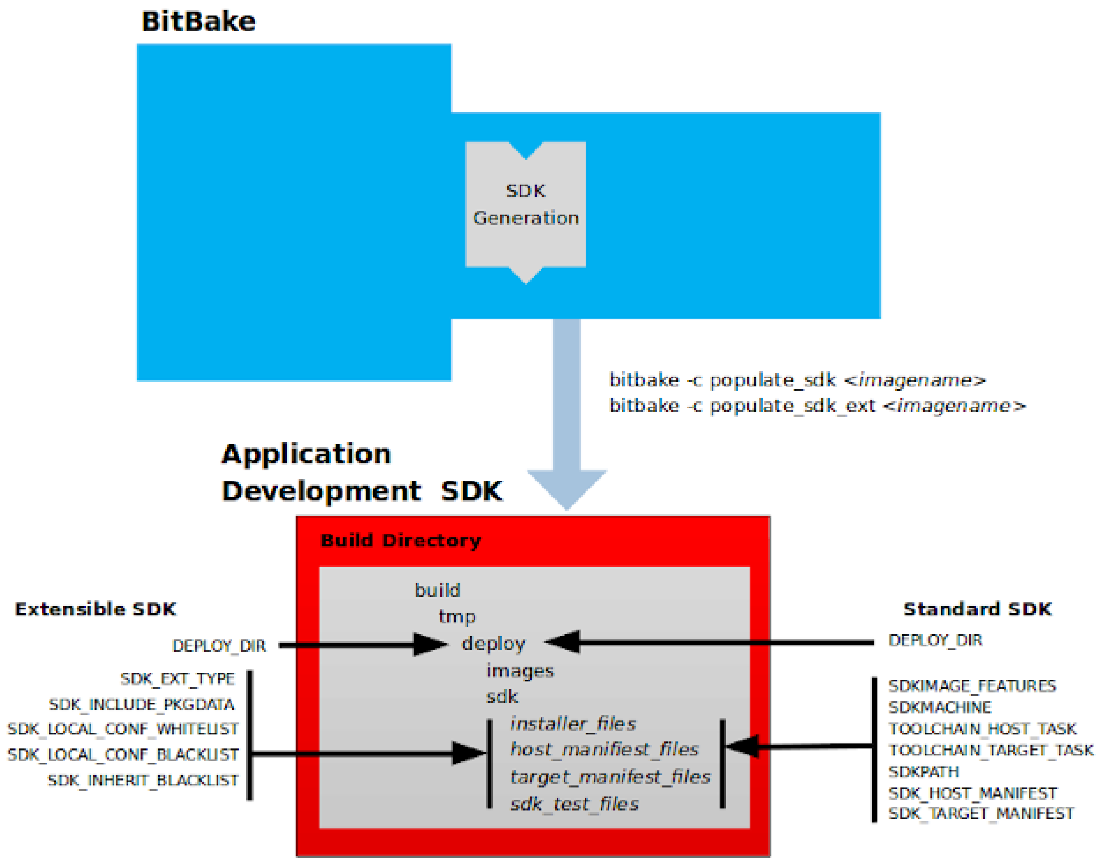


Figure 14.1: Application Development SDK with Details

In Figure 7.1, the output labeled *Application Development SDK* represents an SDK. The SDK generation process differs depending on whether you build an *Extensible SDK* (e.g. **`bitbake -c populate_sdk_ext imagename`**) or a *Standard SDK* (e.g. **`bitbake -c populate_sdk imagename`**).

Hint:

- The Yocto Project supports several methods by which you can set up this cross-development environment. These methods include downloading pre-built SDK installers (e.g. from a tool vendor) or building and installing your own SDK installer with the help of the Yocto Project.

- For background information on cross-development toolchains in the Yocto Project development environment, see the Cross-Development Toolchain Generation section as part of the *BitBake Tool Area*.
- For information on setting up a cross-development environment, see the Yocto Project Application Development and the Extensible Software Development Kit (eSDK) manual.

The specific form of this output is a set of files that includes a self-extracting SDK installer (`*.sh`), host and target manifest files, and files used for SDK testing. When the SDK installer file is run, it installs the SDK. The SDK consists of a cross-development toolchain, a set of libraries and headers, and an SDK environment setup script. Running this installer essentially sets up your cross-development environment. You can think of the cross-toolchain as the *host* part because it runs on the SDK machine. You can think of the libraries and headers as the *target* part because they are built for the target hardware. The environment setup script is added so that you can initialize the environment before using the tools.

Two different SDK types

All the output files for an SDK are written to the `deploy/sdk` folder inside the *Build Directory* as shown in Figure 14.1. Depending on the type of SDK, several variables exist that help configure these files. For a side-by-side comparison of main features supported for an *Extensible SDK* as compared to a *Standard SDK*, see the Introduction chapter in the Yocto Project Application Development and the Extensible Software Development Kit (eSDK).

Table 14.1: Features summary for the two SDK types

Feature	*Standard SDK*	*Extensible SDK*
Toolchain	Yes	Yes[1]
Debugger	Yes	Yes[1]
Size	100+ MB	1+ GB (or 300+ MB for minimal w/toolchain)
devtool	No	Yes
Build Images	No	Yes
Updateable	No	Yes
Managed Sysroot[2]	No	Yes
Installed Packages	No[3]	Yes[4]
Construction	Packages	Shared State

[1] *Extensible SDK* contains the toolchain and debugger if SDK_EXT_TYPE is `full` or SDK_INCLUDE_TOOLCHAIN is 1, **which is the default**.
[2] Sysroot is managed through the use of the **`devtool`** command. Thus, it is less likely that you will **corrupt your SDK sysroot** when you try to add additional libraries.
[3] You can add runtime package management to the *Standard SDK* but it **is not supported by default**.
[4] You **must build and make the Shared State Cache available** to *Extensible SDK* users for *packages* you want to enable users to install.

Extensible SDK

```
bitbake -c populate_sdk_ext imagename
```

The Yocto Project *Extensible SDK* (eSDK) has tools that allow you to easily add new applications and libraries to an image, modify the source of an existing component and test changes on the target hardware. The main benefit over the *Standard SDK* is improved team workflow due to tighter integration with the OpenEmbedded build system and have access to developer tools. For a detailed description see the Using the Extensible SDK chapter in the Yocto Project Application Development and the Extensible Software Development Kit (eSDK). News about development can read on the Yocto Project Wiki: Extensible SDK.

The following list shows the variables associated with an *Extensible SDK*:

- DEPLOY_DIR: Points to the *Deploy Directory* inside the *Build Directory*.
- SDK_EXT_TYPE: Controls whether or not shared state artifacts are copied into the *Extensible SDK*. By default, all required shared state artifacts are copied into the SDK.
- SDK_INCLUDE_PKGDATA: Specifies whether or not package data is included in the *Extensible SDK* for all recipes in the *world* target.
- SDK_INCLUDE_TOOLCHAIN: Specifies whether or not the toolchain is included when building the *Extensible SDK*.
- SDK_LOCAL_CONF_WHITELIST: A list of variables allowed through from the build system configuration into the *Extensible SDK* configuration.
- SDK_LOCAL_CONF_BLACKLIST: A list of variables not allowed through from the build system configuration into the *Extensible SDK* configuration.
- SDK_INHERIT_BLACKLIST: A list of classes to remove from the INHERIT value globally within the *Extensible SDK* configuration.

See also:

- Yocto Project Wiki: Application Development with Extensible SDK
- Yocto Project Wiki: Extensible SDK

Standard SDK

```
bitbake -c populate_sdk imagename
```

The Standard SDK provides a cross-development toolchain and libraries tailored to the contents of a specific image. You would use the *Standard SDK* if you want a more traditional toolchain experience as compared to the *Extensible SDK*. For a detailed description see the Using the Standard SDK chapter in the Yocto Project Application Development and the Extensible Software Development Kit (eSDK). Some use case scenarios can read on the Yocto Project Wiki: SDK Generator.

