Copyright 2021 © Embedded Artists AB

Working with Yocto to Build Linux

Embedded Artists AB
Rundelsgatan 14
SE-211 36 Malmö
Sweden

http://www.EmbeddedArtists.com

Copyright 2021 © Embedded Artists AB. All rights reserved.

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Embedded Artists AB.

Disclaimer

Embedded Artists AB makes no representation or warranties with respect to the contents hereof and specifically disclaim any implied warranties or merchantability or fitness for any particular purpose. Information in this publication is subject to change without notice and does not represent a commitment on the part of Embedded Artists AB.

Feedback

We appreciate any feedback you may have for improvements on this document. Send your comments by using the contact form: www.embeddedartists.com/contact.

Trademarks

All brand and product names mentioned herein are trademarks, services marks, registered trademarks, or registered service marks of their respective owners and should be treated as such.

Table of Contents

Copyright 2021 © Embedded Artists AB

Copyright 2021 © Embedded Artists AB

Copyright 2021 © Embedded Artists AB

1 Document Revision History

Revision	*Date*	*Description*
A	2015-09-30	First release
B	2015-11-12	- Added instructions for iMX6 UltraLite COM Board - Added Section 5.4
C	2016-01-12	- Updated Table 1 with information about new 3.14.52 branch - Updated section 4.3 with new instructions
D	2016-02-15	- Updated section 5.1
E	2016-04-06	- Added new machine configuration for iMX7 Dual uCOM Board in section 4.2
F	2016-04-21	- Added new machine configuration for iMX6 DualLite COM Board in section 4.2
G	2016-08-30	- Added new machine configuration for iMX7 Dual COM Board in section 4.2
H	2016-10-20	- Linux **4.1.15** is now supported. Removed description of working with 3.14.28.
I	2016-12-21	- Added section 8.1 describing how to build and use meta-toolchain. - Added section 10.1 describing how to put the root file system on an SD card
J	2017-05-19	- Updated section 3.4 – New distribution is now available. Linux version is still 4.1.15. U-boot has been updated to **2016.03**.
K	2018-01-29	- Removed chapter ”Ubuntu 14.04 Virtual Machine” since it is out-dated. Use instructions in chapter 7 instead. - Updated section 3.4 with 4.9.11 branch (Linux kernel **4.9.11** is now supported) - Updated section 8.1 - Added sections 10.2 and 10.3
L	2018-11-20	- Linux 4.9.123 and u-boot 2017.03 now supported on ea-yocto-base branch ea-4.9.123. Section 3.4 updated.
M	2019-02-06	- Linux 4.14.78 and u-boot 2018.03 now supported on ea-yocto-base branch ea-4.14.78. Added ea-image-base. Sections 2, 3.4, 4.1, 5.1.3 updated.
N	2019-05-03	- Added section about uuu (intro in section 5 + new section 5.2)
O	2019-05-08	- Added information about iMX8M Quad COM
P	2019-10-04	- Added information about iMX8M Mini uCOM and iMX7ULP uCOM
Q	2019-11-14	- Corrected GCC version command - Added section 9.3.1 about iMX8 and u-boot - Updated example paths to match latest u-boot/Linux branches
R	2020-05-06	- Added information about iMX8M Nano uCOM - Added chapter 9 about how to customize Yocto - Added section 10.4 about how to use devtool - Added section 10.5 about how to use caching to reduce build times.

Copyright 2021 © Embedded Artists AB

		- Added chapter 11 – Frequently Asked Questions
S	2020-09-17	- Linux 5.4.24 and u-boot 2020.04 now supported on ea-yocto-base. Section 3.4 updated.
T	2021-01-13	- Linux 5.4.47 now supported on ea-yocto-base. Section 3.4 updated.
U	2021-09-21	- Linux 5.10.35 and u-boot 2021.04 now supported on ea-yocto-base. Section 3.4 updated.

Copyright 2021 © Embedded Artists AB

2 Introduction

This document provides you with step-by-step instructions to setup the Yocto build system, build boot loaders, Linux kernel and file system for the Embedded Artists i.MX6, i.MX7, and i.MX8 based COM boards.

Additional documentation you might need is.

- *NXP Yocto Project User's Guide* – The document is available in the Linux bundle found at the software and tools section on NXP's website.
- The *Getting Started* document for the board you are using.
- Yocto project overview - https://www.yoctoproject.org/software-overview/
- Yocto Project Quick Start
- Yocto Project Reference Manual
- Yocto Training – Instructions for using Yocto at the NXP community site
- Bitbake Cheat Sheet: https://elinux.org/Bitbake_Cheat_Sheet

2.1 Conventions

A number of conventions have been used throughout to help the reader better understand the content of the document.

`Constant width text` – is used for file system paths and command, utility and tool names.

```
$ This field illustrates user input in a terminal running on the
development workstation, i.e., on the workstation where you edit,
configure and build Linux
```

```
# This field illustrates user input on the target hardware, i.e.,
input given to the terminal attached to the COM Board
```

```
This field is used to illustrate example code or excerpt from a
document.
```

Copyright 2021 © Embedded Artists AB

3 Linux Host Setup

3.1 Introduction

The Yocto build system requires a Linux host machine. You can either run this host as a standalone / native computer or as a virtual machine on, for example, a Microsoft Windows PC. The minimum available hard disk space is 50 GB, but it is recommended that the host machine has at least 120 GB to be able to build the largest image / distribution.

Several Linux distributions are supported by the Yocto project. Please refer to the Supported Linux Distributions section in the Yocto reference manual for a complete list.

The instructions in this document have been tested on an Ubuntu 14.04 and a Lubuntu 14.04 distribution.

- Chapter 7 describes how to install and setup Lubuntu 14.04 as a virtual machine. A VMWare appliance is not downloaded in this case. Instead an ISO image of Lubuntu is downloaded and installed into a newly created virtual machine.

3.2 Required Packages

The Yocto Project requires several packages to be installed on the host machine. If you are using any of the distributions in section 3.1 follow the instructions below. If you, however, are using another distribution refer to the Required Packages for the Host Development System section in the Yocto reference manual.

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

$ 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

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

3.3 Install the `repo` tool

The `repo` tool has been developed to make it easier to manage multiple Git repositories. Instead of downloading each repository separately the `repo` tool can download all with one instruction. Download and install the tool by following the instructions below.

1. Create a directory for the tool. The example below creates a directory named `bin` in your home folder.

```
$ mkdir ~/bin
```

2. Download the tool

```
$ curl http://commondatastorage.googleapis.com/git-repo-
downloads/repo > ~/bin/repo
```

3. Make the tool executable

```
$ chmod a+x ~/bin/repo
```

Copyright 2021 © Embedded Artists AB

4. Add the directory to the PATH variable. The line below could be added to your .bashrc file so the path is available in each started shell/terminal

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

3.4 Download Yocto recipes

The Yocto project consists of many recipes used when building an image. These recipes come from several repositories and the repo tool is used to download these repositories.

In step 3 below a branch must be selected of the ea-yocto-base repository. The table below lists the available branches.

Branch name	Description
ea-5.10.35	u-boot: 2021.04. Linux: 5.10.35.
ea-5.4.47	u-boot: 2020.04. Linux: 5.4.47.
ea-5.4.24	u-boot: 2020.04. Linux: 5.4.24.
ea-4.14.98	u-boot: 2018.03. Linux: 4.14.98.
ea-4.14.78	u-boot: 2018.03. Linux: 4.14.78.
ea-4.9.123	u-boot: 2017.03. Linux: 4.9.123.
ea-4.9.11_1.0.0	u-boot: 2016.03. Linux: 4.9.11.
ea-4.1.15_2.0.0	u-boot: 2016.03. Linux: 4.1.15.

Table 1 - ea-yocto-base branches

1. Create a directory for the downloaded files (`ea-bsp` in the example below)

```
$ mkdir ea-bsp
$ cd ea-bsp
```

2. Configure Git if you haven't already done so. Change "Your name" to your actual name and "Your e-mail" to your e-mail address.

```
$ git config --global user.name "Your name"
$ git config --global user.email "Your e-mail"
```

3. Initialize repo. The file containing all needed repositories is downloaded in this step. Change <selected branch> to a branch name according to Table 1.

```
$ repo init -u https://github.com/embeddedartists/ea-yocto-base -b
<selected branch>
```

4. Start to download files

```
$ repo sync
```

All files have now been downloaded into the `ea-bsp` directory. Most of the files will actually be available in the sub-directory called `sources`.

Copyright 2021 © Embedded Artists AB

4 Building Images

Yocto is using the BitBake tool to generate complete Linux images/distributions, that is, all needed to boot and run a Linux system. This is typically boot loader(s), Linux kernel, and root file system with selected utilities and applications.

4.1 Available Images

The recipes that have been downloaded contain many different images. The table below describe only a few of the images that are available.

Image name	Description
ea-image-base	Only available on branch ea-4.14.78 and later. Based on core-image-base and added packages for peripheral testing and verification. This is Embedded Artists **default image** and the one included in the pre-build packages on **imx.embeddedartsits.com**.
core-image-minimal	A small image allowing a device to boot
core-image-base	A console only image that fully supports the target device
fsl-image-gui	Build an image with GUI support, but without Qt content. Works with X11, DirectFB, frame buffer and Wayland backends.
fsl-image-qt5	Build a Qt5 image. Works with X11, Frame buffer, and Wayland backends.
fsl-image-mfgtool-initramfs	Build u-boot, kernel and file system that can be used by NXP's Manufacturing tool.
meta-toolchain	Builds an installable toolchain (cross-complier)
meta-toolchain-qt5	Builds toolchain/SDK for Qt5. Can be used when developing Qt5 applications.

Usually, the name of the file that defines an image contains the string "*image*". You can search for such files using the `find` command. Please note that not all images follow this naming convention. The toolchain images are images that doesn't contain "image" in the name.

```
$ cd ~/ea-bsp/sources
$ find -name *image*.bb
```

An alternative way to find images is to use the `bitbake-layers` command.

```
$ bitbake-layers show-recipes | grep image
```

4.2 Machine Configurations

A machine configuration must be specified before a build can be started.

The table below contains the machine configurations available for Embedded Artists boards. It is also possible to find the configuration files in the directory `~/ea-bsp/sources/meta-ea/conf/machine`.

