NXP Semiconductors
User's Guide

Document Number: IMXLXYOCTOUG
Rev. L4.9.88_2.1.0_8mm-alpha, 06/2018

i.MX Yocto Project User's Guide

Contents

1 Overview

This document describes how to build an image for an i.MX board by using a Yocto Project build environment. It describes the i.MX release layer and i.MX-specific usage.

The Yocto Project is an open-source collaboration focused on embedded Linux® OS development. For more information on Yocto Project, see the Yocto Project page: www.yoctoproject.org/. There are several documents on the Yocto Project home page that describe in detail how to use the system. To use the basic Yocto Project without the i.MX release layer, follow the instructions in the *Yocto Project Quick Start* found at www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html.

The FSL Yocto Project Community BSP (found at freescale.github.io) is a development community outside NXP providing support for i.MX boards in the Yocto Project environment. i.MX joined the Yocto Project community providing a release based on the Yocto Project framework. Information specific to FSL community BSP use is available on the community web page. This document is an extension of the community BSP documentation.

Files used to build an image are stored in layers. Layers contain different types of customizations and come from different sources. Some of the files in a layer are called recipes. Yocto Project recipes contain the mechanism to retrieve source code, build and package a component. The following lists show the layers used in this release.

i.MX release layer

- meta-fsl-bsp-release
  - meta-bsp - updates for meta-freescale, poky, and meta-openembedded layers
  - meta-sdk - updates for meta-freescale-distros

Yocto Project community layers

- meta-freescale: provides support for the base and for i.MX Arm® reference boards.
- meta-freescale-3rdparty: provides support for 3rd party and partner boards.
- meta-freescale-distro: additional items to aid in development and exercise board capabilities.
- fsl-community-bsp-base: often renamed to base. Provides base configuration for FSL Community BSP.
- meta-openembedded: Collection of layers for the OE-core universe. See layers.openembedded.org/.
- poky: basic Yocto Project items in Poky. See the Poky README for details.
- meta-browser: provides several browsers.
- meta-qt5: provides Qt5.

References to community layers in this document are for all the layers in Yocto Project except meta-fsl-bsp-release. i.MX boards are configured in the meta-fsl-bsp-release and meta-freescale layers. This includes U-Boot, the Linux kernel, and reference board-specific details.

i.MX provides an additional layer called the i.MX BSP Release, named meta-fsl-bsp-release, to integrate a new i.MX release with the FSL Yocto Project Community BSP. The meta-fsl-bsp-release layer aims to release the updated and new Yocto Project recipes and machine configurations for new releases that are not yet available on the existing meta-freescale and meta-freescale-distro layers in the Yocto Project. The contents of the i.MX BSP Release layer are recipes and machine configurations. In many test cases, other layers implement recipes or include files and the i.MX release layer provides updates to the recipes by either appending to a current recipe, or including a component and updating with patches or source locations. Most i.MX release layer recipes are very small because they use what the community has provided and update what is needed for each new package version that is unavailable in the other layers.

The i.MX BSP Release layer also provides image recipes that include all the components needed for a system image to boot, making it easier for the user. Components can be built individually or through an image recipe, which pulls in all the components required in an image into one build process.

The i.MX kernel and U-Boot releases are accessed through i.MX public git servers. However, several components are released as packages on the i.MX mirror. The package-based recipes pull files from the i.MX mirror instead of a git location and generate the package needed.

All packages which are released as binary are built with hardware floating point enabled as specified by the DEFAULTTUNE defined in each machine configuration file. Software floating point packages are not provided starting with the jethro releases.

Release L4.9.88_2.1.0_8mm_alpha is released for Yocto Project 2.4 (Rocko). The same recipes for Yocto Project 2.4 are going to be upstreamed and made available on the next release of the Yocto Project release. The Yocto Project release cycle lasts roughly six months.

The recipes and patches in meta-fsl-bsp-release are upstreamed to the community layers. After that is done for a particular component, the files in meta-fsl-bsp-release are no longer needed and the FSL Yocto Project Community BSP will provide support. The community supports i.MX reference boards, community boards, and third-party boards. A complete list can be found at freescale.github.io/doc/release-notes/2.2/index.html#document-bsp-scope. All board references in this document are related to the i.MX machine configuration files only.

1.1 End user licence agreement

During the setup environment process of the Freescale Yocto Project Community BSP, the NXP End User License Agreement (EULA) is displayed. To continue to use the i.MX Proprietary software, users must agree to the conditions of this license. The agreement to the terms allows the Yocto Project build to untar packages from the i.MX mirror.

NOTE

Read this license agreement carefully during the setup process, because once accepted, all further work in the i.MX Yocto Project environment is tied to this accepted agreement.

1.2 References

i.MX has multiple families supported in software. The folllowing are the listed families and SoCs per family. The i.MX Linux® Release Notes will describe which SoC is supported in current release. Some previously released SoC might be buildable in current release but not validated if they are at the previous validated level.

- i.MX 6 Family: 6QuadPlus, 6Quad, 6DualLite, 6SoloX, 6SoloLite, 6SLL 6UltraLite, 6ULL
- i.MX 7 Family: 7Dual, 7ULP
- i.MX 8 Family: 8QuadMax
- i.MX 8M Family: 8MQuad, 8MMini
- i.MX 8X Family: 8QuadXPlus

This release includes the following references and additional information.