This next list, shows the variables associated with a *Standard SDK*:

- DEPLOY_DIR: Points to the deploy directory.
- SDKMACHINE: Specifies the architecture of the machine on which the cross-development tools are run to create packages for the target hardware.

- SDKIMAGE_FEATURES: Lists the features to include in the *target* part of the SDK.
- TOOLCHAIN_HOST_TASK: Lists packages that make up the host part of the SDK (i.e. the part that runs on the SDKMACHINE). When you use BitBake to create the SDK, a set of default packages apply. This variable allows you to add more packages.
- TOOLCHAIN_TARGET_TASK: Lists packages that make up the target part of the SDK (i.e. the part built for the target hardware).
- SDKPATH: Defines the default SDK installation path offered by the installation script.
- SDK_HOST_MANIFEST: Lists all the installed packages that make up the host part of the SDK. This variable also plays a minor role for *Extensible SDK* development as well. However, it is mainly used for the *Standard SDK*.
- SDK_TARGET_MANIFEST: Lists all the installed packages that make up the target part of the SDK. This variable also plays a minor role for *Extensible SDK* development as well. However, it is mainly used for the *Standard SDK*.

See also:

- Yocto Project Wiki: Application Development with Legacy SDK
- Yocto Project Wiki: TipsAndTricks/Cmake,Eclipse, and SDKS

15

Final Closer Look

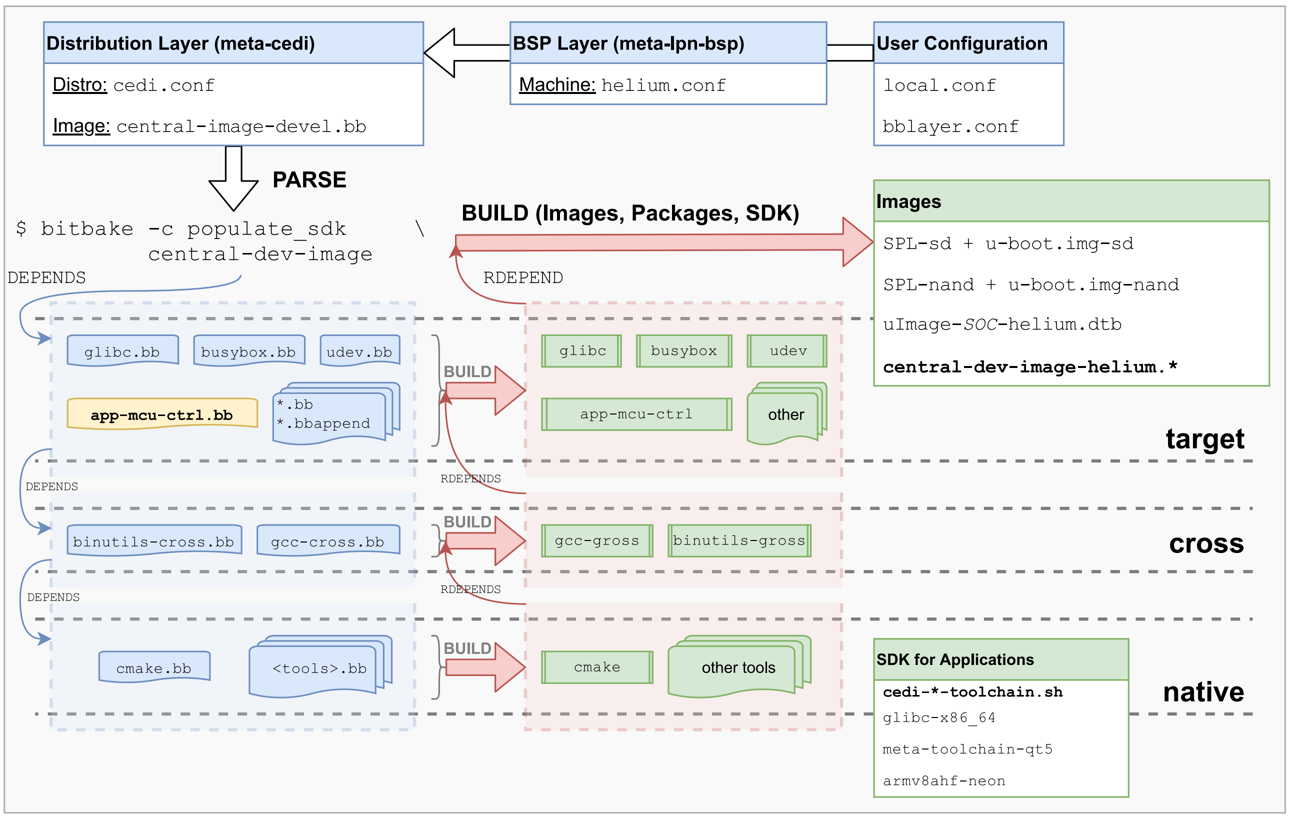


Figure 15.1: The Canadian Cross (reduced for host and target only)

16 Read-The-Docs

16.1 Documentations

- The project documents: https://www.yoctoproject.org/documentation
  - Yocto Project Quick Build
  - Yocto Project Overview and Concepts Manual
  - Yocto Project Reference Manual
  - Yocto Project Development Tasks Manual
  - Yocto Project Board Support Package Developer's Guide
  - Yocto Project Linux Kernel Development Manual
  - Yocto Project Profiling and Tracing Manual
  - Yocto Project Test Environment Manual
  - Yocto Project Application Development and the Extensible Software Development Kit (eSDK)
  - BitBake User Manual
- The project classroom: https://www.yoctoproject.org/learn
- The project Wiki-pages: https://wiki.yoctoproject.org/
  - Yocto Project Wiki: YP DevDay Austin 2020
  - Yocto Project Wiki: DevDay San Diego 2019
  - Yocto Project Wiki: DevDay Portland 2018 and Edinburgh 2018
  - Yocto Project Wiki: DevDay US 2017 and Prague 2017
- Bootlin (former Free Electrons) Material: https://bootlin.com/docs/
  - Training Materials: https://bootlin.com/doc/training/yocto/
  - Conference Materials: https://bootlin.com/pub/conferences/

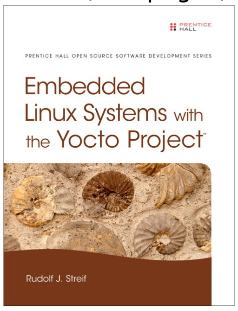
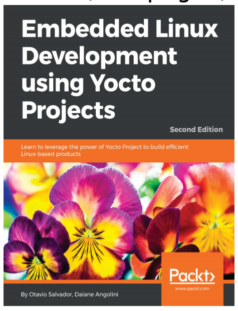
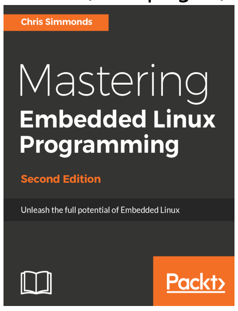
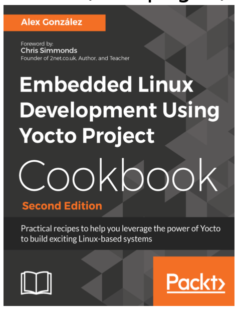
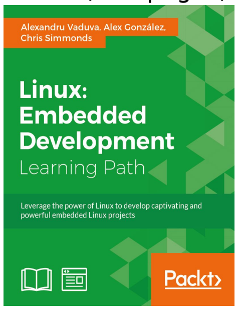
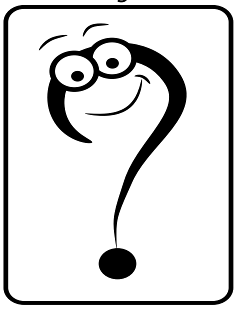
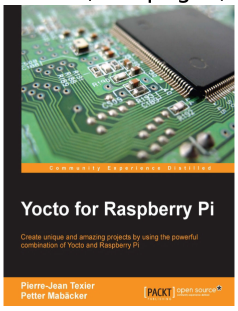
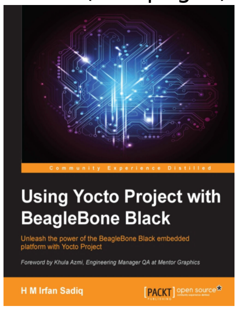
16.2 Presentations