Machine	Description
imx6sxea-com	Machine configuration for Embedded Artists iMX6 SoloX

Copyright 2021 © Embedded Artists AB

	COM Board / Kit
imx6qea-com	Machine configuration for Embedded Artists iMX6 Quad COM Board / Kit
imx6dlea-com	Machine configuration for Embedded Artists iMX6 DualLite COM Board / Kit
imx6ulea-com	Machine configuration for Embedded Artists iMX6 UltraLite COM Board / Kit
imx7dea-ucom	Machine configuration for Embedded Artists iMX7 Dual uCOM Board / Kit
imx7dea-com	Machine configuration for Embedded Artists iMX7 Dual COM Board / Kit
imx8mqea-com	Machine configuration for Embedded Artists iMX8M Quad COM Board / Kit
imx8mmea-ucom	Machine configuration for Embedded Artists iMX8M Mini uCOM Board / Kit
imx8mnea-ucom	Machine configuration for Embedded Artists iMX8M Nano uCOM Board / Kit
imx7ulpea-ucom	Machine configuration for Embedded Artists iMX7ULP uCOM Board / Kit

4.3 Initialize Build

Before starting the build, it must be initialized. In this step the build directory and local configuration files are created.

A distribution must be selected when initializing the build, see section 4.3.1 for available distributions.

In the example below the machine `imx6sxea-com`, the build directory `build_dir` and the `fsl-imx-fb` distribution is selected.

```
$ DISTRO=fsl-imx-fb MACHINE=imx6sxea-com source ea-setup-
release.sh -b build_dir
```

4.3.1 Distro configurations

When initializing a build a distribution is specified. Several different are supported as listed in the table below.

DISTRO	Description
fsl-imx-fb	Linux Frame buffer – no X11 or Wayland. **NOTE**: Not supported by iMX8M Quad, iMX8M Mini and iMX8M Nano.
fsl-imx-x11	Only X11 (X Window System) graphics
fsl-imx-wayland	Wayland Weston graphics
fsl-imx-xwayland	Wayland graphics and X11. X11 applications using EGL are not supported

4.3.2 Restart a Build

If you need to restart a build in a new terminal window or after a restart of the host computer you don't need to run the `ea-setup-release.sh` script again. Instead, you run the `setup-`

Copyright 2021 © Embedded Artists AB

environment script. If you don't run the setup-environment script you won't have access to needed tools and utilities, such as bitbake.

```
$ source setup-environment build_dir
```

4.4 Starting the Build

Everything has now been setup to start the actual build. The example below shows how the ea-image-base image is being built. Please note that depending on the capabilities of your host computer building an image can take many hours.

```
$ bitbake ea-image-base
```

When the build has finished the images will be available in the directory specified below. Please note that this directory will be different if you are using another build directory or machine configuration.

~/ea-bsp/build_dir/tmp/deploy/images/imx6sxea-com.

Go to chapter 5 for instructions of how to deploy images to the target hardware.

4.5 Bitbake Options

This section contains a few examples of how to use bitbake. This is by no means a complete list of all available bitbake options, but instead a list of examples that you might find useful.

4.5.1 Clean Build for a Specific Image/Recipe

The "-c" option executes a specific task for an image or recipe. In the example below a previous build of the u-boot boot loader is cleaned.

```
$ bitbake -c cleansstate u-boot-ea
```

To build u-boot after it has been cleaned just specify the image name u-boot-ea.

```
$ bitbake u-boot-ea
```

4.5.2 Kernel Configuration

If you would like to check or change the Linux kernel configuration you can start the Linux configuration tool using the option below.

```
$ bitbake -c menuconfig linux-ea
```

4.5.3 Show Yocto Layer Append Dependencies

One nice feature with Yocto is the ability to extend an already existing recipe. This is done by so called bbappend files. The Embedded Artists layer (meta-ea) is constructed this way, that is, it is appending to existing recipes such as u-boot (u-boot-imx) and Linux kernel (linux-imx).

When creating your own append files it can be useful to get a list of "appends" that are considered to be active for you build.

```
$ bitbake-layers show-appends
```

Copyright 2021 © Embedded Artists AB

5 Deploying Images

NXP's Manufacturing Tool currently exists in two versions. MFGTool is the old version and UUU is the new version.

	MFGTool	UUU
MFGTool version	V2	V3
Actively developed	No	Yes
OS Support	Windows only	Windows + Linux
Source Code Available	No	Yes, GitHub

As of May 2019, Embedded Artists plan to keep MFGTool support for old kernel releases (i.e. prior to Linux 4.14.78). For Linux 4.14.78 there will be a transition time where both MFGTool and UUU zip files will be available but at some point, only the UUU zip file will be updated. For all releases after Linux 4.14.78, only UUU will be supported.

5.1 Manufacturing Tool

NXP's Manufacturing Tool (MFGTool) can be used to write images to the board. This tool is sending files and instructions over USB and the board must be set in OTG boot mode for it to work.

At the moment the tool is only available for Microsoft Windows and a version which has been prepared for Embedded Artists boards is available on http://imx.embeddedartists.com/ for the board you are using.

5.1.1 Download the Tool

Download the zip file containing the manufacturing tool from http://imx.embeddedartists.com/

Unpack this zip file somewhere on your computer running Microsoft Windows. Below is a description of some of the content in the zip file.

- `mfgtool` (root): Contains the actual tool as well as vbs files which can be used to run a specific download configuration.
- `mfgtool/Document`: Contains documentation for the manufacturing tool. This documentation has been written by NXP.
- `mfgtool/Profiles/Linux/OS Firmware/ucl2.xml`: This file contains the actual download configurations.
- `mfgtool/Profiles/Linux/OS Firmware/files`: Contains pre-compiled versions of images. The tool will look in this directory when selecting images to download to the board.

5.1.2 Prepare hardware

Begin by reading the *Getting Started* document for the board you are using. It shows how to setup the board and also gives an overview of the hardware.

The next step is to put the board into OTG boot mode. If you have an early version of the iMX6 SoloX Developer's Kit, that is, a version with a DIP switch mounted as shown in Figure 3, read section 5.1.4 for instructions. Read section 5.1.3 if you have another iMX based developer's kit or a new version of the iMX6 SoloX Developer's kit.

Copyright 2021 © Embedded Artists AB

5.1.3 OTG boot mode – J2 jumper

To download images using the manufacturing tool the board must be put into OTG boot mode.

This is accomplished by closing the J2 jumper on the Carrier board; see Figure 1 to locate the jumper. Please note that in the figure the jumper is in open state which means that the COM board will boot from eMMC.

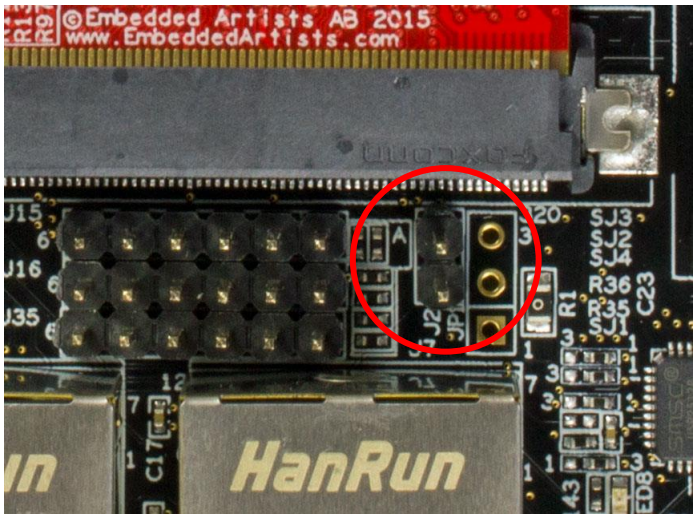


Figure 1 - J2 jumper (opened state) on COM Carrier Board V1

Figure 2 - J2 jumper (opened state) on a COM Carrier Board V2

5.1.4 OTG boot mode – DIP switches

The first version of the iMX6 SoloX COM boards had boot jumpers (DIP switches) mounted on them, see Figure 3. If you have such a COM board you need to set the boot jumpers as described below to force it into OTG boot mode instead of closing jumper J2 as described above.

Copyright 2021 © Embedded Artists AB

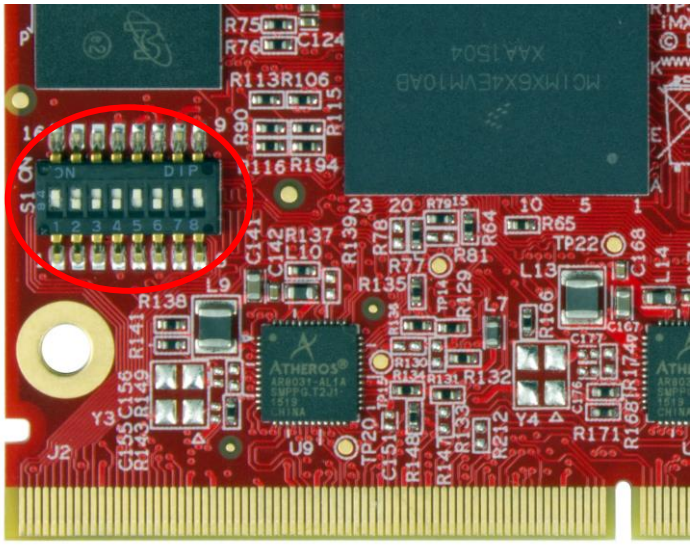


Figure 3 - DIP switch on iMX6 SoloX COM board

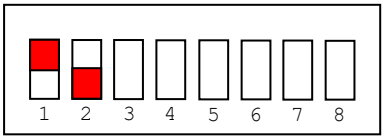


Figure 4 – Boot jumper setting for USB OTG

1. Set the boot jumpers in OTG boot mode as shown in Figure 4. First jumper in the *up* position and the second jumper in *down* position. Please note that the orientation of the DIP switch component can be different on different COM boards. Make sure you are changing the correct jumper by looking at the number by the jumper.
2. Make sure a USB cable is connected between the board (micro-B connector on carrier board) and your PC
3. Reset the board

Note: When you want to boot the software from eMMC you have to reverse the setting, that is, first jumper in *down* position and second jumper in *up* position.

5.1.5 Configurations

Several configurations of the tool have been prepared in order to help you download specific images. Shortcuts to these configurations are available as vbs files in the root of the MFGTool directory. All you need to do is to double-click on one of these files and the manufacturing tool will start.

- `ea-com-emmc_bootloaders.vbs` – will install only the bootloaders. This should only be used if you want to restore the bootloaders or download your own bootloaders to the board.
- `ea-com-emmc_kernel.vbs` – will install kernel and dtb files. This should only be used if you want to update the kernel or dtb file.
- `ea-com-emmc_full_tar.vbs` – will install bootloaders, Linux kernel and root file system. The root file system will be installed from a tar.bz2 file.
- `ea-com-emmc_update_rootfs.vbs` – will only download the root file system (the ext3 file) to the board.