- *i.MX Linux® Release Notes* (IMXLXRN) - Provides the release information.
- *i.MX Linux® User's Guide* (IMXLUG) - Contains the information on installing U-Boot and Linux OS and using i.MX-specific features.
- *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG) - Contains the instructions for setting up and building Linux OS in the Yocto Project.
- *i.MX Reference Manual* (IMXLXRM) - Contains the information on Linux drivers for i.MX.
- *i.MX Graphics User's Guide* (IMXGRAPHICUG) - Describes the graphics features.
- *i.MX BSP Porting Guide* (IMXXBSPPG) - Contains the instructions on porting the BSP to a new board.

2 Features

i.MX Yocto Project Release layers have the following features:

- Linux kernel recipe
  - The kernel recipe resides in the recipes-kernel folder and integrates a i.MX kernel from the source downloaded from the i.MX git server. This is done automatically by the recipes in the project.
  - L4.9.88_2.1.0_8mm_alpha is a Linux kernel released for the Yocto Project.
- U-Boot recipe
  - The U-Boot recipe resides in the recipes-bsp folder and integrates a i.MX uboot-imx.git from the source downloaded from the i.MX git server.
  - Certain i.MX boards use different U-Boot versions.
  - i.MX release L4.9.88_2.1.0_8mm_alpha for the i.MX 8 devices uses an updated v2017.03 i.MX U-Boot version. This version has not been updated for all i.MX hardware.
  - The i.MX Yocto Project Community BSP uses u-boot-fslc from the mainline, but this is only supported by the U-Boot community and is not supported with the L4.9.88 kernel.
  - The i.MX Yocto Project Community BSP updates U-Boot versions frequently, so the information above might change as new U-Boot versions are integrated to meta-freescale layers and updates from i.MX u-boot-imx releases are integrated into the mainline.
- Graphics recipes
  - Graphics recipes reside in recipes-graphics folder.

    - Graphics recipes integrate the i.MX graphics package release. For the i.MX boards that have a GPU, the imx-gpu-viv recipes package the graphic components for each DISTRO – X11, frame buffer (FB), XWayland, Wayland backend, and Weston compositor (Weston). Only i.MX 6 and i.MX 7 support X11 and Frame Buffer.
    - Xorg-driver integrates the xserver-xorg.
- i.MX package recipes

  imx-lib, imx-test, and firmware-imx reside in recipes-bsp and pull from the i.MX mirror to build and package into image recipes.

- Multimedia recipes
    - Multimedia recipes reside in recipes-multimedia.
    - Recipes include imx-codec, imx-parser, libvpuwrap, and imx-gstreamer-plugins that pull from the i.MX mirror to build and package into image recipes.
    - Some recipes are provided for codecs that are restricted. Packages for these are not on the i.MX mirror. These packages are available separately. Contact your i.MX Marketing representative to acquire these.
- Core recipes

  Some recipes for rules, such as udev, provide updated i.MX rules to be deployed in the system. These recipes are usually updates of policy and are used for customization only. Releases only provide updates if needed.

- Demo recipes

  Demonstration recipes reside in the meta-sdk directory. This layer contains image recipes and recipes for customization, such as touch calibration, or recipes for demonstration applications.

3 Host Setup

To get the Yocto Project expected behavior in a Linux Host Machine, the packages and utilities described below must be installed. An important consideration is the hard disk space required in the host machine. For example, when building on a machine running Ubuntu, the minimum hard disk space required is about 50 GB for the X11 backend. It is recommended that at least 120 GB is provided, which is enough to compile all backends together.

The recommended minimum Ubuntu version is 14.04 or later. Earlier versions may cause the Yocto Project build setup to fail, because it requires python versions only available starting with Ubuntu 12.04. See The Yocto Project reference manual for more information.

Ubuntu 16.04 users have commented on errors during build for SDL. To fix this comment out in local.conf the following lines as such adding # character

```
#PACKAGECONFIG_append_pn-qemu-native = " sdl"
#PACKAGECONFIG_append_pn-nativesdk-qemu = " sdl"
```

3.1 Host packages

A Yocto Project build requires that some packages be installed for the build that are documented under the Yocto Project. Go to Yocto Project Quick Start and check for the packages that must be installed for your build machine.

Essential Yocto Project host packages are:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
  build-essential chrpath  socat  libsdl1.2-dev
```

i.MX layers host packages for a Ubuntu 12.04 or 14.04 host setup are:

```
$ sudo apt-get install libsdl1.2-dev xterm  sed cvs subversion coreutils texi2html \
docbook-utils python-pysqlite2 help2man make gcc g++ desktop-file-utils \
libgl1-mesa-dev libglu1-mesa-dev mercurial autoconf automake groff curl lzop asciidoc
```

i.MX layers host packages for a Ubuntu 12.04 host setup only are:

```
$ sudo apt-get install uboot-mkimage
```

i.MX layers host packages for a Ubuntu 14.04 host setup only are:

```
$ sudo apt-get install u-boot-tools
```

The configuration tool uses the default version of grep that is on your build machine. If there is a different version of grep in your path, it may cause builds to fail. One workaround is to rename the special version to something not containing "grep".

3.2 Setting up the repo utility

Repo is a tool built on top of Git that makes it easier to manage projects that contain multiple repositories, which do not need to be on the same server. Repo complements very well the layered nature of the Yocto Project, making it easier for users to add their own layers to the BSP.

To install the “repo” utility, perform these steps:

1. Create a bin folder in the home directory.

   ```
   $ mkdir ~/bin (this step may not be needed if the bin folder already exists)
   $ curl https://storage.googleapis.com/git-repo-downloads/repo  > ~/bin/repo
   $ chmod a+x ~/bin/repo
   ```

2. Add the following line to the `.bashrc` file to ensure that the `~/bin` folder is in your PATH variable.

   ```
   export PATH=~/bin:$PATH
   ```

4 Yocto Project Setup

First make sure that git is set up properly with the commands below.