- *Jeffrey Osier-Mixon*, *Kevin McGrath*: **Yocto Project and Embedded OS (2015)** [OMM15]
- *Rudolf J. Streif*: **Introduction to the Yocto Project (2015)** [Str15] ▯ [Str16]
- *Otavio Salvador*: **Yocto Training : Your next development platform for Embedded Linux (2013)** [Sal13] ▯ [SA17]
- *Alex González*: **Introduction to Yocto (2012)** [Gon12] ▯ [Gon18][VGS16]
- *Mark Hatle*, *Bruce Ashfield*: **Introduction to the Yocto Project (2012)** [HA12]
- *Yi-Hsiu Hsu*: **Yocto Project Introduction (2015)** [Hsu15]
- *Yen-Chin Lee*: **Build your own Embedded Linux distributions by Yocto Project (2015)** [Lee15]
- *Nicolas Dechesne*, *Riku Voipio*, *Trevor Woerner*: **OpenEmbedded and Yocto Introduction (2013)** [DVW13]
- *Marcelo A.L. Sanz*: **Yocto Project Open Source Build System and Collaboration Initiative (2012)** [San12]

16.3 Books

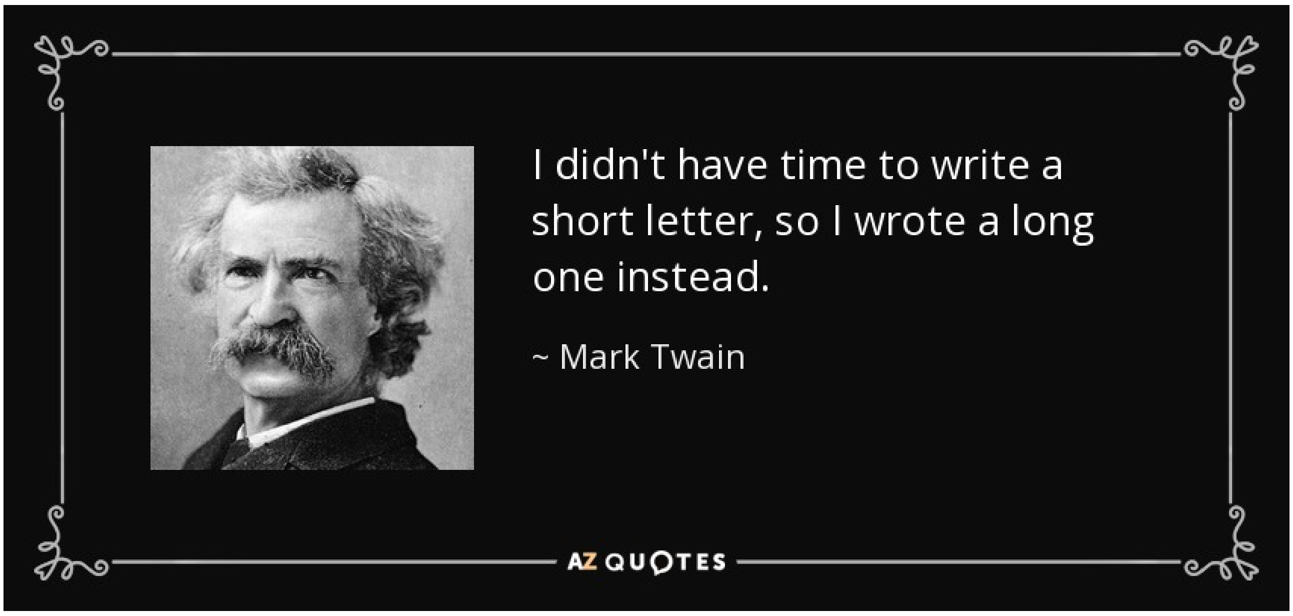
- https://www.yoctoproject.org/learn/books/
- https://old.yoctoproject.org/blogs/jefro/2016/yocto-project-books

Table 16.1: Books about and with Yocto

2016 (480 pages)	**2017** (162 pages)	**2017** (478 pages)	**2018** (456 pages)
			
978-0-13344-324-0	*978-1-78847-046-9*	*978-1-78728-328-2*	*978-1-78839-921-0*
[Str16]	[SA17]	[Sim17]	[Gon18]
2016 (989 pages)	*missing one?*	**2016** (214 pages)	**2015** (144 pages)
			
978-1-78712-445-5		*978-1-78528-195-2*	*978-1-78528-973-6*
[VGS16]		[TM15]	[Sad15]

17 Thank You

Todo: complete chapter



I didn't have time to write a short letter, so I wrote a long one instead.

~ Mark Twain

AZ QUOTES

A General Information

A.1 License

Listing 1.1: License text of "Learning Yocto"

(continues on next page)

(continued from previous page)

```
   recast, transformed, or adapted including in any form recognizably
   derived from the original, except that a work that constitutes a
   Collection will not be considered an Adaptation for the purpose of
   this License. For the avoidance of doubt, where the Work is a musical
   work, performance or phonogram, the synchronization of the Work in
   timed-relation with a moving image (”synching”) will be considered an
   Adaptation for the purpose of this License.
b. ”Collection” means a collection of literary or artistic works, such as
   encyclopedias and anthologies, or performances, phonograms or
   broadcasts, or other works or subject matter other than works listed
   in Section 1(f) below, which, by reason of the selection and
   arrangement of their contents, constitute intellectual creations, in
   which the Work is included in its entirety in unmodified form along
   with one or more other contributions, each constituting separate and
   independent works in themselves, which together are assembled into a
   collective whole. A work that constitutes a Collection will not be
   considered an Adaptation (as defined below) for the purposes of this
   License.
c. ”Creative Commons Compatible License” means a license that is listed
   at https://creativecommons.org/compatiblelicenses that has been
   approved by Creative Commons as being essentially equivalent to this
   License, including, at a minimum, because that license: (i) contains
   terms that have the same purpose, meaning and effect as the License
   Elements of this License; and, (ii) explicitly permits the relicensing
   of adaptations of works made available under that license under this
   License or a Creative Commons jurisdiction license with the same
   License Elements as this License.
d. ”Distribute” means to make available to the public the original and
   copies of the Work or Adaptation, as appropriate, through sale or
   other transfer of ownership.
e. ”License Elements” means the following high-level license attributes
   as selected by Licensor and indicated in the title of this License:
   Attribution, ShareAlike.
f. ”Licensor” means the individual, individuals, entity or entities that
   offer(s) the Work under the terms of this License.
g. ”Original Author” means, in the case of a literary or artistic work,
   the individual, individuals, entity or entities who created the Work
   or if no individual or entity can be identified, the publisher; and in
   addition (i) in the case of a performance the actors, singers,
   musicians, dancers, and other persons who act, sing, deliver, declaim,
   play in, interpret or otherwise perform literary or artistic works or
   expressions of folklore; (ii) in the case of a phonogram the producer
   being the person or legal entity who first fixes the sounds of a
   performance or other sounds; and, (iii) in the case of broadcasts, the
   organization that transmits the broadcast.
h. ”Work” means the literary and/or artistic work offered under the terms
   of this License including without limitation any production in the
   literary, scientific and artistic domain, whatever may be the mode or
   form of its expression including digital form, such as a book,
   pamphlet and other writing; a lecture, address, sermon or other work
```

