Image

What is an image 2

What is an image

- Image is a top level recipe. (It inherits an image.bbclass)
- Building an image creates an entire Linux distribution from source
- Compiler, tools, libraries
- BSP: Bootloader, Kernel
- Root filesystem:
    - Base OS
    - services
    - Applications
    - etc

Creating custom images

- You often need to create your own Image recipe in order to add new packages or functionality

Two ways

- creating an image from scratch
- extend an existing recipe (preferable)

Package group

- A package group is a set of packages that can be included on any image.
- A package group can contain a set of packages.
- Using the packagegroup name in IMAGE_INSTALL variable install all the packages defined by the package group into the root file system of your target image.
- There are many package groups. There are present in subdirectories named "packagegroups"
    - $ find . -name 'packagegroups'
- They are recipe files(.bb) and starts with packagegroup-
- For example,
    - packagegroup-core-boot: Provides the minimum set of packages necessary to create a bootable image with console.

Creating an image from scratch

- The simplest way is to inherit the core-image bbclass, as it provides a set of image features that can be used very easily
- inherit core-image
- Which tells us that the definition of what actually gets installed is defined in the core-image.bbclass.
- Image recipes set IMAGE_INSTALL to specify the packages to install into an image through image.bbclass.
- Create an images directory
    - $ mkdir -p recipes-examples/images
- Create the image recipe
    - $ vi recipes-examples/images/lwl-image.bb
    - SUMMARY = "A small boot image for LWL learners"
    - LICENSE = "MIT"
- inherit core-image
    - # Core files for basic console boot
    - IMAGE_INSTALL = "packagegroup-core-boot"
    - IMAGE_ROOTFS_SIZE ?= "8192"
    - #Add our needed applications
    - IMAGE_INSTALL += "usbutils"

Reusing an existing image

- When an image mostly fits our needs and we need to do minor adjustments on it,it is very convenient to reuse its code
- This makes code maintenance easier and highlights the functional differences
- For example, if we want to include an application (lsusb)
- Create another recipe:
    - $ vim recipes-examples/images/lwl-image-reuse.bb
    - require recipes-core/images/core-image-minimal.bb
    - IMAGE_INSTALL_append = " usbutils"

Customizing Images Using Custom IMAGE_FEATURES and EXTRA_IMAGE_FEATURES

- Another method for customizing your image is to enable or disable high-level image features by using the IMAGE_FEATURES and EXTRA_IMAGE_FEATURES variables
- IMAGE_FEATURES/EXTRA_IMAGE_FEATURES is made to enable special features for your image, such as empty password for root, debug image, special packages, x11, splash, ssh-server

What's the difference between IMAGE_FEATURES and EXTRA_IMAGE_FEATURES

- Best practice is to
  - Use IMAGE_FEATURES from a recipe
  - Use EXTRA_IMAGE_FEATURES from local.conf

How it works?

- To understand how these features work, the best reference is meta/classes/core-image.bbclass
- This class lists out the available IMAGE_FEATURES of which most map to package groups while some, such as debug-tweaks and read-only-rootfs, resolve as general configuration settings
- In summary, the file looks at the contents of the IMAGE_FEATURES variable and then maps or configures the feature accordingly
- Based on this information, the build system automatically adds the appropriate packages or configurations to the IMAGE_INSTALL variable.

Example of IMAGE_FEATURES

- To illustrate how you can use these variables to modify your image, consider an example that selects the SSH server.
- The Yocto Project ships with two SSH servers you can use with your images: Dropbear and OpenSSH.
- OpenSSH is a well-known standard SSH server implementation
- Dropbear is a minimal SSH server appropriate for resource-constrained environments

- By default, the core-image-sato image is configured to use Dropbear. The core-image-full-cmdline and core-image-lsb images both include OpenSSH.
- The core-image-minimal image does not contain an SSH server.

debug-tweaks

- In the default state, local.conf file has EXTRA_IMAGE_FEATURES set to "debug-tweaks"
- debug-tweaks features enable password-less login for the root user
- Advantage: makes logging in for debugging or inspection easy during development
- Disadvantage: anyone can easily log in during production.
- So, you need to remove the 'debug-tweaks' feature from production image

Read-Only Root Filesystem

Why do we need read-only rootfs

- Reduce wear on flash memory
- Eliminate system file corruption

How to do it?

- To create the read-only root filesystem, simply add the "read-only-rootfs" feature to your image.
- IMAGE_FEATURES = "read-only-rootfs" in your recipe
  - or
- EXTRA_IMAGE_FEATURES += "read-only-rootfs" in local.conf

Boot Splash screen

- IMAGE_FEATURES += "splash"
  - or
- EXTRA_IMAGE_FEATURES += "splash"

Some other Features

- tools-debug: Installs debugging tools such as strace and gdb.
- tools-sdk: Installs a full SDK that runs on the device.

IMAGE_LINGUAS

- Specifies the list of locales to install into the image during the root filesystem construction process
- IMAGE_LINGUAS = "zh-cn"
- Inside qemu image
    - $ locale -a

IMAGE_FSTYPES

- The IMAGE_FSTYPES variable determines the root filesystem image type
- If more than one format is specified, one image per format will be generated
- Image formats instructions are delivered in Poky: meta/classes/image_types.bbclass
    - $ bitbake -e <image_name> | grep ^IMAGE_FSTYPES=

Types supported

Btrfs *container* *cpio* *cpio.gz* *cpio.lz4* *cpio.lzma* *cpio.xz* *cramfs* *elf* *ext2* *ext2.bz2* *ext2.gz* *ext2.lzma* *ext3* *ext3.gz* *ext4* *ext4.gz* *f2fs* *ddimg* *iso* *jffs2* *jffs2.sum* *multiubi* *squashfs* *squashfs-lz4* *squashfs-zo* *squashfs-xz* *tar* *tar.bz2* *tar.gz* *tar.lz4* *tar.xz* *ubi* *ubifs* *wic* *ic.bz2* *wic.gz* *wic.lzma*

Creating your own image type

- If you have a particular layout on your storage (for example bootloader location on an SD card), you may want to create your own image type
- This is done through a class that inherits from image_types
- It has to define a function named IMAGE_CMD_<type>
- Example: sdcard_image-rpi.bbclass in meta-raspberrypi

IMAGE_NAME

- The name of the output image files minus the extension
- This variable is derived using the IMAGE_BASENAME, MACHINE, and DATETIME variables
- IMAGE_NAME = "${IMAGE_BASENAME}-${MACHINE}-${DATETIME}"

IMAGE_MANIFEST

- The manifest file for the image
- This file lists all the installed packages that make up the image.
- The file contains package information on a line-per-package basis as follows:
    - packagename packagearch version
- The image class defines the manifest file as follows:
    - IMAGE_MANIFEST = "${DEPLOY_DIR_IMAGE}/${IMAGE_NAME}.rootfs.manifest"

Challenge

Try other IMAGE_FEATURES