```
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"
$ git config --list
```

The i.MX Yocto Project BSP Release directory contains a `sources` directory, which contains the recipes used to build one or more build directories, and a set of scripts used to set up the environment.

The recipes used to build the project come from both the community and i.MX. The Yocto Project layers are downloaded to the `sources` directory. This sets up the recipes that are used to build the project.

The following example shows how to download the i.MX Yocto Project Community BSP recipe layers. For this example, a directory called `imx-yocto-bsp` is created for the project. Any name can be used instead of this.

```
$ mkdir imx-yocto-bsp
$ cd imx-yocto-bsp
$ repo init -u https://source.codeaurora.org/external/imx/imx-manifest  -b imx-linux-rocko -
m imx-4.9.88-2.1.0-8mm_alpha.xml
$ repo sync
```

NOTE

https://source.codeaurora.org/external/imx/imx-manifest/tree/?h=imx-linux-rocko has a list of all manifest files supported in this release.

When this process is completed, the source code is checked out into the directory `imx-yocto-bsp/sources`.

You can perform repo synchronization, with the command `repo sync`, periodically to update to the latest code.

If errors occur during repo initialization, try deleting the .repo directory and running the repo initialization command again.

The `repo init` is configured for the latest patches in the line. Follow the instructions in index: imx-manifest.git to retrieve the original GA. Otherwise, GA plus patches are picked up by default. To pick up previous releases from rocko base add -m (release manifest name) at the end of repo init line and it will retrieve previous releases. Examples are provided in the README in the link provided above.

5 Image Build

This section provides the detailed information along with the process for building an image.

5.1 Build configurations

i.MX provideds a script, `fsl-setup-release.sh`, that simplifies the setup for i.MX machines. To use the script, the name of the specific machine to be built for needs to be specified as well as the desired graphical backend. The script sets up a directory and the configuration files for the specified machine and backend.

In the `meta-fsl-bsp-release` layer, i.MX provides new or updated machine configurations that overlay the `meta-freescale` machine configurations. These files are copied into the `meta-freescale/conf/machine` directory by the `fsl-setup-release.sh` script. The following are i.MX machine configuration files that can be selected. Check either the release notes or the machine directory for the latest additions.

- imx8mqevk
- imx8mmevk

Each build folder must be configured in such way that they only use one distro. Each time the variable DISTRO_FEATURES is changed, a clean build folder is needed. Each graphical backend Frame Buffer, Wayland, XWayland, and X11 each have a distro configuration. If no DISTRO file is specified, the xwayland distro is set up as default. Distro configurations are saved in the local.conf in the DISTRO setting and are displayed when the bitbake is running. In past releases, we used the poky distro and customized versions and providers in our layer.conf but a custom distro is a better solution. When the default poky distro is used, the default community configuration is used. As a i.MX release, we prefer to have a set of configurations that NXP supports and has been testing.

Here are the list of DISTRO configurations. Note that DirectFB is no longer supported and fsl-imx-wayland and fsl-imx-fb are not validated in our test cycle.

- fsl-imx-x11 - X11 graphics are not supported on i.MX 8.
- fsl-imx-wayland - Wayland weston graphics.
- fsl-imx-xwayland - Wayland graphics and X11. X11 applications using EGL are not supported.
- fsl-imx-fb - Frame Buffer graphics - no X11 or Wayland. Frame Buffer is not supported on i.MX 8.

Users are welcome to create their own distro file based on one of these to customize their environment without updating the local.conf to set preferred versions and providers.

The syntax for the fsl-setup-release.sh script is shown below.

```
$ DISTRO=<distro name> MACHINE=<machine name> source fsl-setup-release.sh -b <build dir>
```

DISTRO=<distro configuration name> is the distro, which configures the build environment and it is stored in meta-fsl-bsp-release/imx/meta-sdk/conf/distro.

MACHINE=<machine configuration name> is the machine name which points to the configuration file in conf/machine in meta-freescale and meta-fsl-bsp-release.

-b <build dir> specifies the name of the build directory created by the fsl-setup-release.sh script.

When the script is run, it prompts the user to accept the EULA. Once the EULA is accepted, the acceptance is stored in local.conf inside each build folder and the EULA acceptance query is no longer displayed for that build folder.

After the script runs, the working directory is the one just created by the script, specified with the -b option. A conf folder is created containing the files bblayers.conf and local.conf.

The <build dir>/conf/bblayers.conf file contains all the metalayers used in the i.MX Yocto Project release.

The local.conf file contains the machine and distro specifications:

```
MACHINE ??= 'imx7ulpevk'
DISTRO ?= 'fsl-imx-x11'
ACCEPT_FSL_EULA = "1"
```

The MACHINE configuration can be changed by editing this file, if necessary.

ACCEPT_FSL_EULA in the local.conf file indicates that you have accepted the conditions of the EULA.

In the meta-fsl-bsp-release layer, consolidated machine configurations (imx6qpdlsolox.conf and imx6ul7d.conf) are provided for i.MX 6 and i.MX 7 machines. i.MX uses these to build a common image with all the device trees in one image for testing. Do not use these machines for anything other than testing.

5.2 Choosing an i.MX Yocto project image

The Yocto Project provides some images which are available on different layers. Poky provides some images, meta-freescale and meta-freescale-distro provide others, and additional image recipes are provided in the meta-fsl-bsp-release layer. The following table lists various key images, their contents, and the layers that provide the image recipes.

Table 1. i.MX Yocto project images

Image name	Target	Provided by layer
core-image-minimal	A small image that only allows a device to boot.	Poky
core-image-base	A console-only image that fully supports the target device hardware.	Poky
core-image-sato	An image with Sato, a mobile environment and visual style for mobile devices. The image supports X11 with a Sato theme and uses Pimlico applications. It contains a terminal, an editor and a file manager.	Poky
fsl-image-machine-test	An FSL Community i.MX core image with console environment - no GUI interface.	meta-freescale-distro
fsl-image-validation-imx	Builds an i.MX image with a GUI without any Qt content.	meta-fsl-bsp-release/imx/meta-sdk
fsl-image-qt5-validation-imx	Builds an opensource Qt 5 image. These images are only supported for i.MX SoC with hardware graphics. They are not supported on the i.MX 6UltraLite, i.MX 6UltraLiteLite, and i.MX 7Dual.	meta-fsl-bsp-release/imx/meta-sdk

5.3 Building an image

The Yocto Project build uses the bitbake command. For example, `bitbake <component>` builds the named component. Each component build has multiple tasks, such as fetching, configuration, compilation, packaging, and deploying to the target rootfs. The bitbake image build gathers all the components required by the image and build in order of the dependency per task. The first build is the toolchain along with the tools required for the components to build.

The following command is an example on how to build an image:

```
$ bitbake fsl-image-validation-imx
```

5.4 Bitbake options

The bitbake command used to build an image is `bitbake <image name>`. Additional parameters can be used for specific activities described below. Bitbake provides various useful options for developing a single component. To run with a bitbake parameter, the command looks like this:

```
bitbake  <parameter> <component>
```

`<component>` is a desired build package.

The following table provides some bitbake options.

Table 2. Bitbake options

Bitbake paramater	Description
-c fetch	Fetches if the downloads state is not marked as done.
-c cleanall	Cleans the entire component build directory. All the changes in the build directory are lost. The rootfs and state of the component are also cleared. The component is also removed from the download directory.
-c deploy	Deploys an image or component to the rootfs.
-k	Continues building components even if a build break occurs.
-c compile -f	It is not recommended that the source code under the tmp directory is changed directly, but if it is, the Yocto Project might not rebuild it unless this option is used. Use this option to force a recompile after the image is deployed.
-g	Lists a dependency tree for an image or component.
-DDD	Turns on debug 3 levels deep. Each D adds another level of debug.

5.5 U-Boot configuration

U-Boot configurations are defined in the main machine configuration file. The configuration is specified by using the UBOOT_CONFIG settings. This requires setting UBOOT_CONFIG in `local.conf`. Otherwise, the U-Boot build uses SD boot by default.

These can be built separately by using the following commands (change MACHINE to the correct target).

U-Boot type	Build setup	Build command
U-Boot EIM-NOR	`$ echo "UBOOT_CONFIG = \"eimnor\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot SPI-NOR	`$ echo "UBOOT_CONFIG = \"spinor\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot NAND	`$ echo "UBOOT_CONFIG = \"nand\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot SATA	`$ echo "UBOOT_CONFIG = \"sata\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot Arm® Cortex®-M4 core	`$ echo "UBOOT_CONFIG = \"m4fastup\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot QSPI1	`$ echo "UBOOT_CONFIG = \"qspi1\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot QSPI2	`$ echo "UBOOT_CONFIG = \"qspi2\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot EMMC	`$ echo "UBOOT_CONFIG = \"emmc\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot m4fastup	`$ echo "UBOOT_CONFIG = \"m4fastup\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot epdc	`$ echo "UBOOT_CONFIG = \"epdc\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`
U-Boot FSPI	`$ echo "UBOOT_CONFIG = \"fspi\"" >> conf/local.conf`	`$ MACHINE=<machine name> bitbake -c deploy u-boot-imx`

5.6 Build scenarios

The following are build setup scenarios for various configurations.

Set up the manifest and populate the Yocto Project layer sources with these commands:

```
$ mkdir imx-yocto-bsp
$ cd imx-yocto-bsp
$ repo init -u https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-rocko -m
imx-4.9.88-2.1.0_8mm_alpha.xml
$ repo sync
```

The following sections give some specific examples. Replace the machine names and the backends specified to customize the commands.

5.6.1 X-11 image on i.MX 6Quad SABRE-SD

```
$ DISTRO=fsl-imx-x11 MACHINE=imx6qsabresd source fsl-setup-release.sh -b build-x11
$ bitbake fsl-image-validation-imx
```

This builds an X11 image without Qt 5. To build with Qt 5, use fsl-image-qt5-validation-imx instead.

5.6.2 Frame Buffer image on i.MX 6QuadPlus SABRE-AI