(continues on next page)

(continued from previous page)

```
    of the same nature; a dramatic or dramatico-musical work; a
    choreographic work or entertainment in dumb show; a musical
    composition with or without words; a cinematographic work to which are
    assimilated works expressed by a process analogous to cinematography;
    a work of drawing, painting, architecture, sculpture, engraving or
    lithography; a photographic work to which are assimilated works
    expressed by a process analogous to photography; a work of applied
    art; an illustration, map, plan, sketch or three-dimensional work
    relative to geography, topography, architecture or science; a
    performance; a broadcast; a phonogram; a compilation of data to the
    extent it is protected as a copyrightable work; or a work performed by
    a variety or circus performer to the extent it is not otherwise
    considered a literary or artistic work.
 i. ”You” means an individual or entity exercising rights under this
    License who has not previously violated the terms of this License with
    respect to the Work, or who has received express permission from the
    Licensor to exercise rights under this License despite a previous
    violation.
 j. ”Publicly Perform” means to perform public recitations of the Work and
    to communicate to the public those public recitations, by any means or
    process, including by wire or wireless means or public digital
    performances; to make available to the public Works in such a way that
    members of the public may access these Works from a place and at a
    place individually chosen by them; to perform the Work to the public
    by any means or process and the communication to the public of the
    performances of the Work, including by public digital performance; to
    broadcast and rebroadcast the Work by any means including signs,
    sounds or images.
 k. ”Reproduce” means to make copies of the Work by any means including
    without limitation by sound or visual recordings and the right of
    fixation and reproducing fixations of the Work, including storage of a
    protected performance or phonogram in digital form or other electronic
    medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce,
limit, or restrict any uses free from copyright or rights arising from
limitations or exceptions that are provided for in connection with the
copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License,
Licensor hereby grants You a worldwide, royalty-free, non-exclusive,
perpetual (for the duration of the applicable copyright) license to
exercise the rights in the Work as stated below:

 a. to Reproduce the Work, to incorporate the Work into one or more
    Collections, and to Reproduce the Work as incorporated in the
    Collections;
 b. to create and Reproduce Adaptations provided that any such Adaptation,
    including any translation in any medium, takes reasonable steps to
    clearly label, demarcate or otherwise identify that changes were made
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
   must, to the extent practicable, remove from the Collection any credit
   as required by Section 4(c), as requested. If You create an
   Adaptation, upon notice from any Licensor You must, to the extent
   practicable, remove from the Adaptation any credit as required by
   Section 4(c), as requested.
b. You may Distribute or Publicly Perform an Adaptation only under the
   terms of: (i) this License; (ii) a later version of this License with
   the same License Elements as this License; (iii) a Creative Commons
   jurisdiction license (either this or a later license version) that
   contains the same License Elements as this License (e.g.,
   Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible
   License. If you license the Adaptation under one of the licenses
   mentioned in (iv), you must comply with the terms of that license. If
   you license the Adaptation under the terms of any of the licenses
   mentioned in (i), (ii) or (iii) (the ”Applicable License”), you must
   comply with the terms of the Applicable License generally and the
   following provisions: (I) You must include a copy of, or the URI for,
   the Applicable License with every copy of each Adaptation You
   Distribute or Publicly Perform; (II) You may not offer or impose any
   terms on the Adaptation that restrict the terms of the Applicable
   License or the ability of the recipient of the Adaptation to exercise
   the rights granted to that recipient under the terms of the Applicable
   License; (III) You must keep intact all notices that refer to the
   Applicable License and to the disclaimer of warranties with every copy
   of the Work as included in the Adaptation You Distribute or Publicly
   Perform; (IV) when You Distribute or Publicly Perform the Adaptation,
   You may not impose any effective technological measures on the
   Adaptation that restrict the ability of a recipient of the Adaptation
   from You to exercise the rights granted to that recipient under the
   terms of the Applicable License. This Section 4(b) applies to the
   Adaptation as incorporated in a Collection, but this does not require
   the Collection apart from the Adaptation itself to be made subject to
   the terms of the Applicable License.
c. If You Distribute, or Publicly Perform the Work or any Adaptations or
   Collections, You must, unless a request has been made pursuant to
   Section 4(a), keep intact all copyright notices for the Work and
   provide, reasonable to the medium or means You are utilizing: (i) the
   name of the Original Author (or pseudonym, if applicable) if supplied,
   and/or if the Original Author and/or Licensor designate another party
   or parties (e.g., a sponsor institute, publishing entity, journal) for
   attribution (”Attribution Parties”) in Licensor's copyright notice,
   terms of service or by other reasonable means, the name of such party
   or parties; (ii) the title of the Work if supplied; (iii) to the
   extent reasonably practicable, the URI, if any, that Licensor
   specifies to be associated with the Work, unless such URI does not
   refer to the copyright notice or licensing information for the Work;
   and (iv) , consistent with Ssection 3(b), in the case of an
   Adaptation, a credit identifying the use of the Work in the Adaptation
   (e.g., ”French translation of the Work by Original Author,” or
   ”Screenplay based on original Work by Original Author”). The credit
```

(continues on next page)

(continued from previous page)

```
    required by this Section 4(c) may be implemented in any reasonable
    manner; provided, however, that in the case of a Adaptation or
    Collection, at a minimum such credit will appear, if a credit for all
    contributing authors of the Adaptation or Collection appears, then as
    part of these credits and in a manner at least as prominent as the
    credits for the other contributing authors. For the avoidance of
    doubt, You may only use the credit required by this Section for the
    purpose of attribution in the manner set out above and, by exercising
    Your rights under this License, You may not implicitly or explicitly
    assert or imply any connection with, sponsorship or endorsement by the
    Original Author, Licensor and/or Attribution Parties, as appropriate,
    of You or Your use of the Work, without the separate, express prior
    written permission of the Original Author, Licensor and/or Attribution
    Parties.
 d. Except as otherwise agreed in writing by the Licensor or as may be
    otherwise permitted by applicable law, if You Reproduce, Distribute or
    Publicly Perform the Work either by itself or as part of any
    Adaptations or Collections, You must not distort, mutilate, modify or
    take other derogatory action in relation to the Work which would be
    prejudicial to the Original Author's honor or reputation. Licensor
    agrees that in those jurisdictions (e.g. Japan), in which any exercise
    of the right granted in Section 3(b) of this License (the right to
    make Adaptations) would be deemed to be a distortion, mutilation,
    modification or other derogatory action prejudicial to the Original
    Author's honor and reputation, the Licensor will waive or not assert,
    as appropriate, this Section, to the fullest extent permitted by the
    applicable national law, to enable You to reasonably exercise Your
    right under Section 3(b) of this License (right to make Adaptations)
    but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR
OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY
KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE,
INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY,
FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF
LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS,
WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION
OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE
LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR
ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES
ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS
BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

 a. This License and the rights granted hereunder will terminate
```

(continues on next page)