Copyright 2021 © Embedded Artists AB

5.1.6 Download Your Own Images

The simplest way to download your own images is to replace the existing file(s) with your file(s). If you keep the file names intact the vbs files will download your version of the file.

In many cases you would, however, like to keep the pre-compiled versions of the files and just add your own files. You could then copy an existing vbs file that is closest to what you want to do, for example, copy the "...`update_rootfs.vbs`" file if you want to update only the root file system. When you open this copy of the file you can see that several options are sent to the manufacturing tool. For example, one option is called `board` and another is called `rootfs`. These options are used in the configuration file (`ucl2.xml`) to construct the actual file names of the files the tool is accessing.

The name of the file for the root file system is, for example, constructed by using both the `rootfs` and `board` options. This is how it looks in the ucl2.xml file: `files/%rootfs%-%board%.rootfs.tar.bz2`.

One more alternative is to create a new configuration. In this case you need to open the ucl2.xml file and then copy an existing configuration (the LIST tag and all its children), give it a new name (change the name attribute) and then modify the instructions so your images are downloaded.

5.1.7 Run the Tool

Double click on one of the vbs files to start the manufacturing tool. If the tool can find the board it will write "HID-compliant device" in the status field, see Figure 5 below. If it cannot find the board it will write "No Device Connected".

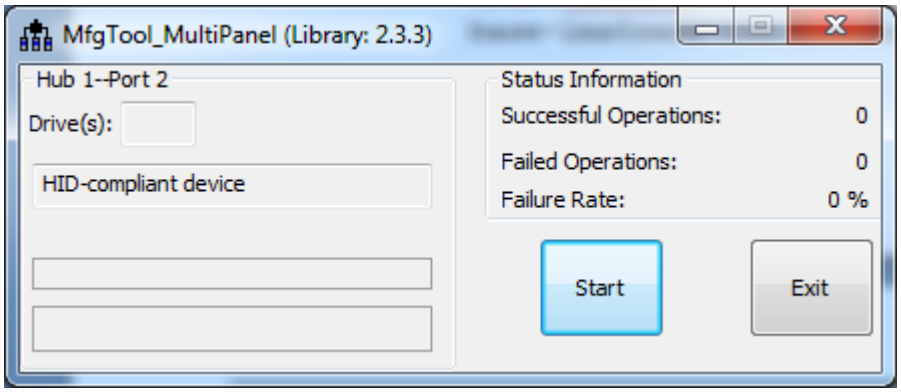


Figure 5 - Manufacturing Tool

Click the Start button to start the download of files. If all operations are successful the progress bars will turn green, see Figure 6. Click the Stop button and then Exit to close the manufacturing tool. If an operation fails the progress bars will turn red. In this case it can be helpful to have a look at the log `MfgTool.log` which is found in the same directory as the manufacturing tool.

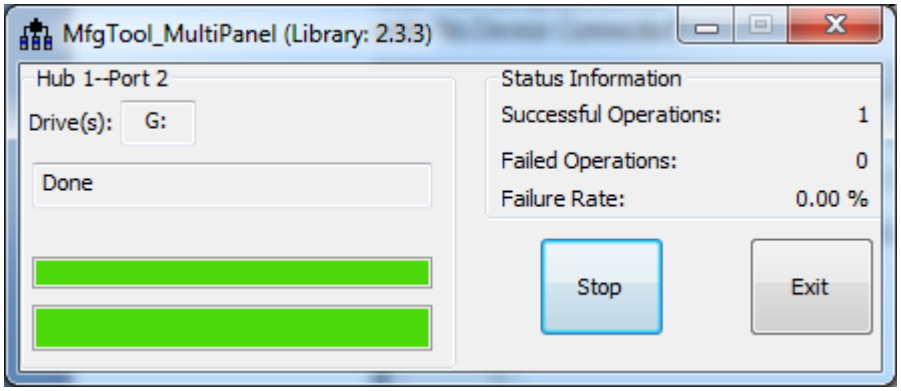


Figure 6 Manufacturing Tool successful download

Copyright 2021 © Embedded Artists AB

5.2 UUU

UUU (Universal Update Utility) is version 3 of MFGTool but it has been rewritten, is publicly available on GitHub (https://github.com/NXPmicro/mfgtools) and it can be run on both Windows and Linux while the older versions of MFGTool were limited to Windows only.

UUU can be used to write images to the board. This tool is sending files and instructions over USB and the board must be set in OTG boot mode for it to work.

Starting with Linux 4.14.78, a uuu-zip file will be made available in addition to the mfgtool zip files on http://imx.embeddedartists.com/ for the board you are using. Note that iMX8M Quad COM, iMX8M Mini uCOM, iMX8M Nano, and iMX7ULP uCOM boards will not have an mfgtool zip file.

Prerequisites:

- Ubuntu 16.04 or above, 64-bit
- Windows 10, 64-bit
- Windows 7, 64-bit - note that there might be problems with drivers and that it might not even work with the driver fixes applied even if the documentation says it does. The Windows 7 specific instructions can be found here: https://github.com/NXPmicro/mfgtools/wiki/WIN7-User-Guide

Useful links:

- UUU on GitHub: https://github.com/NXPmicro/mfgtools
- UUU release page: https://github.com/NXPmicro/mfgtools/releases

5.2.1 Download the Tool

Download the zip file for the board you are using from http://imx.embeddedartists.com/

Unpack this zip file somewhere on your computer. Below is a description of some of the content in the zip file.

- `uuu` (root): Contains a README file.
- `uuu/*.uuu`: The different download configurations.
- `uuu/files/`: Contains pre-compiled versions of images. The tool will look in this directory when selecting images to download to the board.

5.2.2 Prepare hardware

The instructions here are identical to the ones for MFGTool available in 5.1.2

5.2.3 OTG boot mode – J2 jumper

The instructions here are identical to the ones for MFGTool available in 5.1.3

5.2.4 OTG boot mode – DIP switches

The instructions here are identical to the ones for MFGTool available in 5.1.4

5.2.5 Configurations

Several configurations (*.uuu files) for the tool have been prepared in order to help you download specific images.

- `bootloader.uuu` – will install only the bootloaders. This should only be used if you want to restore the bootloaders or download your own bootloaders to the board.

Copyright 2021 © Embedded Artists AB

- `bootloader_combined.uuu` – will install only the bootloaders. This is a faster alternative to bootloader.uuu but it requires a binary where SPL and the u-boot have been combined (see below). This should only be used if you want to restore the bootloaders or download your own bootloaders to the board.
- `kernel.uuu` – will install kernel and dtb files. This should only be used if you want to update the kernel or dtb files.
- `full_tar.uuu` – will install bootloaders, Linux kernel and root file system. The root file system will be installed from a tar.bz2 file.
- `raw_sdcard_example.uuu` – will overwrite the eMMC with the content of an sdcard file. The sdcard file is copied directly to the eMMC overwriting everything including bootloaders, Linux kernel and file system.

If you want to create the combined binary to use with bootloader_combined.uuu run the following commands in Linux:

```
$ dd if=SPL of=spl_and_uboot.bin bs=1024
$ dd if=u-boot.img of=spl_and_uboot.bin bs=1024 seek=68
```

5.2.6 Download Your Own Images

The uuu zip file that you download from http://imx.embeddedartists.com/ contain the latest build from Embedded Artists.

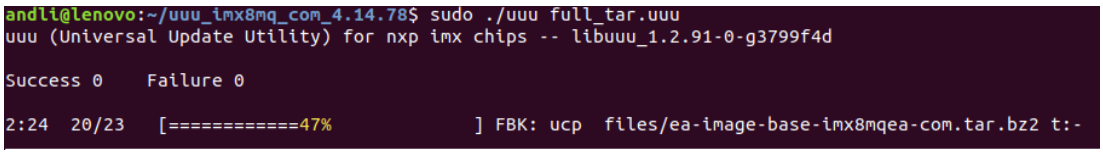
The simplest way to download your own images is to replace the existing file(s) with your own file(s). If you keep the file names intact the *.uuu configurations will download your version of the file.

5.2.7 Run the Tool in Ubuntu

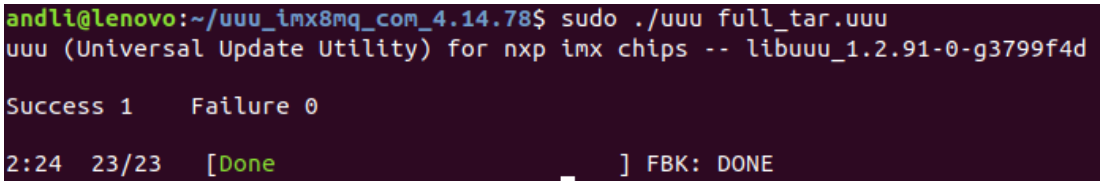
On Linux open a terminal, navigate to the folder where the uuu zip file was unpacked, make sure that the tool is executable and then execute the tool:

```
$ cd ~/uuu_imx8mq_com_4.14.78
$ chmod +x ./uuu
$ sudo ./uuu full_tar.uuu
```

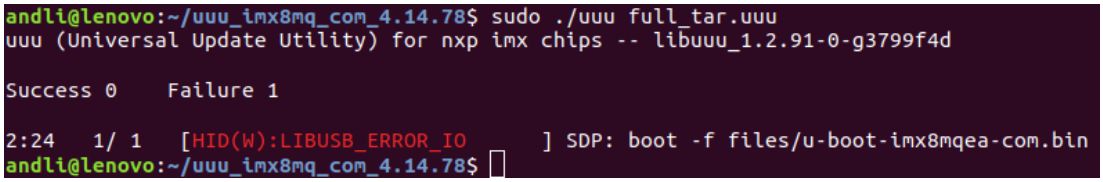
The terminal will show a progress bar like this while it is running:



After a successful run it will look like this:



If a problem occurs then the program will terminate and print an error message like this



Copyright 2021 © Embedded Artists AB

5.2.8 Run the Tool in Windows

On Windows open a Command Prompt, navigate to the folder where the uuu zip file was unpacked and then run the tool:

```
C:\> cd c:\temp\uuu_imx8mq_com_4.14.78
C:\temp\uuu_imx8mq_com_4.14.78> uuu.exe full_tar.uuu
```

The terminal will show a progress bar like this while it is running:

```
c:\temp\uuu_imx8mq_com_4.14.78>uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 0    Failure 0

1:23  20/23   [========>   34%           ] FBK: ucp  files/ea-image-base-imx8mqea-com.tar.bz2 t:-
```

After a successful run it will look like this:

```
c:\temp\uuu_imx8mq_com_4.14.78>uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 1    Failure 0

1:23  23/23   [Done                       ] FBK: DONE

c:\temp\uuu_imx8mq_com_4.14.78>
```

If a problem occurs then the program will terminate and print an error message like this

```
c:\temp\uuu_imx8mq_com_4.14.78>uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 0    Failure 1

1:23  20/23   [Bulk(R):LIBUSB_ERROR_PIPE   ] FBK: ucp  files/ea-image-base-imx8mqea-com.tar.bz2 t:-

c:\temp\uuu_imx8mq_com_4.14.78>
```

5.2.9 Troubleshoot

Some common problems and solutions:

- **The first time you run uuu on your computer it fails.**
  This is likely because of USB driver installation. Let the driver install, reset the hardware and then run the uuu command again. In Windows it is three different drivers that are needed so this procedure might have to be repeated three times - each time the procedure gets a little bit further.
- **UUU appears to hang with a "Wait for Known USB Device Appear..." message like this:**

  C:\Windows\System32\cmd.exe - uuu full_tar.uuu

  ```
  c:\temp\uuu_imx8mq_com_4.14.78>uuu full_tar.uuu
  uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

  Wait for Known USB Device Appear...
  ```

  This means that the hardware is either not connected to the computer with the USB cable or it

Copyright 2021 © Embedded Artists AB

is not in the OTG boot mode. Check 5.1.2 to 5.1.4 again and then run the uuu command again.