```
$ DISTRO=fsl-imx-fb MACHINE=imx6qpsabreauto source fsl-setup-release.sh –b build-fb
$ bitbake fsl-image-qt5-validation-imx
```

This builds Qt 5 on a frame buffer backend. To build without Qt 5, use image recipe fsl-image-validation-imx.

5.6.3 XWayland image on i.MX 8MQuad EVK

```
$ DISTRO=fsl-imx-xwayland MACHINE=imx8mqevk source fsl-setup-release.sh -b build-xwayland
$ bitbake fsl-image-qt5-validation-imx
```

This builds an XWayland image with Qt 5. To build without Qt 5, use fsl-image-validation-imx instead.

5.6.4 Wayland image on i.MX8 MQuad EVK

```
$ DISTRO=fsl-imx-wayland MACHINE=imx8mqevk source fsl-setup-release.sh -b build-wayland
$ bitbake fsl-image-qt5-validation-imx
```

This builds a Qt 5 Weston Wayland image. To build without Qt 5, build fsl-image-validation-imx.

5.6.5 Restarting a build environment

If a new terminal window is opened or the machine is rebooted after a build directory is set up, the setup environment script should be used to set up the environment variables and run a build again. The full `fsl-setup-release.sh` is not needed.

```
$ source setup-environment <build-dir>
```

5.6.6 Chromium Browser on X11, XWayland, and Wayland

The Yocto Project community has chromium-imx recipes to enable hardware acceleration for X11, XWayland, and Wayland version Chromium Browser for i.MX SoC with GPU hardware. Note that NXP does not support or test the patches from the community, neither for VPU or GPU support. This section describes how to integrate Chromium into your rootfs and enable hardware accelerated rendering of WebGL. The Chromium browser requires additional layers added in the `fsl-release-setup.sh` script automatically.

In `local.conf`, you can perform the following operations:

- Add Chromium into your image.

  ```
  CORE_IMAGE_EXTRA_INSTALL += "chromium libexif"
  ```

- Add the commercial white list into `local.conf`.

  ```
  LICENSE_FLAGS_WHITELIST="commercial"
  ```

- Add chromium blacklist disable into into `local.conf`.

  ```
  PNBLACKLIST[chromium] = ""
  ```

This allows proprietary code to be included into your image. This enables several recipes flagged as "commercial" to be built and included in the final image. It is a huge set of commercial licensed packages, with different licenses for each one. Therefore, this allows the build and the installation. Additional license obligations will need to be met for these additions. Ensure that you know what these are and that you are in compliance with them.

5.6.7 Qt 5 and QtWebEngine browsers

Qt 5 has both a commercial and an open source license. When building in Yocto Project the open source license is the default. Make sure to understand the differences between these licenses and choose appropriately. After custom Qt 5 development has started on the open source license it cannot be used with the commercial license. Work with a legal representative to understand the differences between these licenses.

There are three Qt 5 browsers available. QtWebEngine browsers can be found in `/usr/share/qt5/examples/webenginewidgets/browser`, `/usr/share/qt5/examples/webenginewidgets/fancybrowser` and `/usr/share/qt5/examples/webengine/quicknanobrowser`.

To run any of these, after booting up Linux OS on your device, tell Qt 5 which graphics to use by setting the environment variable below. See Section "Qt 5" in the *i.MX Linux® User's Guide* (IMXLUG) for the information on the graphics for different graphical backends.

```
$export QT_QPA_PLATFORM=$Graphics
```

All three browsers can be run by going to the directory above and running the executable found there. Touchscreen can be enabled by adding the parameters `-plugin evdevtouch:/dev/input/event0` to the executable. The DISPLAY variable may need to be set in the environment before beginning:

```
export DISPLAY=:0.0
```

The command line might look like one of these:

```
./browser -plugin evdevtouch:/dev/input/event0
./fancybrowser -plugin evdevtouch:/dev/input/event0
./quicknanobrowser -plugin evdevtouch:/dev/input/event0
```

5.6.8 Systemd

Systemd is enabled as default initialization manager. To disable systemd as default, go to the fsl-imx-preferred-env.inc and comment out the systemd section.

5.6.9 Multilib Enablement

For i.MX 8, building 32 bit applications on 64 bit OS can be supported using the multilib configuration. Multilib offers the ability to build libraries with different target optimizations or architecture formats and combine these together into one system image. Multilib is enabled by adding the MULTILIB, DEFAULTTUNE and IMAGE_INSTALL declaration to your local.conf file.

The MULTILIBS declaration is typically lib32 or lib64 and need to be defined in MULTILIB_GLOBAL_VARIANTS variable as shown here.

```
MULTILIBS = "multilib:lib32"
```

DEFAULTTUNE must be one of the AVAILTUNES values for this alternative library type as shown here.

```
DEFAULTTUNE_virtclass-multilib-lib32 = "armv7athf-neon "
```

IMAGE_INSTALL will add to the image the 32bit libraries required by the specific application as shown here:

```
IMAGE_INSTALL_append += " lib32-bash"
```

For the case on i.MX 8 building a 32 bit application support would require the following statements in local.conf. This configuration specifies a 64-bit machine as the main machine type and adds multilib:lib32 where those libraries are compiled with the “armv7athf-neon” tune, then includes to all images the lib32- packages.