(continued from previous page)

```
   automatically upon any breach by You of the terms of this License.
   Individuals or entities who have received Adaptations or Collections
   from You under this License, however, will not have their licenses
   terminated provided such individuals or entities remain in full
   compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will
   survive any termination of this License.
 b. Subject to the above terms and conditions, the license granted here is
   perpetual (for the duration of the applicable copyright in the Work).
   Notwithstanding the above, Licensor reserves the right to release the
   Work under different license terms or to stop distributing the Work at
   any time; provided, however that any such election will not serve to
   withdraw this License (or any other license that has been, or is
   required to be, granted under the terms of this License), and this
   License will continue in full force and effect unless terminated as
   stated above.

8. Miscellaneous

 a. Each time You Distribute or Publicly Perform the Work or a Collection,
   the Licensor offers to the recipient a license to the Work on the same
   terms and conditions as the license granted to You under this License.
 b. Each time You Distribute or Publicly Perform an Adaptation, Licensor
   offers to the recipient a license to the original Work on the same
   terms and conditions as the license granted to You under this License.
 c. If any provision of this License is invalid or unenforceable under
   applicable law, it shall not affect the validity or enforceability of
   the remainder of the terms of this License, and without further action
   by the parties to this agreement, such provision shall be reformed to
   the minimum extent necessary to make such provision valid and
   enforceable.
 d. No term or provision of this License shall be deemed waived and no
   breach consented to unless such waiver or consent shall be in writing
   and signed by the party to be charged with such waiver or consent.
 e. This License constitutes the entire agreement between the parties with
   respect to the Work licensed here. There are no understandings,
   agreements or representations with respect to the Work not specified
   here. Licensor shall not be bound by any additional provisions that
   may appear in any communication from You. This License may not be
   modified without the mutual written agreement of the Licensor and You.
 f. The rights granted under, and the subject matter referenced, in this
   License were drafted utilizing the terminology of the Berne Convention
   for the Protection of Literary and Artistic Works (as amended on
   September 28, 1979), the Rome Convention of 1961, the WIPO Copyright
   Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996
   and the Universal Copyright Convention (as revised on July 24, 1971).
   These rights and subject matter take effect in the relevant
   jurisdiction in which the License terms are sought to be enforced
   according to the corresponding provisions of the implementation of
   those treaty provisions in the applicable national law. If the
   standard suite of rights granted under applicable copyright law
```

(continues on next page)

(continued from previous page)

A.2 Credits

Listing 1.2: Authors contributing to “Learning Yocto” development

```
________________________________________________________________________________

Stephan Linz <linz@li-pro.net>                    Technical Writer
________________________________________________________________________________
```

Listing 1.3: Supporter contributing to “Learning Yocto” development

```
________________________________________________________________________________

JENETRIC GmbH                                     Commissioned Projects
Method Park Holding AG                            Commissioned Projects
Navimatix GmbH                                    Commissioned Projects
Yocto Project                                     Manuals and Graphics
________________________________________________________________________________
```

B Glossary

B.1 Common Terms

Docutils Docutils is an open-source text processing system for processing plaintext documentation into useful formats, such as *HTML*, *LaTeX*, man-pages, open-document or *XML*. It includes *reStructuredText*, the easy to read, easy to use, what-you-see-is-what-you-get plaintext markup language.

See also:

- English Wikipedia: reStructuredText

LaTeX LaTeX is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing. LaTeX uses the *TeX* typesetting program for formatting its output, and is itself written in the *TeX* macro language.

See also:

- English Wikipedia: LaTeX

Pybtex Pybtex is a *BibTeX*-compatible bibliography processor written in *Python*. Pybtex aims to be 100% compatible with *BibTeX*. It accepts the same command line options, fully supports *BibTeX*'s `.bst` styles and produces byte-identical output. Additionally, Pybtex is Unicode aware and Pybtex supports bibliography formats other than *BibTeX*: BibTeXML (BibTeX as XML) and YAML (YAML Ain't Markup Language).

PyEnchant PyEnchant is a *Python* binding for *Enchant*.

Pygments Pygments is a generic syntax highlighter written in *Python* which supports a wide range of over 500 languages with related lexers and other text formats and is ready for new languages and formats added easily.

ReportLab ReportLab Toolkit is an Open Source *Python* library for generating *PDF*s and graphics.

See also:

- https://www.reportlab.com/opensource/
- https://www.reportlab.com/dev/docs/
- https://hg.reportlab.com/hg-public/
- https://pypi.org/project/reportlab/

reStructuredText reStructuredText (**RST**, **ReST**, or **reST**) is a file format for textual data used primarily in the *Python* programming language community for technical documentation. It is part of the *Docutils* project of the *Python* Doc-SIG (Documentation Special Interest Group).

See also:

- English Wikipedia: reStructuredText

Sphinx Sphinx is a documentation generator written and used by the *Python* community. It is written in *Python*, and also used in other environments. Sphinx converts *reStructuredText* files into *HTML* websites and other formats including PDF, EPub, Texinfo and man.

reStructuredText is extensible, and Sphinx exploits its extensible nature through a number of extensions–for auto generating documentation from source code, writing mathematical notation or highlighting source code, etc.

See also:

- English Wikipedia: Sphinx (documentation generator)
- [Has19]

B.2 Programming Languages

C C is a general-purpose, imperative procedural computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system. It was designed to be compiled to provide low-level access to memory and language constructs that map efficiently to machine instructions, all with minimal runtime support. C has been standardized by the ANSI (American National Standards Institute) *X3J11* since 1989 (**ANSI C**) and by the ISO (International Organization for Standardization)/IEC (International Electrotechnical Commission) JTC1/SC22/WG14 (**ISO C**).

See also:

- English Wikipedia: C (programming language)
- English Wikipedia: Compatibility of C and C++
- English Wikipedia: C18 (C standard revision): standard ratified in 2018 as **ISO/IEC 9899:2018**
- English Wikipedia: C11 (C standard revision): standard ratified in 2011 as *ISO/IEC 9899:2011*
- English Wikipedia: C99: standard ratified in 1999 as *ISO/IEC 9899:1999*
- English Wikipedia: C95 (C version): Amendment 1 ratified in 1995 as *ISO/IEC 9899:1990/AMD1:1995*
- English Wikipedia: C90 (C version): standard ratified in 1990 as *ISO/IEC 9899:1990*
- English Wikipedia: C89 (C version): standard ratified in 1989 as *ANSI X3.159-1989*

C++ C++ is a general-purpose programming language as an extension of the C programming language, or *"C with Classes"*. Modern C++ implementations now has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. C++ is standardized by the ISO/IEC JTC1/SC22/WG14 since 1998.

See also:

- English Wikipedia: C++
- English Wikipedia: Compatibility of C and C++
- English Wikipedia: C++17: standard ratified in 2017 as **ISO/IEC 14882:2017**
- English Wikipedia: C++14: standard ratified in 2014 as *ISO/IEC 14882:2014*
- English Wikipedia: C++11: standard ratified in 2011 as *ISO/IEC 14882:2011*
- English Wikipedia: C++03: standard ratified in 2003 as *ISO/IEC 14882:2003*
- initially standardized in 1998 as *ISO/IEC 14882:1998*

ES (ECMAScript)

ECMAScript ES is a general-purpose programming language, standardized by Ecma International since 1997 according to the document ECMA-262. It is a *JavaScript* standard meant to ensure the interoperability of Web pages across different Web browsers. ES is standardized by the ISO/IEC JTC1/SC22 since 1998.

See also:

- English Wikipedia: ECMAScript
- English Wikipedia: ECMAScript engine

- English Wikipedia: List of ECMAScript engines
- ES Edition 11: standard ratified in 2020 as **ECMA-262-11:2020**
- ES Edition 5.1: standard ratified in 2011 as *ISO/IEC 16262:2011*
- ES Edition 2: initially standardized in 1998 as *ISO/IEC 16262:1998*

JS (JavaScript)

JavaScript JS is a programming language that conforms to the *ECMAScript* specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside *HTML* and *CSS*, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it.

See also:

- English Wikipedia: JavaScript
- English Wikipedia: JavaScript engine
- English Wikipedia: List of JavaScript engines

Python Python is an interpreted, high-level and general-purpose programming language. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, a free and open-source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

CPython is the reference implementation of Python. It is written in *C*, meeting the *C*89 standard with several select *C*99 features. Python's development is conducted largely through the PEP (Python Enhancement Proposal) process, the primary mechanism for proposing major new features, collecting community input on issues and documenting Python design decisions. Python coding style is covered in **PEP 8**.