- **Windows 7 fail to flash with an error like this:**

```
3:14>Start Cmd:FB: acmd ${kboot} ${loadaddr} ${initrd_addr} ${fdt_addr}
3:14>Okay
New USB Device Attached at 3:14
3:14>Fail Failure open usb device
←[?25h
```

It could be due to a driver problem. Follow instructions here:
https://github.com/NXPmicro/mfgtools/wiki/WIN7-User-Guide

- **Windows 7 terminal does not appear as in the screenshots**
This is because Windows 7 does not support what the UUU tool calls "VT mode" so it defaults to verbose mode which has a lot more printouts and no progress bar.

- **Running raw_sdcard_example.uuu complains about a missing .sdcard file**
That file is not supplied in the downloaded zip file but you will find it in the "deploy" folder after you complete your own yocto build.

- **UUU in Ubuntu reports failure to open usb device:**

```
andli@lenovo:~/uuu_imx8mq_com_4.14.78$ ./uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 0    Failure 1

2:24    1/ 0   [Failure open usb device,Try ]
andli@lenovo:~/uuu_imx8mq_com_4.14.78$
```

This happens if the uuu program is not executed with the correct rights. Either use "sudo uuu" or setup udev rules so that sudo rights are not needed. The instructions for how to create the udev rules are built into the tool so run "uuu -udev" and then follow the steps:

```
andli@lenovo:~/uuu_imx8mq_com_4.14.78$ ./uuu -udev
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="012f", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0129", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0076", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0054", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0061", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0063", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0071", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="007d", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0080", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0128", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0126", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0135", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0134", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="012b", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0525", ATTRS{idProduct}=="b4a4", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0525", ATTRS{idProduct}=="b4a4", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0525", ATTRS{idProduct}=="b4a4", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="066f", ATTRS{idProduct}=="9afe", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="066f", ATTRS{idProduct}=="9bff", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0525", ATTRS{idProduct}=="a4a5", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="18d1", ATTRS{idProduct}=="0d02", MODE="0666"

1: put above udev run into /etc/udev/rules.d/99-uuu.rules
        sudo sh -c "uuu -udev >> /etc/udev/rules.d/99-uuu.rules"
2: update udev rule
        sudo udevadm control --reload-rules
andli@lenovo:~/uuu_imx8mq_com_4.14.78$
```

Copyright 2021 © Embedded Artists AB

5.3 From within u-boot

An alternative to the manufacturing tool is to use the u-boot bootloader. This bootloader is usually already programmed onto the board when delivered from Embedded Artists.

U-boot supports loading files from many sources (network, SD card, USB memory stick), but this section will only describe how to load from a USB memory stick. Copy your files to a USB memory stick and then insert this into the USB Host Connector on the Carrier Board, see Figure 7.

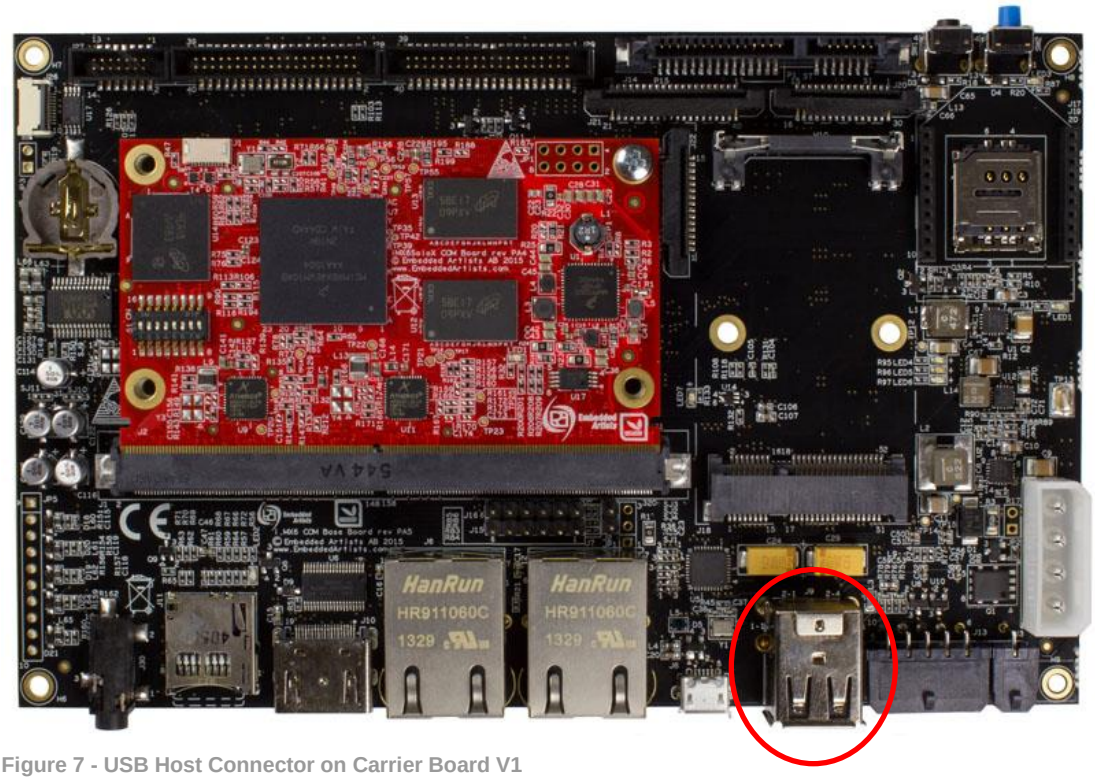


Figure 7 - USB Host Connector on Carrier Board V1

5.3.1 Find the USB Memory Stick

Follow the steps described in the *Getting Started* document for the board you are using to setup the board. You need to have a terminal/console application such as Tera Term connected to the board. When the board boots stop u-boot from continuing with the boot (by default you have 3 seconds to hit any key so the autoboot stops).

1. Enable the USB interface

```
=> usb start
```

2. Find attached storage devices. In the example below only one device is attached and will therefore get device index zero (‘0’). You need to know this index when accessing the device in later steps.

```
=> usb storage
  Device 0: Vendor: Kingston Rev: 1.00 Prod: DataTraveler R
            Type: Removable Hard Disk
            Capacity: 959.0 MB = 0.9 GB (1964032 x 512)
```

3. List the content on the USB device to see that your files are available on the device. The zero in the instructions below is the device index.

Copyright 2021 © Embedded Artists AB

```
=> fatls usb 0
```

5.3.2 Load the Root File System

To replace only the root file system, use the ext3 image, that is the file with extension ext3. You must have followed the instructions in section 5.3.1 to make sure the USB interface is enabled.

Important: Files are loaded into RAM memory which means that they must be smaller than the memory on the board.

1. Load the ext3 image into memory. In the example below USB device 0 is accessed and the file `rootfs.ext3` is loaded to address 0x83000000. The size of the loaded file is **79691776** bytes. This information is **important** and needed in step 3. The size may be different in your case.

```
=> fatload usb 0 0x83000000 rootfs.ext3
reading rootfs.ext3
79691776 bytes read in 3466 ms (21.9 MiB/s)
```

2. Find the offset to the file system partition. If you are using the default setup from Embedded Artists the root file system will be in partition 2 which as shown in the example below is available at offset 24576 (0x6000). Partition 1 contains the Linux kernel and device tree (dtb) files.

```
=> mmc part

Partition Map for MMC device 1  --   Partition Type: DOS

Part    Start Sector    Num Sectors     UUID            Type
  1     8192            16384           00027f23-01     0c
  2     24576           155648          00027f23-02     83
```

3. Write the file to eMMC. The eMMC memory is available on mmc device 1 and as found in the previous step partition 2 starts at offset 0x6000. The amount of data to write to the mmc device must be given in number of blocks where each block is 512 bytes. The value must be written as a hexadecimal value. 79691776 / 512 = 155648 ➔ 0x26000.

```
=> mmc write 0x83000000 0x6000 0x26000
```

4. A new root file system has now been written to the device and the board can be rebooted.

5.4 From within Linux

It is also possible to do updates to the system from within Linux.

5.4.1 Kernel image and dtb files

Kernel images and dtb (device tree) files are available on a partition of eMMC that is normally not mounted. The instructions below show how to update the kernel image and dtb file (or add new dtb files) from a USB memory stick to the eMMC partition.

1. Copy the files to a USB memory stick
2. Boot into Linux and then insert this into the USB Host Connector on the Carrier Board, see Figure 7. You will see output in the console as below.

Copyright 2021 © Embedded Artists AB

```
new high-speed USB device number 3 using ci_hdrc
usb-storage 1-1.3:1.0: USB Mass Storage device detected

scsi0 : usb-storage 1-1.3:1.0
scsi 0:0:0:0: Direct-Access     SanDisk  Cruzer           8.02 PQ:
0 ANSI: 0 CCS
scsi 0:0:0:1: CD-ROM            SanDisk  Cruzer           8.02 PQ:
0 ANSI: 0
sd 0:0:0:0: [sda] Attached SCSI removable disk
sd 0:0:0:0: [sda] 15704063 512-byte logical blocks: (8.04 GB/7.48
GiB)
sd 0:0:0:0: [sda] No Caching mode page found
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] No Caching mode page found
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
```

3. Mount the USB memory stick and eMMC partition. The USB memory stick has in this example the device name **sda1** as can be seen in the output in step 2 above. The partition on the eMMC is in this example available at **/dev/mmcblk2p1**, but this could be different on different COM boards.

```
# mkdir /mnt/usb
# mount /dev/sda1 /mnt/usb
# mkdir /mnt/mmcboot
# mount /dev/mmcblk2p1 /mnt/mmcboot
```

4. Copy the bin file and / or dtb file from the USB memory stick to the boot partition.

```
# cp /mnt/usb/zImage /mnt/mmcboot
# cp /mnt/usb/imx6sxea-com-kit.dtb /mnt/mmcboot
```

5. Un-mount devices

```
# umount /mnt/usb
# umount /mnt/mmcboot
```

Copyright 2021 © Embedded Artists AB

6 Extend Image with Additional Functionality

There are several ways to enable and add more functionality to an image than what's included by the image recipe. The functionality is enabled by modifying the `build_dir/conf/local.conf` file.

