Yocto Project Training

Part 1 - Overview and Concepts

Fabio Berton

May 11, 2020

Outline

Official Documentation

- Yocto Project Concepts

Configuration Files

Files that hold global definitions of variables, user-defined variables, and hardware configuration information.

Layer

A collection of related recipes. Layers allow you to consolidate related metadata to customize your build.

Metadata

A key element of the Yocto Project is the Metadata that is used to construct a Linux distribution and is contained in the files that the OpenEmbedded build system parses when building an image.

OpenEmbedded Build System

The terms "BitBake" and "build system" are sometimes used for the OpenEmbedded Build System.

BitBake

Is a generic task execution engine that allows shell and Python tasks to be run efficiently and in parallel while working within complex inter-task dependency constraints

OpenEmbedded-Core (OE-Core)

OE-Core is metadata comprised of foundation recipes, classes, and associated files that are meant to be common among many different OpenEmbedded-derived systems, including the Yocto Project.

Packages

In the context of the Yocto Project, this term 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" something by running it through BitBake.

Recipe

The most common form of metadata.

Bitbake

A generic task executor

- User manual
- Interprets metadata, decides what tasks are required to run, and executes those tasks.
- Controls how software is built.
- GNU Make achieves its control through "makefiles", while BitBake uses "recipes".
- Includes a fetcher library for obtaining source code from various places.

Bitbake: Usage and syntax

Build a recipe

```
bitbake recipe
```

See help

```
bitbake -h
```

Most used options:

- `-c CMD` Specify the task to execute
- `-k` Continue as much as possible after an error
- `-f` Force the specified targets/task

Bitbake

See all recipe related tasks

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

Recipes are stored in `.bb` files

- Information about the package (author, homepage, license. . .)
- Version of the recipe
- Existing dependencies (both build and runtime dependencies)
- Where the source code resides and how to fetch it
- Whether the source code requires any patches, where to find them, and how to apply them
- How to configure and compile the source code
- How to assemble the generated artifacts into one or more installable packages
- Where on the target machine to install the package or packages created

Recipe append are stored in `.bbappend` files

- Extend or override information in an existing recipe file

Recipe Style Guide:

- See the online Style Guide

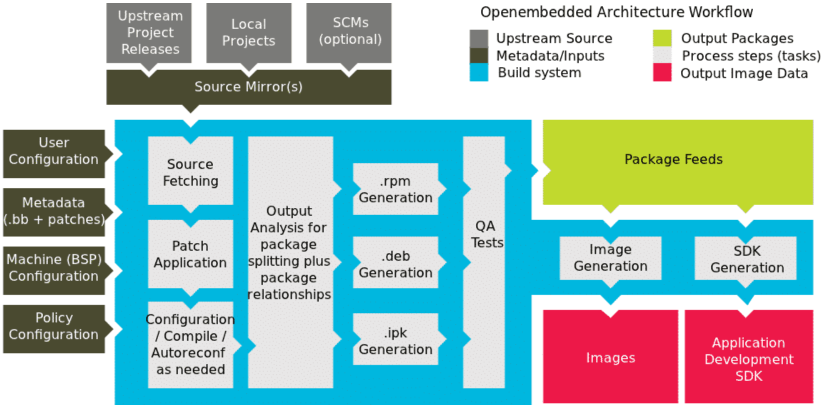
Configuration (.conf) and underlying include (.inc) files - `conf` directory

- Define various configuration variables that govern the project's build process.
- Machine configuration
- Distribution configuration
- Compiler tuning
- User configuration

And classes (.bbclass) files - `classes` directory

- Contain information that is useful to share between metadata files
- A class usually contains definitions for standard basic tasks such as:
- Fetch, unpack, configure, compile, install, package

The OpenEmbedded Build System Workflow



Yocto Explorer (ye)

Documentation

The online documentation can be found here

Yocto Explorer (ye)

Most used commands

- See documentation: `ye d var`
- View package content: `ye pv recipe`
- Expand variable: `ye x recipe var`
- Get workdir location: `ye wd recipe`
- See logs: `ye l recipe`