See also:

- English Wikipedia: Python (programming language)
- English Wikipedia: CPython
- [Swe19]
- [Swe20]

B.3 Technologies

BibTeX BibTeX is a widely used bibliography management tool in *LaTeX*, with BibTeX the bibliography entries are kept in a separate file and then imported into the main document.

See also:

- English Wikipedia: BibTeX

CORBA (Common Object Request Broker Architecture)

MOF (Meta-Object Facility) MOF is an OMG (Object Management Group) standard for model-driven engineering. Its purpose is to provide a type system for entities in the *CORBA* architecture and a set of interfaces through which those types can be created and manipulated.

See also:

- *OCL* and *QVT*
- English Wikipedia: Meta-Object Facility
- MOF 2.5.1: initially standardized in 2016 **OMG Meta Object Facility Core Specification 2.5.1 2016/10/01**
- MOF 2.5: initially standardized in 2015 *OMG Meta Object Facility Core Specification 2.5 2015/06/05*
- ISO 19508:2014: **MOF 2.4.2** standard formally published in 2014 **ISO/IEC 19508:2014(E) 2014/04/05**
- MOF 2.4.2: initially standardized in 2014 *OMG Meta Object Facility Core Specification 2.4.2 2014/04/03*
- MOF 2.4.1: initially standardized in 2013 *OMG Meta Object Facility Core Specification 2.4.1 2013/06/01*
- MOF 2.0: initially standardized in 2006 *Meta Object Facility Core Specification 2.0 2006/01/01*
- ISO 19502:2005: *MOF 1.4.1* standard formally published in 2005 *ISO/IEC 19502:2005(E) 2005/05/05*
- MOF 1.4: initially standardized in 2002 *Meta Object Facility Specification 1.4 2002/04/03*

CSS (Cascading Style Sheets) CSS is a style sheet language used for describing the presentation of a document written in a markup language like *HTML*. CSS is a cornerstone technology of the WWW (World Wide Web), alongside *HTML* and *JavaScript*. In addition to *HTML*, other markup languages support the use of CSS including plain *XML* and *SVG*. The CSS specifications is standardized by the W3C (World Wide Web Consortium)/TR/CSS since 1996.

See also:

- English Wikipedia: CSS
- CSS 2.1: standard ratified in 2011 **W3C REC-CSS2-20110607**
- CSS 2.0: standard ratified in 1998 *W3C REC-CSS2-19980512*
- CSS 1.0: initially standardized in 1996 *W3C REC-CSS1-961217*

Enchant Enchant is a free software project developed as part of the AbiWord word processor with the aim of unifying access to the various existing spell-checker software.

See also:

- English Wikipedia: Enchant (software)

HTML (Hypertext Markup Language) HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as *CSS* and scripting languages such as *JavaScript*. The HTML specifications is standardized by the W3C/TR/HTML since 1997 and ISO/IEC JTC1/SC34 since 1998.

See also:

- English Wikipedia: HTML
- English Wikipedia: HTML5: latest live standard was released in 2017 **W3C REC-HTML52-20171214**
- English Wikipedia: HTML4: standard ratified in 1999 *W3C REC-HTML40* and 2000 *ISO/IEC 15445:2000*
- English Wikipedia: HTML3: standard ratified in 1997 *W3C REC-HTML32*
- English Wikipedia: HTML2: initially standardized in 1995 as *RFC 1866* (**RFC 1866**)

OCL (Object Constraint Language) OCL is a declarative language describing rules applying to *UML* models and is now part of the *UML* standard but as separate document. Initially, OCL was merely a formal specification language extension for *UML*. OCL may now be used with any *MOF* meta-model, including *UML*, and is a precise text language that provides constraint and object query expressions on any such kind of meta-model that cannot otherwise be expressed by diagrammatic notation. OCL is a key component of the new OMG standard recommendation for transforming models, the *QVT* specification.

See also:

- *UML*, *MOF* and *QVT*
- English Wikipedia: Object Constraint Language
- OCL 2.4: standard ratified in 2014 **Object Constraint Language 2.4 2014/02/03**
- ISO 19507:2012: **OCL 2.3.1** standard formally published in 2012 **ISO/IEC 19507:2012(E) 2012/05/09**
- OCL 2.3.1: standard ratified in 2011 *Object Constraint Language 2.3.1 2012/01/01*
- OCL 2.2: standard ratified in 2010 *Object Constraint Language 2.2 2010/02/01*
- OCL 2.0: standard ratified in 2006 *Object Constraint Language 2.0 2006/05/01*
- UML 1.3 (*Chapter* 7): initially standardized in 2000

PDF (Portable Document Format) PDF is a file format developed by Adobe in 1993 to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems. Based on the PS (PostScript) language, each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, vector graphics, raster images and other information needed to display it. PDF is standardized by the ISO TC171/SC2/WG8 since 2008, and no longer requires any royalties for its implementation.

ISO standardized subsets of PDF:

- English Wikipedia: PDF/X: since 2001, series of *ISO 15929* and *ISO 15930* standards
- English Wikipedia: PDF/A: since 2005, series of *ISO 19005* standards
- English Wikipedia: PDF/E: since 2008, series of *ISO 24517*
- English Wikipedia: PDF/VT: since 2010, *ISO 16612-2*
- English Wikipedia: PDF/UA: since 2012, *ISO 14289-1*

See also:

- English Wikipedia: PDF
- English Wikipedia: History of the Portable Document Format (PDF)
- PDF 2.0: standard ratified in 2017 as **ISO 32000-2:2017**
- PDF 1.7: initially standardized in 2008 as *ISO 32000-1:2008*

QVT (Query/View/Transformation) QVT is a standard set of languages for model transformation defined by the OMG.

See also:

- *OCL* and *MOF*
- English Wikipedia: QVT
- QVT 1.3: standard ratified in 2016 **MOF 2.0 Query/View/Transformation Specification 1.3 2016/06/03**
- QVT 1.2: standard ratified in 2015 *MOF 2.0 Query/View/Transformation Specification 1.2 2015/02/01*
- QVT 1.1: standard ratified in 2011 *MOF 2.0 Query/View/Transformation Specification 1.1 2011/01/01*
- QVT 1.0: initially standardized in 2008 *MOF 2.0 Query/View/Transformation Specification 1.0 2008/04/03*

SVG (Scalable Vector Graphics) SVG is an *XML*-based vector image format for two-dimensional graphics with support for interactivity and animation. The SVG specification is standardized by the W3C/TR/SVG since 1999 as an open standard.

SVG drawings can be dynamic and interactive. Time-based modifications to the elements can be described in SMIL (Synchronized Multimedia Integration Language), or can be programmed in a scripting language (e.g. *ECMAScript* or *JavaScript*). The W3C explicitly recommends SMIL as the standard for animation in SVG.

See also:

- English Wikipedia: SVG
- SVG 2.0: latest standard draft was released in 2020
- SVG 1.1 Second Edition: standard ratified in 2011 **W3C REC-SVG11-20110816**
- SVG 1.1: standard ratified in 2003 *W3C REC-SVG11-20030114*
- SVG 1.0: initially standardized in 2001 *W3C REC-SVG-20010904*

SysML (Systems Modeling Language)

BPMN (Business Process Model and Notation)

UML (Unified Modeling Language) UML is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. In 1997, UML was adopted as a standard by the OMG, and has been managed by this organization ever since. In 2005, UML was also published by ISO as an approved standard.

In 2007 with the release of UML 2.1.2, two new significant specifications for certain technology areas have split off. These are: *SysML*, *BPMN*.