6.1 Image Features

Several predefined packages can be enabled by using the `EXTRA_IMAGE_FEATURES` variable in the `local.conf` file. More information about this variable and the features that are available can be found in the Image Features section in the Yocto reference manual.

As an example, the OpenSSH SSH server can be installed by adding `ssh-server-openssh` to the variable.

```
EXTRA_IMAGE_FEATURES = "debug-tweaks ssh-server-openssh"
```

6.2 Additional Packages

The Yocto project includes a lot of recipes for different packages and utilities. Some of them are included in the recipe for the image you are building, but more can be installed into the root file system by adding them to the `IMAGE_INSTALL_append` variable in `local.conf`.

Get a list of available packages in your Yocto setup by running bitbake as below.

```
$ bitbake -s > all_recipes.txt
```

In the example below `e2fsprogs` (file system utilities) and `parted` (manipulates partition tables) has been added to the variable. Please **note** that `IMAGE_INSTALL_append` must start with a space character as in the example below.

```
IMAGE_INSTALL_append = " e2fsprogs parted"
```

Copyright 2021 © Embedded Artists AB

7 Lubuntu Virtual Machine Setup

This chapter describes the steps needed to create a virtual machine and install Lubuntu on it.

7.1 VMware Workstation Player

The virtual machine is run within the VMware Workstation Player. Download and install VMware Player.

Go to www.vmware.com and select Downloads → Free Product Downloads → VMware Workstation Player. Please note that the download path may change. The path described above was valid when writing this document. If you cannot find the player please search on www.vmware.com.

7.2 Download Installation Media

This guide is for Lubuntu 64 bit. The reason for using a 64-bit version is that it is a requirement when compiling Android. If you don't plan to build Android images then the 32-bit version will work just as well.

> **NOTE**. These instructions have been tested with Lubuntu 14.04 and 16.04, but they should work on newer versions.

The 64-bit ISO image can be downloaded from https://lubuntu.me/downloads/

7.3 Creating the VMware Virtual Machine

Start the VMware Player application and select to create a new virtual machine.

Copyright 2021 © Embedded Artists AB

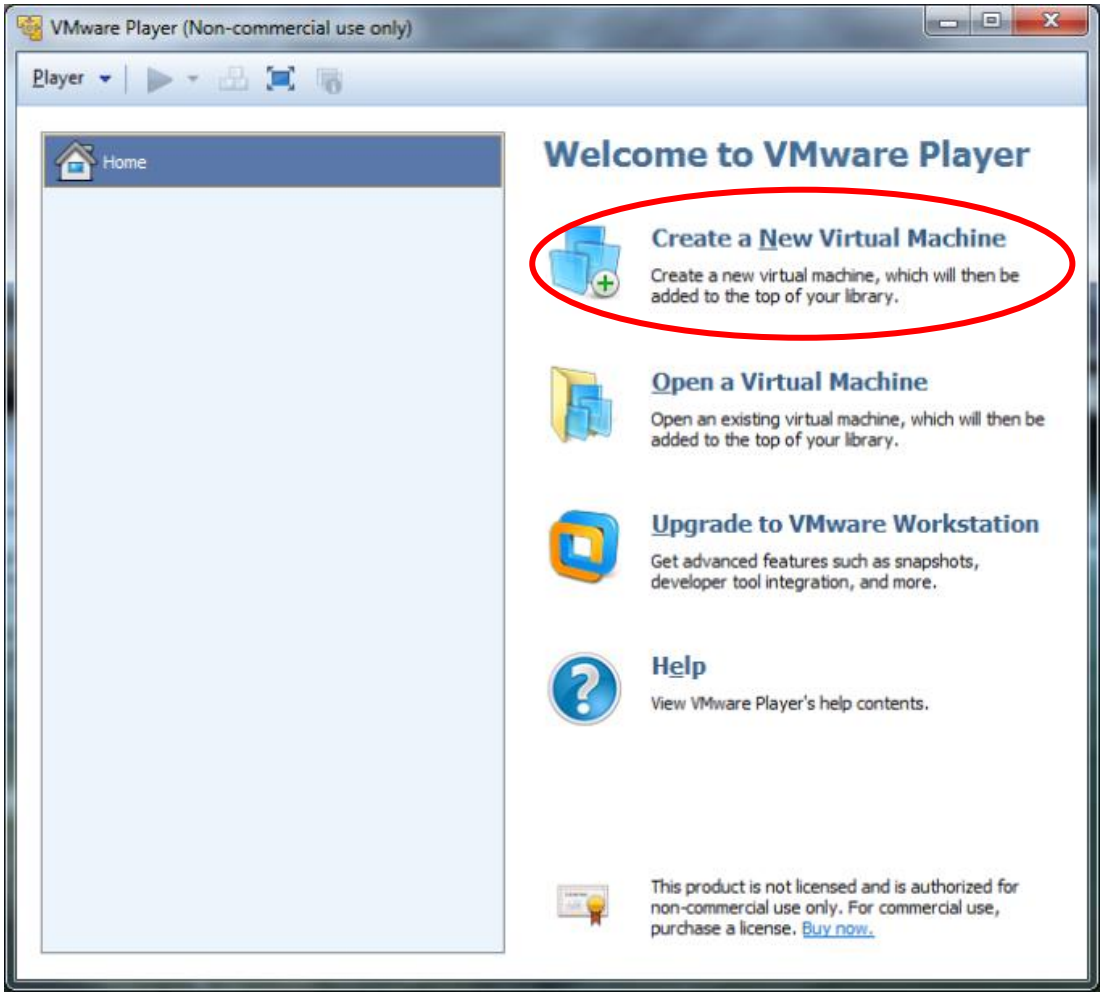


Figure 8 – VMware Player

Follow the instructions in the dialog that appears:

Copyright 2021 © Embedded Artists AB

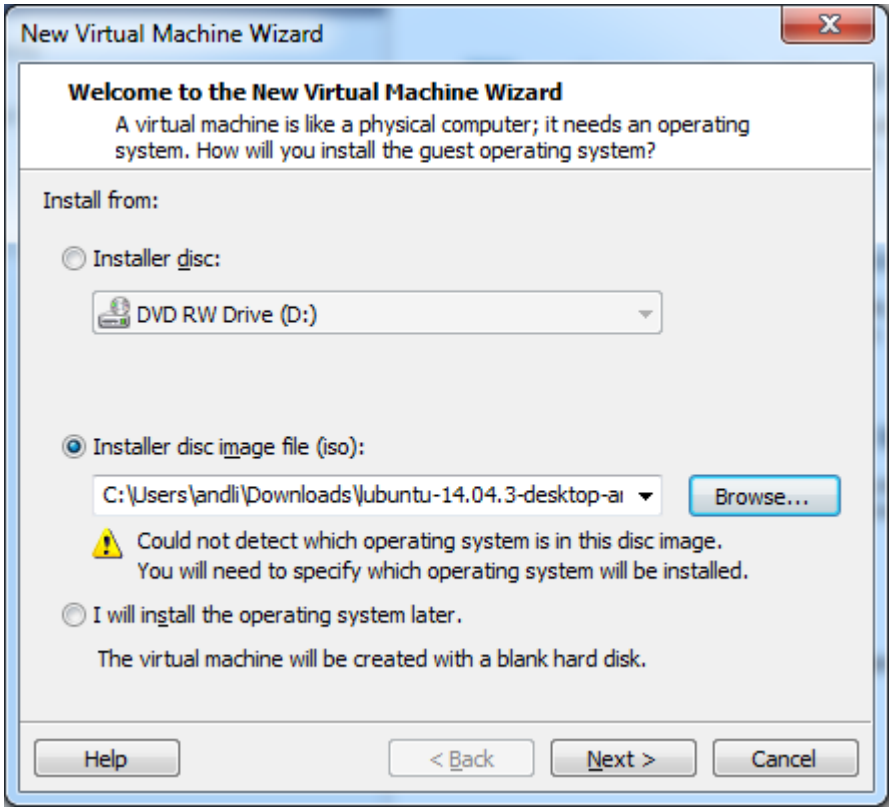


Figure 9 – New Virtual Machine – Select Installation Media

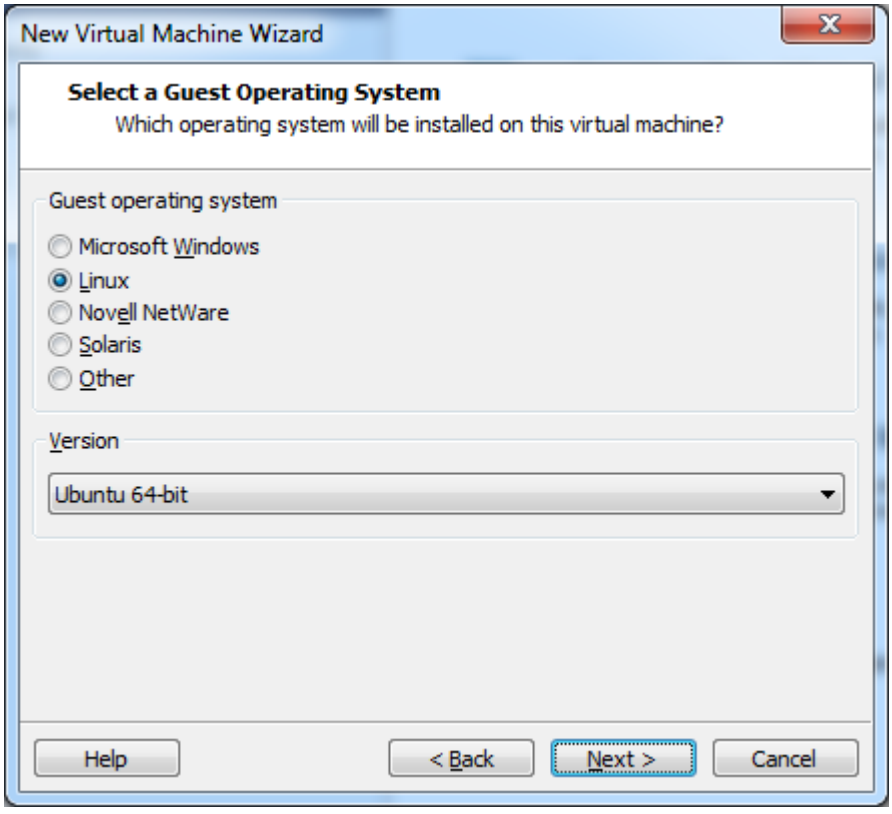


Figure 10 – New Virtual Machine – Select OS Type

Copyright 2021 © Embedded Artists AB

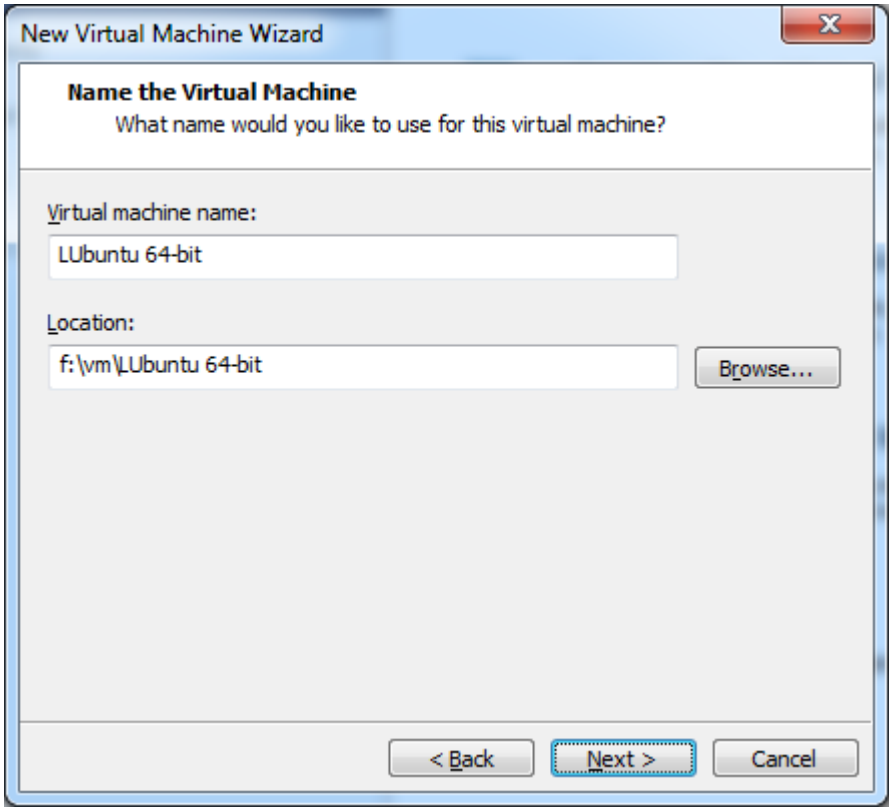


Figure 11 – New Virtual Machine – Select Name and Location

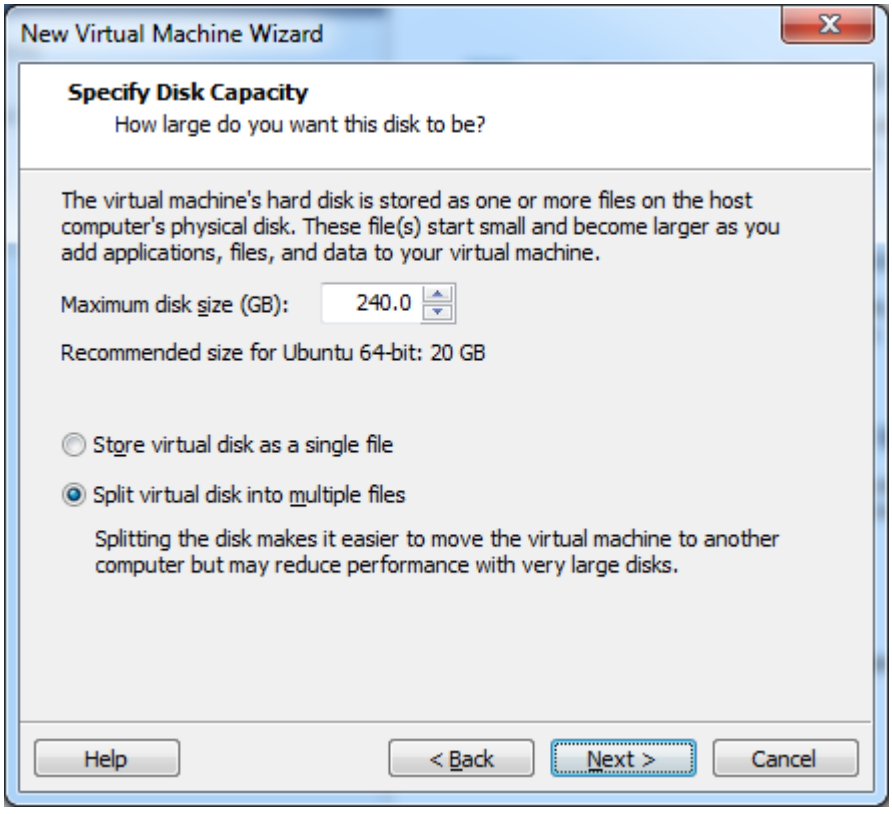


Figure 12 – New Virtual Machine – Select Disc Capcity

The default disc capacity is 20GB which is much too low. It is recommended to have at least 120GB to be able to build. If you plan to build Android then at least 200GB is needed.

Copyright 2021 © Embedded Artists AB

In the summary dialog, select to customize the hardware.

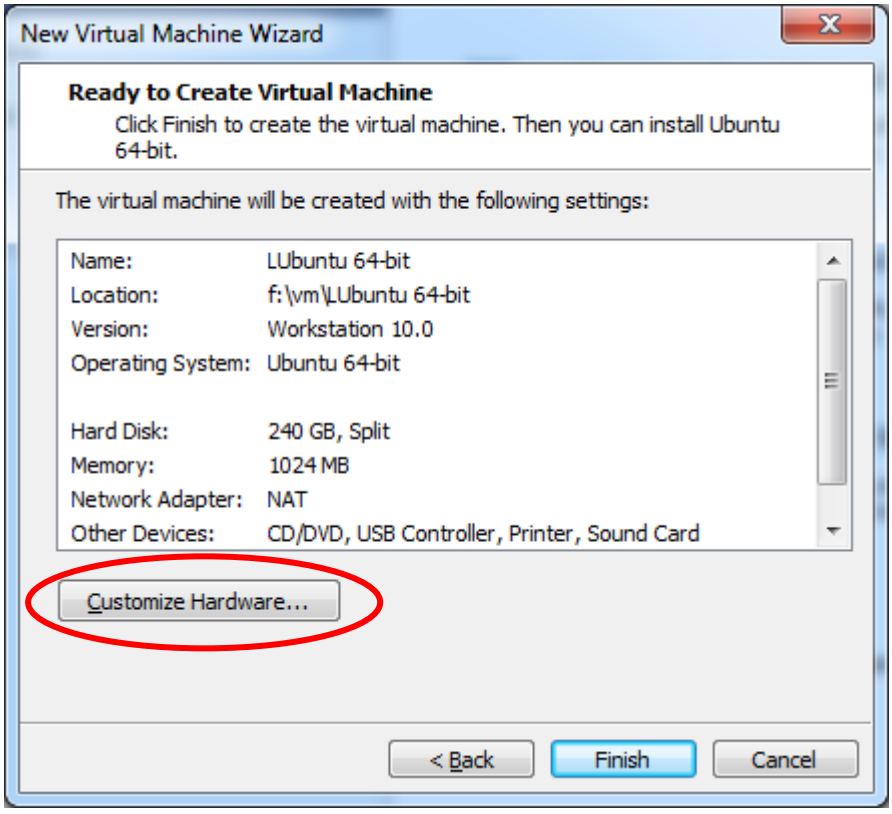


Figure 13 – New Virtual Machine – Summary

Copyright 2021 © Embedded Artists AB

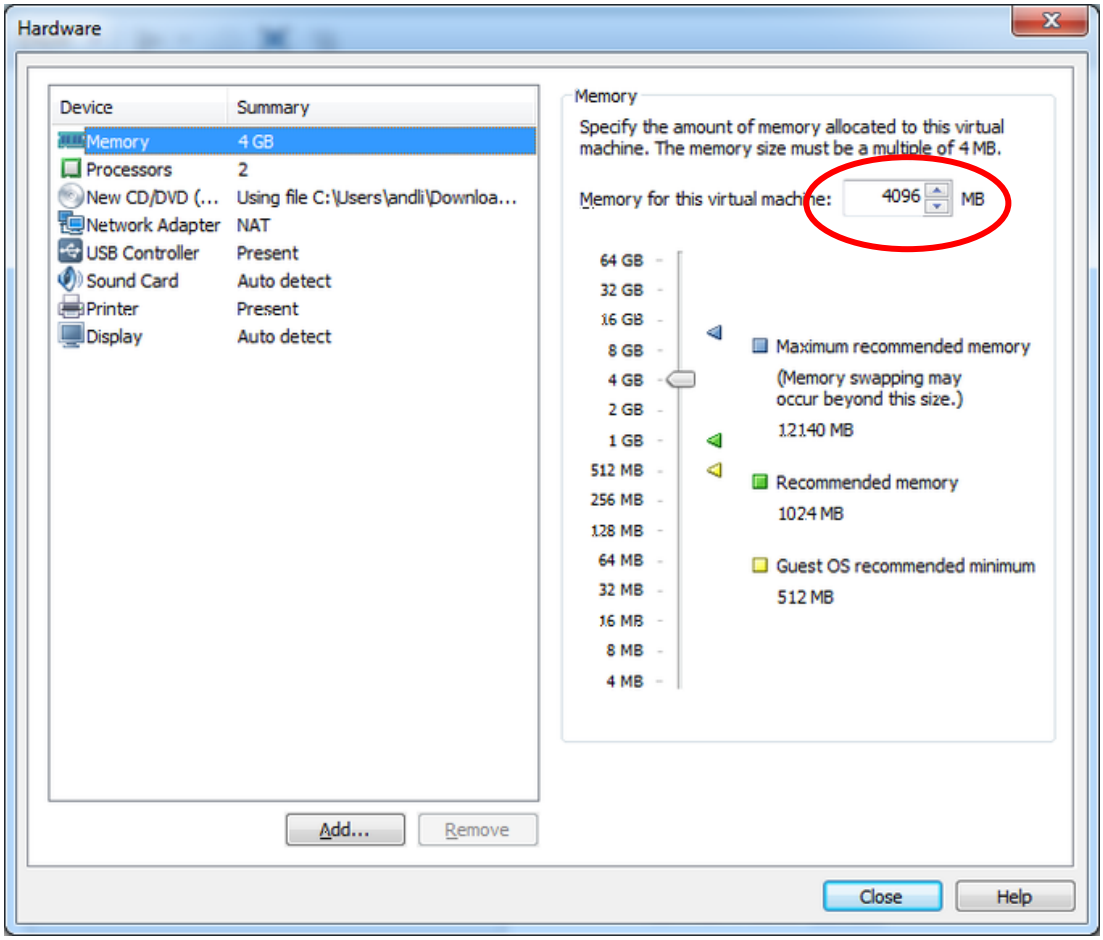


Figure 14 – New Virtual Machine – Memory Configuration

The virtual machine requires at least 3GB of memory but more is better if the host machine (the machine running the VMware Player application) can spare it.