Overrides: Important variables

OVERRIDES

- Documentation

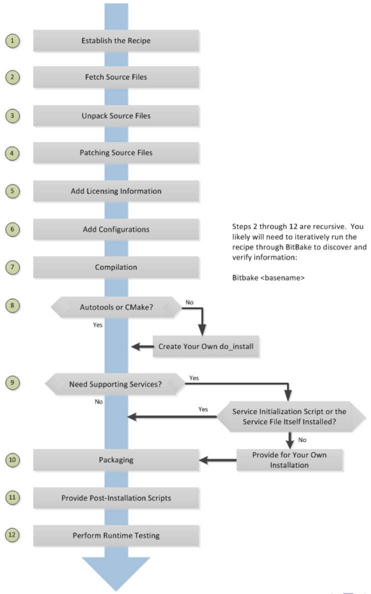
FILESOVERRIDES

- Documentation

FILESPATH

- Documentation

Recipe Overview: Online Image



1 Establish the Recipe
2 Fetch Source Files
3 Unpack Source Files
4 Patching Source Files
5 Add Licensing Information
6 Add Configurations
7 Compilation
8 Autotools or CMake?
No
Yes
Create Your Own do_install
9 Need Supporting Services?
Yes
No
Yes
Service Initialization Script or the Service File Itself Installed?
No
Provide for Your Own Installation
10 Packaging
11 Provide Post-Installation Scripts
12 Perform Runtime Testing

Steps 2 through 12 are recursive. You likely will need to iteratively run the recipe through BitBake to discover and verify information:

Bitbake <basename>

Name convention

All recipe need to follow this pattern

```
<packagename>_<version>.bb
${PN}_${PV}.bb
```

How to create a recipe from start?

The basic parts of a recipe

- Summary and Description
- Package information - Homepage, sections
- License - License checksum
- Package dependency
- Where to fetch source code
- Where unpacked recipe source code resides
- Classes that I need to inherit
- Tasks
- Runtime dependencies
- etc . . .

Simple recipe template

```
DESCRIPTION = ""
LICENSE = ""
LIC_FILES_CHKSUM = ""
DEPENDS = ""

SRC_URI = ""

do_configure() {
    :
}
do_compile() {
    :
}
do_install() {
    :
}

RDEPENDS_${PN} += ""
```

Summary and Description

Information about recipe

```
SUMMARY = "Short summary of package"
DESCRIPTION = "A piece of more detailed information about the package"
```

License: Documentation

Recipe license information

```
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0b
↪   cf8506ecda2f7b4f302"
```

Package dependency: Documentation

Usage

Lists a recipe's build-time dependencies

Examples:

```
DEPENDS = "foo"
DEPENDS = "foo-native bar-native"
DEPENDS += "foo bar"
DEPENDS_append = "bar"
```

TIP

Use ye to get final variable value

Fetching: Documentation

Fetchers

- Local file fetcher
- HTTP/FTP wget fetcher
- Git Fetcher

Examples:

```
SRC_URI = "file://relativefile.patch"
SRC_URI = "http://oe.handhelds.org/not_there.aac"
SRC_URI = "git://git.oe.handhelds.org/git/vip.git;protocol=http"
```

TIP

Use ye to get final variable value

Unpacking: Documentation

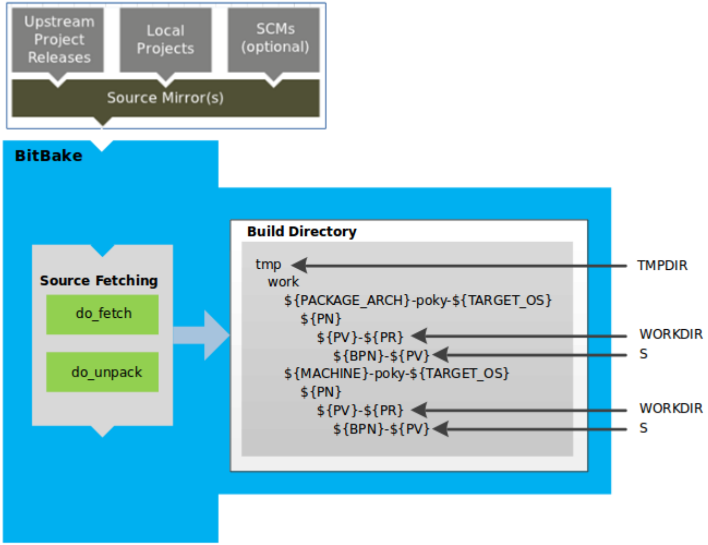
Examples

```
S = "{WORKDIR}/${PN}-${PV}"
S = "${WORKDIR}/git"
```

TIP

Use ye to get final variable value

Unpacking (cont.):