See also:

- *OCL*, *MOF* and *QVT*
- English Wikipedia: Unified Modeling Language
- UML 2.5.1: standard adopted from 2.5 in 2017 **OMG Unified Modeling Language 2.5.1 2017/12/05**
- UML 2.5: standard released in 2012 and ratified in 2015 *OMG Unified Modeling Language 2.5 2015/03/01*
- ISO 19505-1:2012 and ISO 19505-2:2012: **UML 2.4.1** standard formally published in 2012 **ISO/IEC 19505-1:2012(E) 2012/05/06** and **ISO/IEC 19505-2:2012(E) 2012/05/07**
- UML 2.4.1 and UMLDI 1.0: standard ratified in 2011 *OMG UML Superstructure Specification 2.4.1 2011/08/06*, *OMG UML Infrastructure Specification 2.4.1 2011/08/05* and *OMG UML Diagram Interchange 1.0 2006/04/04*
- UML 2.4 and UMLDI 1.0: standard ratified in 2011 *OMG UML Superstructure Specification 2.4 2010/11/14*, *OMG UML Infrastructure Specification 2.4 2010/11/16* and *OMG UML Diagram Interchange 1.0 2006/04/04*
- UML 2.3 and UMLDI 1.0: standard ratified in 2010 *OMG UML Superstructure Specification 2.3 2010/05/05*, *OMG UML Infrastructure Specification 2.3 2010/05/03* and *OMG UML Diagram Interchange 1.0 2006/04/04*
- UML 2.2 and UMLDI 1.0: standard ratified in 2009 *OMG UML Superstructure Specification 2.2 2009/02/02*, *OMG UML Infrastructure Specification 2.2 2009/02/04* and *OMG UML Diagram Interchange 1.0 2006/04/04*
- UML 2.1.2 and UMLDI 1.0: standard ratified in 2007 *OMG UML Superstructure Specification 2.1.2 2007/11/02*, *OMG UML Infrastructure Specification 2.1.2 2007/11/04* and *OMG UML Diagram Interchange 1.0 2006/04/04*
- UML 2.1.1 and UMLDI 1.0: standard ratified in 2007 *OMG UML Superstructure Specification 2.1.1 2007/02/05*, *OMG UML Infrastructure Specification 2.1.1 2007/02/06* and *OMG UML Diagram Interchange 1.0 2006/04/04*
- UML 2.0: standard ratified in 2005 *OMG UML Superstructure Specification 2.0 2005/07/04* and *OMG UML Infrastructure Specification 2.0 2005/07/05*
- ISO 19501:2005: *UML 1.4.2* standard formally published in 2005 *ISO/IEC 19501:2005(E) 2005/04/01*
- UML 1.5: standard ratified in 2003 *OMG Unified Modeling Language Specification 1.5 2003/03/01*

- UML 1.4: standard ratified in 2001 *OMG Unified Modeling Language Specification 1.4 2001/09/07*
- UML 1.3: standard ratified in 2000 *OMG Unified Modeling Language Specification 1.3 2000/03/01*
- UML 1.2: standard ratified in 1999
- UML 1.1: initially standardized in 1997

TeX TeX is a computer language designed for use in typesetting system; in particular, for typesetting math and other technical material. It has been noted as one of the most sophisticated digital typographical systems and is also used for many other typesetting tasks, especially in the form of *LaTeX*, ConTeXt, and other macro packages.

See also:

- English Wikipedia: TeX

XML (Extensible Markup Language) XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures. Several schema systems exist to aid in the definition of XML-based languages. The XML specification is standardized by the W3C/TR/XML since 1998 as an open standard.

See also:

- English Wikipedia: XML
- XML 1.1 Second Edition: standard ratified in 2006 **W3C REC-XML11-20060816**
- XML 1.1: standard ratified in 2004 *W3C REC-XML11-20040204*
- XML 1.0 Fifth Edition: standard ratified in 2008 *W3C REC-XML-20081126*
- XML 1.0: initially standardized in 1998 *W3C REC-SVG-20010904*

Acronyms

This is a list of abbreviations and acronyms used in this document and will be kept up to date manually and is not generated automatically. There is no automatism that assists the maintenance of the items in this list. As long as there is no entry in the glossary for a commonly known abbreviation, it will also be listed here as an acronym.

generally

BSP	Board Support Package
CD	Continuous Deployment / Delivery
CI	Continuous Integration
distro	distribution (distro)
OE	OpenEmbedded
SCMP (Software Configuration Management Plan)	Software Configuration Management Plan
SCMS	Software Configuration Management System
SDK	Software Development Kit

specifically

not yet

Listings

1.1 License text of “Learning Yocto” 89
1.2 Authors contributing to “Learning Yocto” development 97
1.3 Supporter contributing to “Learning Yocto” development 97

List of Tables

1 Document Revisions of “Learning Yocto” 7
14.1 Features summary for the two SDK types 78
16.1 Books about and with Yocto 85

List of Figures

1.1 The Open Source Challenge 9
1.2 Example: UI Toolkits 10
1.3 Collaboration as a Solution 10

4.1 Yocto Project Deliveries [OMM15] 22
4.2 The Yocto Project Community [OMM15] 24

5.1 Yocto Build System Features and their interaction 29

6.1 Revision Control of your Embedded Linux System using Git repo 35

7.1 OpenEmbedded Build System Concepts as part of Yocto Build System Workflow 37
7.2 User Configuration as part of Yocto Build System Workflow 38
7.3 Metadata Layers as part of Yocto Build System Workflow 39
7.4 Sources as part of Yocto Build System Workflow 40
7.5 Package Feeds as part of Yocto Build System Workflow 41
7.6 BitBake Tool as part of Yocto WBuild System orkflow 42
7.7 Images as part of Yocto Build System Workflow 43
7.8 Application Development SDK as part of Yocto Build System Workflow 44

8.1 User Configuration with Details 45

9.1 Metadata Layers with Details 49
9.2 Metadata Layer Priorities (`BBFILE_PRIORITY_LayerName`) 52

10.1 Sources with Details 61

11.1 Package Feeds with Details 65

12.1 Source Fetching with Details 68
12.2 Patching with Details 69
12.3 Configuration, Compilation, and Staging with Details 70
12.4 Package Splitting with Details 71
12.5 Image Generation with Details 1/2 72
12.6 Image Generation with Details 2/2 73

12.7 SDK Generation with Details 74
13.1 Images with Details 75
14.1 Application Development SDK with Details 77
15.1 The Canadian Cross (reduced for host and target only) 81

List of Equations

List of Downloads

Note: *List of Downloads* is not fully supported for *LaTeX*. All entries in the list are not linked and can not be provided together with the document.

Legal Notice, Credits, and Contributions

- `LICENSE`
- `CREDITS`

List of Issues

Todo:

improve table formatting and colspec for *LaTeX*/*PDF*: (1) provide overwrite directions for tabular and longtable environments, (2) start with global colorization of table header and alternately colored rows

(See template folder below /home/travis/build/lipro/lpn-show-learning-yocto/source and add content in `preamble.tex.in`)

Bibliography

[DVW13] Nicolas Dechesne, Riku Voipio, and Trevor Woerner. Openembedded and yocto introduction. 2013. URL: https://www.slideshare.net/linaroorg/linaro-open-embeddedworkshopwednesday (visited on January 2021).

[Gon12] Alex González. Introduction to yocto. 2012. URL: https://www.slideshare.net/alexgonzalezgarcia/introduction-to-yocto (visited on January 2021).

[Gon18] Alex González. *Embedded Linux Development Using Yocto Project Cookbook : Practical recipes to help you leverage the power of Yocto to build exciting Linux-based systems*. Packt Publishing, Birmingham, United Kingdom, 2nd edition, 2018. ISBN 1788399218. URL: https://www.amazon.com/dp/1788399218 (visited on January 2021).