```
MACHINE = imx8mqevk
# Define multilib target
require conf/multilib.conf
MULTILIBS = "multilib:lib32"
DEFAULTTUNE_virtclass-multilib-lib32 = "armv7athf-neon"
# Add the multilib packages to the image
IMAGE_INSTALL_append = "lib32-glibc lib32-libgcc lib32-libstdc++"
```

5.6.10 Optee Enablement

Optee requires 3 components: optee_os, optee-client and optee-test. In addition, the kernel and U-Boot have configurations. The optee_os resides in the bootloader while the optee client and test reside in the rootfs.

To build optee follow the directions on the README in the meta-fsl-bsp-release which describe how to add optee to the DISTRO_FEATURES in the local.conf. This release optee is not enabled as default. Future releases will have optee enabled as default.

6 Image Deployment

After a build is complete, the created image resides in `<build directory>/tmp/deploy/images`. An image is, for the most part, specific to the machine set in the environment setup. Each image build creates a U-Boot, a kernel, and an image type based on the IMAGE_FSTYPES defined in the machine configuration file. Most machine configurations provide an SD card image (.sdcard), an ext3 and tar.bz2. The ext3 is the root file system only. The .sdcard image contains U-Boot, the kernel and the rootfs completely set up for use on an SD card.

6.1 Flashing an SD card image

An SD card image provides the full system to boot with U-Boot and kernel. To flash an SD card image, run the following command:

```
$ bunzip2 -dk -f <image_name>.sdcard.bz2
$ sudo dd if=<image name>.sdcard  of=/dev/sd<partition> bs=1M conv=fsync
```

For more information on flashing, see Section "Preparing an SD/MMC Card to Boot" in the *i.MX Linux® User's Guide* (IMXLUG).

6.2 Manufacturing Tool, MFGTool

One way to place an image on a device is to use the MFGTool. The recipes used to build a manufacturing tool image are `linux-imx-mfgtool` and `u-boot-mfgtool`.

To build a manufacturing image do the following:

```
$ bitbake fsl-image-mfgtool-initramfs
```

For more details on how to use the manufacturing tool, see Section "Serial download mode for the Manufacturing Tool" in the *i.MX Linux® User's Guide* (IMXLUG).

7 Customization

There are three scenarios to build and customize on i.MX Linux OS:

- Building i.MX Yocto Project BSP and validating on an i.MX reference board. The directions in this document describe this method in details.
- Customizing kernel and creating a custom board and device tree with kernel and U-Boot. For more details on how to build an SDK and set up a host machine for building the kernel and U-Boot only outside of the Yocto Project build environment, see Chapter 4.5.12 How to Build U-Boot and Kernel in Standalone Environment in the *i.MX User's Guide* (IMXLUG).
- Customizing a distribution adding or removing packaging from the BSP provided for i.MX Linux releases by creating a custom Yocto Project layer. i.MX provides multiple demo examples to show a custom layer on top of an i.MX BSP release using AGL, Genivi, IOT Gateway, and XBMC on the i.MX git repository on Code Aurora Forum. The remaining sections in this document provides instructions for creating a custom DISTRO and board configuration.

7.1 Creating a custom DISTRO

A custom DISTRO can configure a custom build environment. The DISTRO files released fsl-imx-x11, fsl-imx-wayland, fsl-imx-xwayland, and fsl-imx-fb all show configurations for specific graphical backends. DISTROs can also be used to configure other parameters such as kernel, uboot, and gstreamer. The i.MX DISTRO files are set to create a custom build environment required for testing our i.MX Linux OS BSP releases.

It is recommended for each customer to create their own distro file and use that for setting providers, versions, and custom configurations for their build environment. A DISTRO is created by copying an existing distro file, or including one like poky.conf and adding additional changes, or including one of the i.MX DISTROs and using that as a starting point.

7.2 Creating a custom board configuration

Vendors who are developing reference boards may want to add their board to the FSL Community BSP. Having the new machine supported by the FSL Community BSP makes it easy to share source code with the community, and allows for feedback from the community.

The Yocto Project makes it easy to create and share a BSP for a new i.MX based board. The upstreaming process should start when a Linux OS kernel and a bootloader are working and tested for that machine. It is highly important to have a stable Linux kernel and bootloader (for example, U-Boot) to be pointed to in the machine configuration file, to be the default one used for that machine.

Another important step is to determinate a maintainer for the new machine. The maintainer is the one responsible for keeping the set of main packages working for that board. The machine maintainer should keep the kernel and bootloader updated, and the user-space packages tested for that machine. For more information on the machine maintainer role, see FSL Community BSP Release Notes 2.4 Documentation.

The steps needed are listed below.

1. Customize the kernel config files as needed. The kernel config file is location in arch/arm/configs and the vendor kernel recipe should customize a version loaded through the kernel recipe.

2. Customize U-Boot as needed. See the *i.MX BSP Porting Guide* (IMXBSPPG) for details on this.

3. Assign someone to be the maintainer of the board. This person makes sure that files are updated as needed so the build always works. For more information see FSL Community BSP Release Notes 2.4 Documentation.

4. Set up the Yocto Project build as described in the Yocto Project community instructions, as shown below. Use the community master branch.
   a. Download the needed host package, depending on your host Linux OS distribution, from Yocto Project Quick Start.

   b. Download repo with the command:

      ```
      $ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
      ```

   c. Create a directory to keep everything in. Any directory name can be used. This document is using `fsl-community-bsp`.

      ```
      $ mkdir fsl-community-bsp
      ```

   d. `$ cd fsl-community-bsp`

   e. Initialize the repo with the master branch of the repository.

      ```
      $ repo init -u https://github.com/Freescale/fsl-community-bsp-platform -b master
      ```

   f. Get the recipes that will be used to build.

      ```
      $ repo sync
      ```

   g. Set up the environment with:

      ```
      $ source setup-environment build
      ```

5. Choose a similar machine file in fsl-community-bsp/sources/meta-freescale-3rdparty/conf/machine and copy it, using a name indicative of your board. Edit the new board file with the information about your board. Change the name and description at least. Add MACHINE_FEATURE. See www.yoctoproject.org/docs/2.1/ref-manual/ref-manual.html#ref-features-machine.

6. Test your changes with the latest community master branch, making sure everything works well. Use at least `core-image-minimal`.

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

7. Prepare the patches. Follow the style guide at www.openembedded.org/wiki/Styleguide and git.yoctoproject.org/cgit/cgit.cgi/meta-freescale/tree/README in the section entitled *Contributing*.

8. Upstream into meta-freescale-3rdparty. To upstream, send the patches to meta-freescale@yoctoproject.org.

Appendix A Frequently Asked Questions

A.1 Quick Start

This section summarizes how to set up the Yocto Project on a Linux machine and build an image. Detailed explanations of what this means are in the sections above.

Install the `repo` utility:

To get the BSP you need to have "repo" installed. This only needs to be done once.