Copyright 2021 © Embedded Artists AB

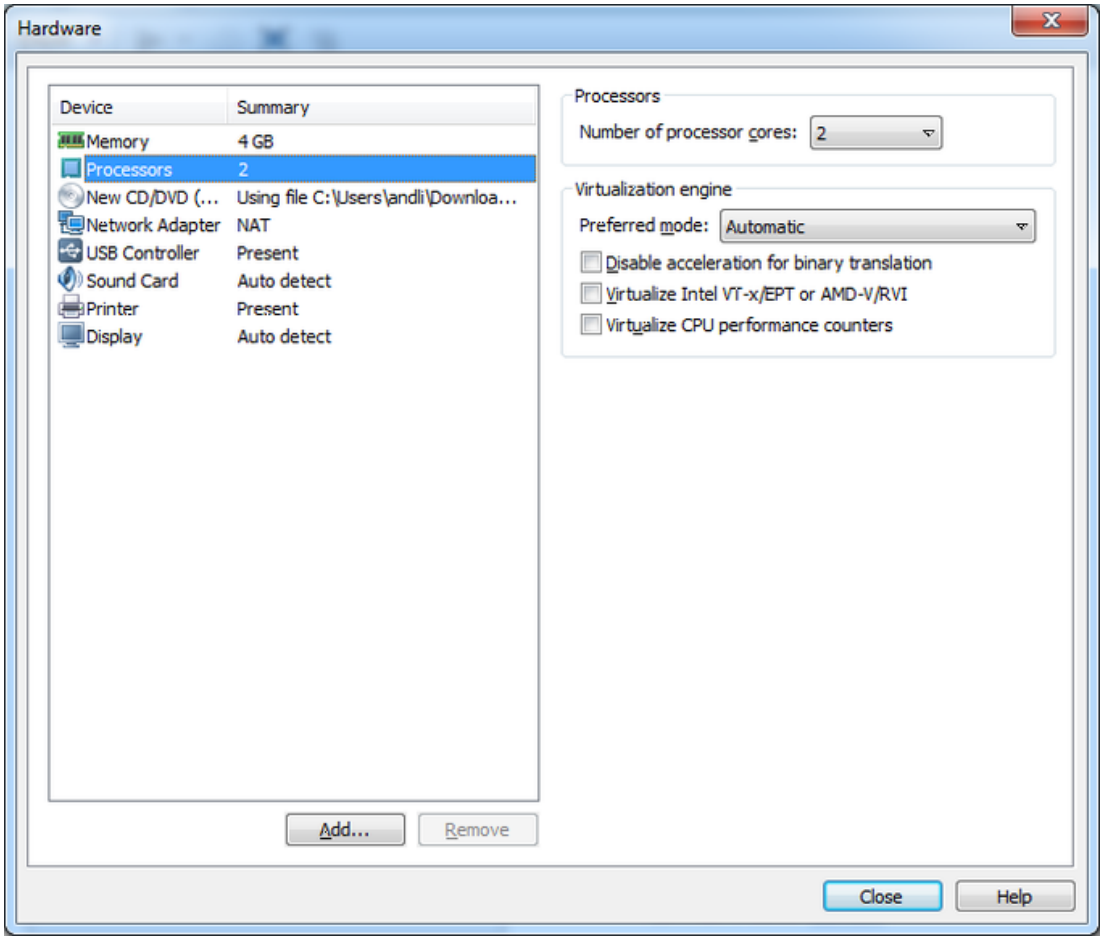


Figure 15 – New Virtual Machine – Processors

If the host machine has a processor with multiple cores then it is possible to assign more than one of those to the virtual machine which will decrease build times.

Copyright 2021 © Embedded Artists AB

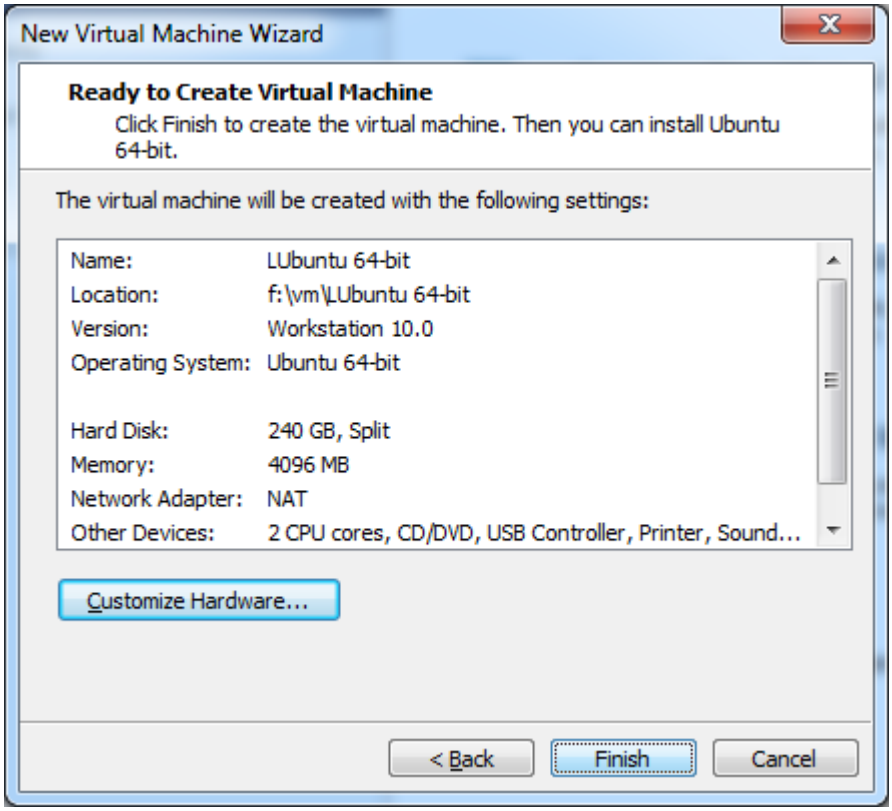


Figure 16 – New Virtual Machine – Finished Configuration

Copyright 2021 © Embedded Artists AB

The new virtual machine is now ready to be started. Press the Play button to boot the machine for the first time.