[Has19] Jan Ulrich Hasecke. *Software-Dokumentation mit Sphinx*. CreateSpace (was part of Amazon.com Inc.), today Kindle Direct Publishing (KDP), Seattle, United States of America, 2nd revised edition, 2019. ISBN 1793008779. URL: https://www.amazon.com/dp/1793008779 (visited on January 2021).

[HA12] Mark Hatle and Bruce Ashfield. Introduction to the yocto project. 2012. URL: https://old.yoctoproject.org/sites/default/files/elc-e_devday_introyocto_2.pdf (visited on January 2021).

[Hsu15] Yi-Hsiu Hsu. Yocto project introduction. 2015. URL: https://www.slideshare.net/YiHsiuHsu/yocto-project-introduction (visited on January 2021).

[Lee15] Yen-Chin Lee. Build your own embedded linux distributions by yocto project. 2015. URL: https://www.slideshare.net/coldnew/build-your-own-embedded-linux-distributions-by-yocto-project (visited on January 2021).

[OMM15] Jeffrey Osier-Mixon and Kevin McGrath. Yocto project and embedded os. 2015. URL: https://www.intel.com/content/dam/www/public/us/en/documents/education/University/yocto-project-and-embedded-os-curriculum.pdf (visited on January 2021).

[Sad15] H M Irfan Sadiq. *Using Yocto Project with BeagleBone Black : Unleash the power of the BeagleBone Black embedded platform with Yocto Project*. Packt Publishing, Birmingham, United Kingdom, 2015. ISBN 178528195X. URL: https://www.amazon.com/dp/178528973X (visited on January 2021).

[Sal13] Otavio Salvador. Yocto training : your next development platform for embedded linux. 2013. URL: https://www.slideshare.net/OtavioSalvador/yocto-training-in-english (visited on January 2021).

[SA17] Otavio Salvador and Daiane Angolini. *Embedded Linux Development using Yocto Projects : Learn to leverage the power of Yocto Project to build efficient Linux-based products*. Packt Publishing, Birmingham, United Kingdom, 2nd revised edition, 2017. ISBN 178847046X. URL: https://www.amazon.com/dp/178847046X (visited on January 2021).

[San12] Marcelo A.L. Sanz. Yocto project open source build system and collaboration initiative. 2012. URL: https://www.slideshare.net/marcelolorenzati/yocto-iua-2012 (visited on January 2021).

[Sim17] Chris Simmonds. *Mastering Embedded Linux Programming : Unleash the full potential of Embedded Linux with Linux 4.9 and Yocto Project 2.2*. Packt Publishing, Birmingham, United Kingdom, 2nd edition, 2017. ISBN 1787283283. URL: https://www.amazon.com/dp/1787283283 (visited on January 2021).

[Str15] Rudolf J. Streif. Introduction to the yocto project. 2015. URL: https://elinux.org/images/a/a8/Getting_Started_with_Embedded_Linux-_Using_the_Yocto_Project_to_Build_your_Own_Custom_Embedded_Linux_Distribution.pdf (visited on January 2021).

[Str16] Rudolf J. Streif. *Embedded Linux Systems with the Yocto Project*. Prentice Hall, Boston, United States of America, 2016. ISBN 0133443248. URL: https://www.amazon.com/dp/0133443248 (visited on January 2021).

[Swe19] Al Sweigart. *Automate the boring stuff with Python : practical programming for total beginners*. No Starch Press, San Francisco, United States of America, 2nd edition, 2019. ISBN 1593279922. URL: https://www.amazon.com/dp/1593279922 (visited on January 2021).

[Swe20] Al Sweigart. *Routineaufgaben mit Python automatisieren : Praktische Programmierlösungen für Einsteiger*. Dpunkt.Verlag GmbH, Heidelberg, Germany, 2nd edition, 2020. ISBN 3864907535. URL: https://www.amazon.com/dp/3864907535 (visited on January 2021).

[TM15] Pierre-Jean Texier and Petter Mabäcker. *Yocto for Raspberry Pi : Create unique and amazing projects by using the powerful combination of Yocto and Raspberry Pi*. Packt Publishing, Birmingham, United Kingdom, 2015. ISBN 178528195X. URL: https://www.amazon.com/dp/178528195X (visited on January 2021).

[VGS16] Alexandru Vaduva, Alex González, and Chris Simmonds. *Linux: Embedded Development Learning Path : Leverage the power of Linux to develop captivating and powerful embedded Linux projects*. Packt Publishing, Birmingham, United Kingdom, 2016. ISBN 1787124452. URL: https://www.amazon.com/dp/B01LYNTT8V (visited on January 2021).

Index

A

Abbreviations
- Presentation Learning Yocto, 108

Acronyms
- Presentation Learning Yocto, 108

Appendix
- Presentation Learning Yocto, 89

B

Bibliography
- Presentation Learning Yocto, 121

BibTeX, **103**

BPMN, **106**

C

C, **101**

C++, **101**

CORBA, **103**

Credits
- Presentation Learning Yocto, 97

CSS, **103**

D

Docutils, **99**

Downloads
- Presentation Learning Yocto, 117

E

ECMAScript, **101**

Enchant, **104**

Equations
- Presentation Learning Yocto, 116

ES, **101**

F

Figures
- Presentation Learning Yocto, 113

G

Glossary
- Presentation Learning Yocto, 99

H

HTML, **104**

I

Issues
- Presentation Learning Yocto, 119

J

JavaScript, **102**

JS, **102**

L

LaTeX, **99**

Learning Yocto
- Abbreviations Presentation, 108
- Acronyms Presentation, 108
- Appendix Presentation, 89
- Bibliography Presentation, 121
- Credits Presentation, 97
- Downloads Presentation, 117
- Equations Presentation, 116
- Figures Presentation, 113
- Glossary Presentation, 99
- Issues Presentation, 119
- License Presentation, 89
- List of Downloads Presentation, 117
- List of Equations Presentation, 116
- List of Figures Presentation, 113
- List of Issues Presentation, 119
- List of Listings Presentation, 109
- List of Tables Presentation, 111
- Listings Presentation, 109
- Tables Presentation, 111

License
- Presentation Learning Yocto, 89

List of Downloads
- Presentation Learning Yocto, 117

List of Equations
- Presentation Learning Yocto, 116

List of Figures
- Presentation Learning Yocto, 113

List of Issues
- Presentation Learning Yocto, 119

List of Listings
    Presentation Learning Yocto, 109
List of Tables
    Presentation Learning Yocto, 111
Listings
    Presentation Learning Yocto, 109

M

MOF, **103**

O

OCL, **104**

P

PDF, **104**
Presentation
    Learning Yocto, Abbreviations, 108
    Learning Yocto, Acronyms, 108
    Learning Yocto, Appendix, 89
    Learning Yocto, Bibliography, 121
    Learning Yocto, Credits, 97
    Learning Yocto, Downloads, 117
    Learning Yocto, Equations, 116
    Learning Yocto, Figures, 113
    Learning Yocto, Glossary, 99
    Learning Yocto, Issues, 119
    Learning Yocto, License, 89
    Learning Yocto, List of Downloads, 117
    Learning Yocto, List of Equations, 116
    Learning Yocto, List of Figures, 113
    Learning Yocto, List of Issues, 119
    Learning Yocto, List of Listings, 109
    Learning Yocto, List of Tables, 111
    Learning Yocto, Listings, 109
    Learning Yocto, Tables, 111
Pybtex, **99**
PyEnchant, **99**
Pygments, **99**
Python, **102**
Python Enhancement Proposals
    PEP 8, 102

Q

QVT, **105**

R

ReportLab, **99**
reStructuredText, **100**
RFC
    RFC 1866, 104

S

Sphinx, **100**
SVG, **105**
SysML, **106**

T

Tables
    Presentation Learning Yocto, 111
TeX, **107**

U

UML, **106**

X

XML, **107**