```
$: mkdir ~/bin
$: curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$: chmod a+x ~/bin/repo
$: PATH=${PATH}:~/bin
```

Download the BSP Yocto Project Environment.

Use the correct name for the release desired in the -b option for `repo init`. This needs to be done once for each release and sets the distribution for the directory created in the first step. `repo sync` can be run to update the recipes under `sources` to the latest.

NOTE

https://source.codeaurora.org/external/imx/imx-manifest/tree/?h=imx-linux-rocko has a list of all manifest files supported in this release.

Setup for Specific Backends

MX8 Framebuffer and X11 distros are not supported. Only use these for MX6 and MX7 SoC.

Setup for X11

```
$: DISTRO=fsl-imx-X11 MACHINE=<machine name> source fsl-setup-release.sh -b build-x11
```

Setup for FB

```
$: DISTRO=fsl-imx-fb MACHINE=<machine name> source fsl-setup-release.sh -b build-fb
```

Setup for Wayland

```
$: DISTRO=fsl-imx-wayland MACHINE=<machine name> source fsl-setup-release.sh -b build-wayland
```

Setup for XWayland

```
$: DISTRO=fsl-imx-xwayland MACHINE=<machine name> source fsl-setup-release.sh -b build-
xwayland
```

Build For All Backends

Build without Qt

```
$: bitbake fsl-image-validation-imx
```

Build with Qt 5

```
$: bitbake fsl-image-qt5-validation-imx
```

A.2 Local configuration tuning

A Yocto Project build can take considerable build resources both in time and disk usage, especially when building in multiple build directories. There are methods to optimize this, for example, use a shared sstate cache (caches the state of the build) and downloads directory (holds the downloaded packages). These can be set to be at any location in the `local.conf` file by adding statements such as these:

```
DL_DIR="/opt/freescale/yocto/imx/download"
SSTATE_DIR="/opt/freescale/yocto/imx/sstate-cache"
```

The directories need to already exist and have appropriate permissions. The shared sstate helps when multiple build directories are set, each of which uses a shared cache to minimize the build time. A shared download directory minimizes the fetch time. Without these settings, Yocto Project defaults to the build directory for the sstate cache and downloads.

Every package downloaded in the DL_DIR directory is marked with a `<package name>.done`. If your network has a problem fetching a package, you can manually copy the backup version of package to the DL_DIR directory and create a <package_name>.done file with the touch command. Then run the bitbake command: `bitbake <component>`.

For more information, see the Yocto Project Reference Manual.

A.3 Recipes

Each component is built by using a recipe. For new components, a recipe must be created to point to the source (SRC_URI) and specify patches, if applicable. The Yocto Project environment builds from a makefile in the location specified by the SRC_URI in the recipe. When a build is established from auto tools, a recipe should inherit autotools and pkgconfig. Makefiles must allow CC to be overridden by Cross Compile tools to get the package built with Yocto Project.

Some components have recipes but need additional patches or updates. This can be done by using a bbappend recipe. This appends to an existing recipe details about the updated source. For example, a bbappend recipe to include a new patch should have the following contents:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
SRC_URI += file://<patch name>.patch
```

FILESEXTRAPATHS_prepend tells Yocto Project to look in the directory listed to find the patch listed in SRC_URI.

NOTE

If a bbappend recipe is not picked up, view the fetch log file (log.do_fetch) under the work folder to check whether the related patches are included or not. Sometimes a git version of the recipe is being used instead of the version in the bbappend files.

A.4 How to select additional packages

Additional packages can be added to images if there is a recipe provided for that package. A searchable list of recipes provided by the community can be found at layers.openembedded.org/. You can search to see if an application already has a Yocto Project recipe and find where to download it from.