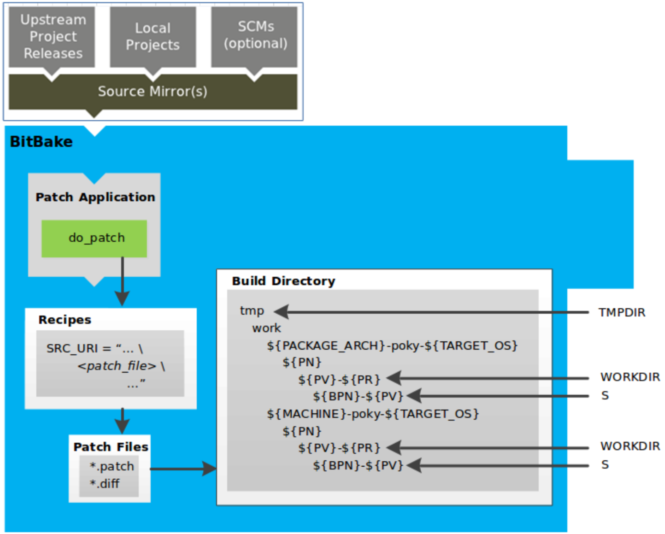


Patching: Documentation

Examples:

```
SRC_URI += "file://foo.patch"
SRC_URI_append = " file://foo.patch"
```

Patching (cont.):



Classes

bbclass files

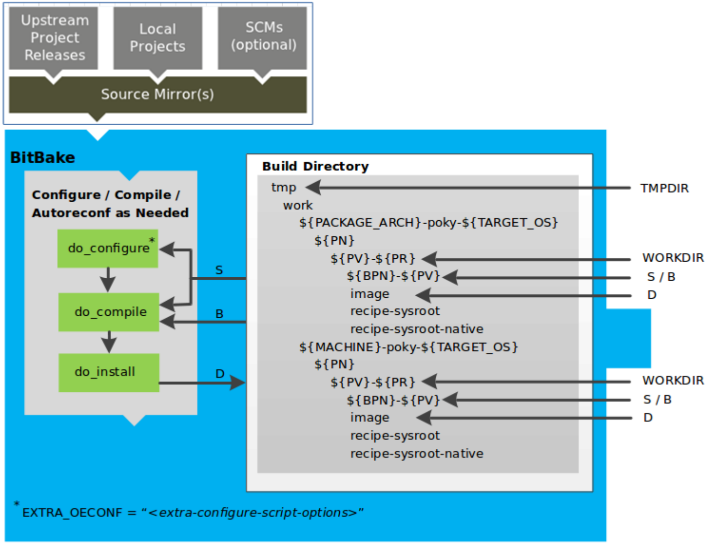
- Why do I need to inherit classes?
- How do I know what classes I need?

Configure, compile, and install

Documentation

- configure documentation
- compile documentation
- install documentation

Workflow: Configure, compile, and install. . .



Runtime dependencies

RDEPENDS

- Lists runtime dependencies of a package
- These dependencies are other packages that must be installed in order for the package to function correctly

TIP:

Use `ye` to get final variable value

Glossary

Documetation

Online Glossary

Using ye

```
ye d variable
```

Build history

Maintaining Build Output Quality

- Documentation
- Is a class that exists to help you maintain the quality of your build output
- It records information about the contents of each package and image
- Highlight unexpected and possibly unwanted changes in the build output

Enabling and Build History

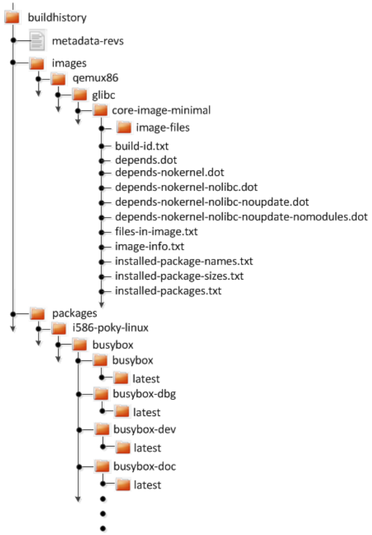
```
INHERIT += "buildhistory"
BUILDHISTORY_COMMIT = "1"
```

Build History (cont.)

Image Information

- image-files: A directory containing selected files from the root filesystem
- files-in-image.txt: A list of files in the image
- installed-package-names.txt: A list of installed packages by name only
- installed-package-sizes.txt: A list of installed packages ordered by size
- installed-packages.txt: A list of installed packages with full package filenames

Understanding What the Build History Contain



References

All this presentation was based on

- Yocto Project Mega Manual