A.4.1 Updating an image

An image is a set of packages and the environment configuration.

An image file (such as fsl-image-gui.bb) defines the packages that go inside the file system. Root file systems, kernels, modules, and the U-Boot binary are available in `build/tmp/deploy/images/`.

NOTE

You can build packages without including it in an image, but you must rebuild the image if you want the package installed automatically on a rootfs.

A.4.2 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. For example, a multimedia task could determine, according to the machine, whether the VPU package is built or not, so the selection of multimedia packages may be automated for every board supported by the BSP, and only the multimedia package is included in the image.

Additional packages can be installed by adding the following line in `<build dir>/local.conf`.

```
CORE_IMAGE_EXTRA_INSTALL += "<package_ name1 package_name2>"
```

There are many package groups. They are in subdirectories named "packagegroup" or "packagegroups".

A.4.3 Preferred version

The preferred version is used to specify the preferred version of a recipe to use for a specific component. A component may have multiple recipes in different layers and a preferred version points to a specific version to use.

In the meta-fsl-bsp-release layer, in `layer.conf`, preferred versions are set for all the recipes to provide a static system for a production environment. These preferred version settings are used for formal i.MX releases but are not essential for future development.

Preferred versions also help when previous versions may cause confusion about which recipe should be used. For example, previous recipes for imx-test and imx-lib used a year-month versioning which has changed to <kernel-version> versioning. Without a preferred version, an older version might be picked up. Recipes that have _git versions are usually picked over other recipes, unless a preferred version is set. To set a preferred version, put the following in `local.conf`.

```
PREFERRED_VERSION_<component>_<soc family> = "<version>"
```

For example, imx-lib would be:

```
PREFERRED_VERSION_imx-lib_mx6 = ""
```

See the Yocto Project manuals for more information on using preferred versions.

A.4.4 Preferred provider

The preferred provider is used to specify the preferred provider for a specific component. A component can have multiple providers. For example, the Linux kernel can be provided by i.MX or by `kernel.org` and preferred provider states the provider to use.

For example, U-Boot is provided by both the community via denx.de and i.MX. The community provider is specified by u-boot-fslc. The i.MX provider is specified by u-boot-imx. To state a preferred provider, put the following in `local.conf`:

```
PREFERRED_PROVIDER_<component>_<soc family> = "<provider>"
PREFERRED_PROVIDER_u-boot_mx6 = "u-boot-imx"
```

A.4.5 SoC family

The SoC family documents a class of changes that apply to a specific set of system chips. In each machine configuration file, the machine is listed with a specific SoC family. For example, i.MX 6DualLite Sabre-SD is listed under the i.MX 6 and i.MX 6DualLite SoC families. i.MX 6Solo Sabre-auto is listed under the i.MX 6 and i.MX 6Solo SoC families. Some changes can be targeted to a specific SoC family in `local.conf` to override a change in a machine configuration file. The following is an example of a change to an mx6dlsabresd kernel setting.

```
KERNEL_DEVICETREE_mx6dl = "imx6dl-sabresd.dts"
```

SoC families are useful when making a change that is specific only for a class of hardware. For example, i.MX 28 EVK does not have a Video Processing Unit (VPU), so all the settings for VPU should use i.MX 5 or i.MX 6 to be specific to the correct class of chips.

A.4.6 Bitbake logs

Bitbake logs the build and package processes in the temp directory in `tmp/work/<architecture>/<component>/temp`.

If a component fails to fetch a package, the log showing the errors is in the file log.do_fetch.

If a component fails to compile, the log showing the errors is in the file log.do_compile.

Sometimes a component does not deploy as expected. Check the directories under the build component directory (tmp/work/<architecture>/<component>). Check the package, packages-split, and sysroot* directories of each recipe to see if the files are placed there (where they are staged prior to being copied to the deploy directory).

Appendix B References

- For details on boot switches, see Section "How to Boot the i.MX Boards" in the *i.MX Linux® User's Guide* (IMXLUG).
- For how to download images using U-Boot, see Section "Downloading Images Using U-Boot" in the *i.MX Linux® User's Guide* (IMXLUG).
- For how to set up an SD/MMC card, see Section "Preparing an SD/MMC Card to Boot" in the *i.MX Linux® User's Guide* (IMXLUG).

Appendix C Revision History

This table provides the revision history.

Table C-1. Revision history

Revision number	Date	Substantive changes
L4.9.51_imx8qxp-alpha	11/2017	Initial release
L4.9.51_imx8qm-beta1	12/2017	Added i.MX 8QuadMax
L4.9.51_imx8mq-beta	12/2017	Added i.MX 8MQuad
L4.9.51_8qm-beta2/8qxp-beta	02/2018	Added i.MX 8QuadMax Beta2 and i.MX 8QuadXPlus Beta
L4.9.51_imx8mq-ga	03/2018	Added i.MX 8MQuad GA

Table continues on the next page...

Table C-1. Revision history (continued)

Revision number	Date	Substantive changes
L4.9.88_2.0.0-ga	05/2018	i.MX 7ULP and i.MX 8MQuad GA release
L4.9.88_2.1.0_8mm-alpha	06/2018	i.MX 8MMini Alpha release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, AMBA, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and µVision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. Arm7, Arm9, Arm11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, Mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.

Document Number IMXLXYOCTOUG
Revision L4.9.88_2.1.0_8mm-alpha, 06/2018