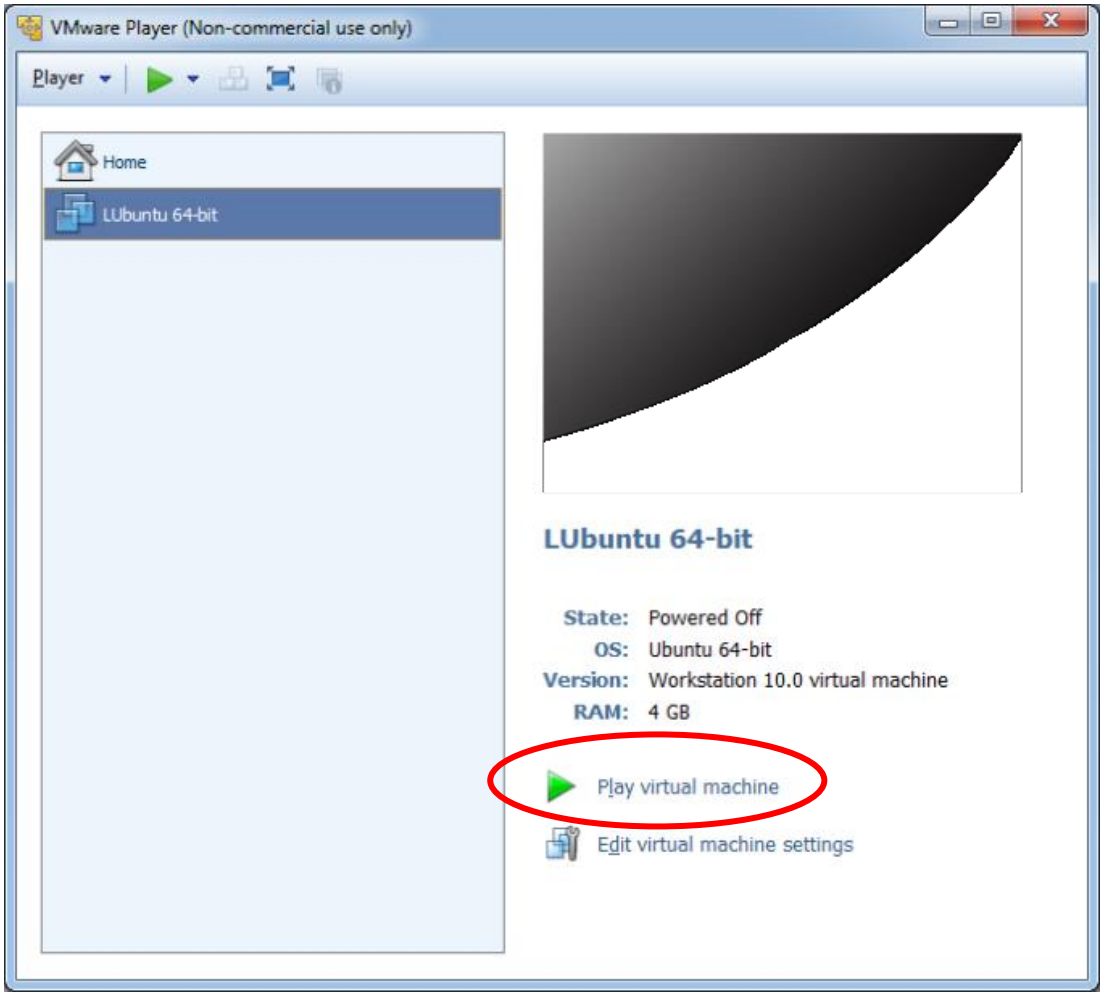


Figure 17 – New Virtual Machine – Start

Copyright 2021 © Embedded Artists AB

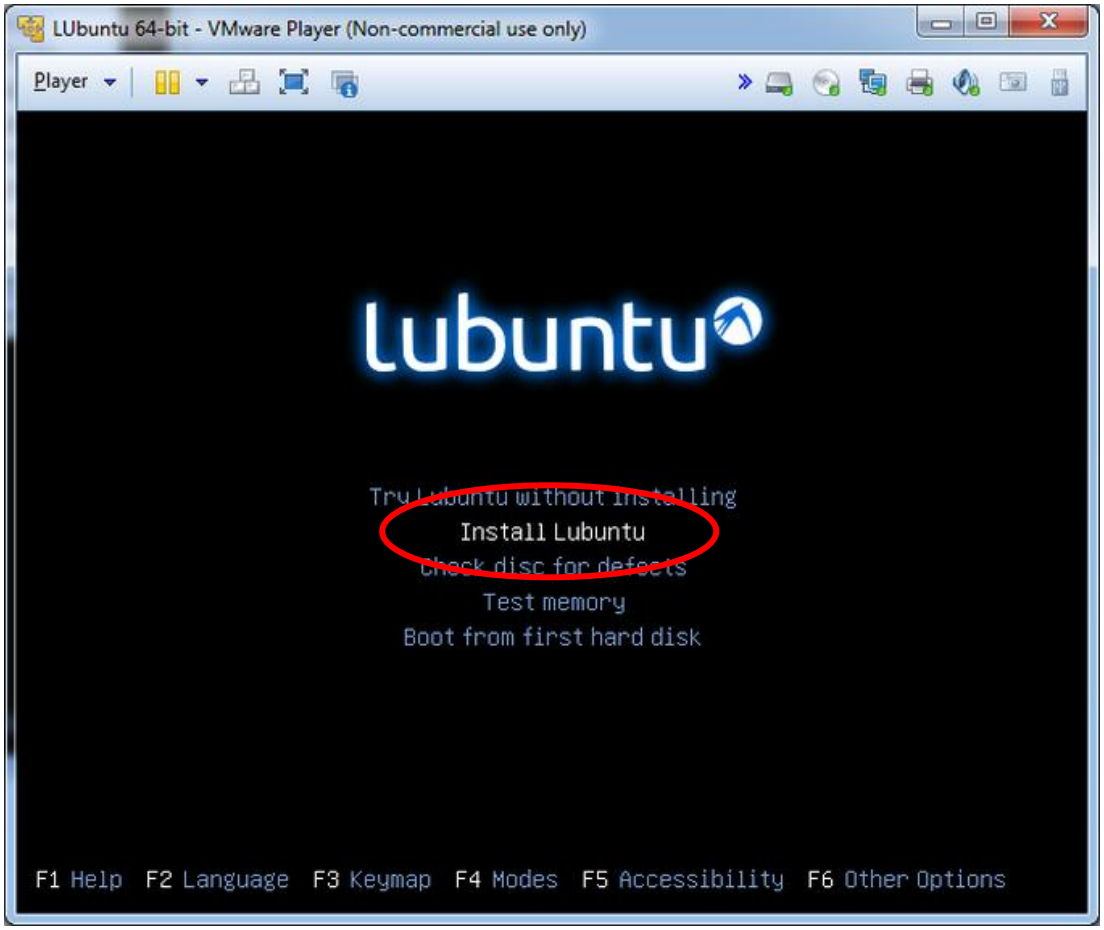


Figure 18 – Installing - Boot Option

Select to install Lubuntu and then follow the on-screen configuration guide.

Copyright 2021 © Embedded Artists AB

The first important page is the installation type page where it is a good idea to click the “Use LVM” option as it will make it easier to expand the disc space in the future. It is optional.

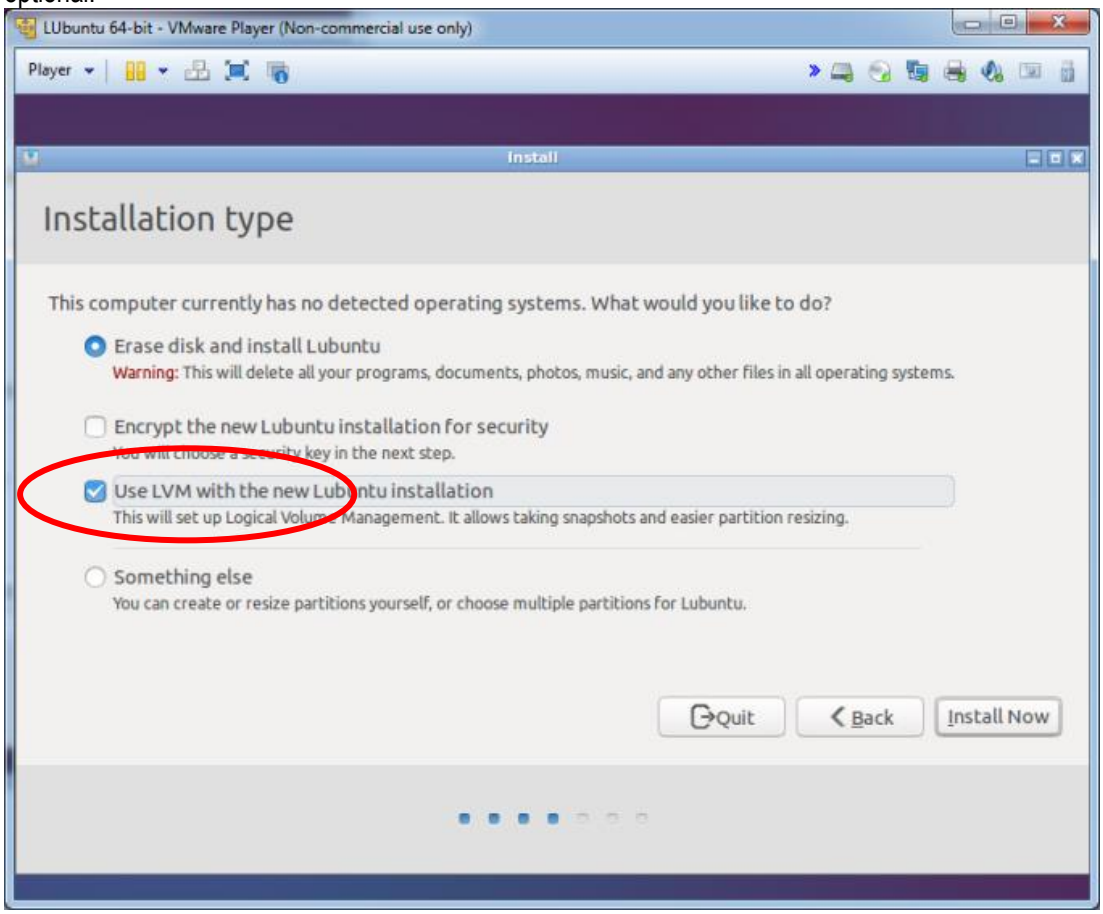


Figure 19 – Installing - Logical Volume Management

Copyright 2021 © Embedded Artists AB

The installation will complete and at the end a restart is required. When rebooting there will be a message stating to remove the installation media:

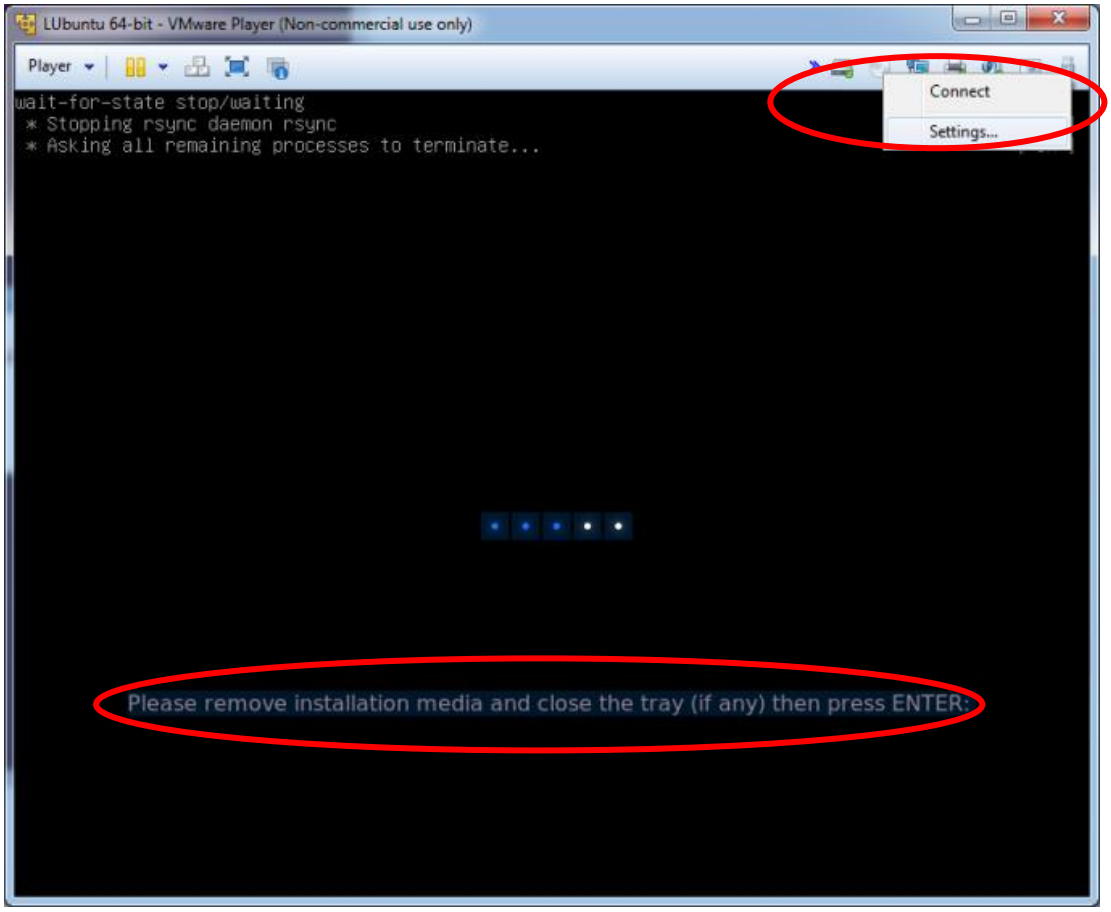


Figure 20 – Installing - Remove Media

To remove the media, right click on the CD icon at the top right of the window and select Settings. Make sure that the “Connect at power on” option is not selected. This is the same as ejecting a CD.

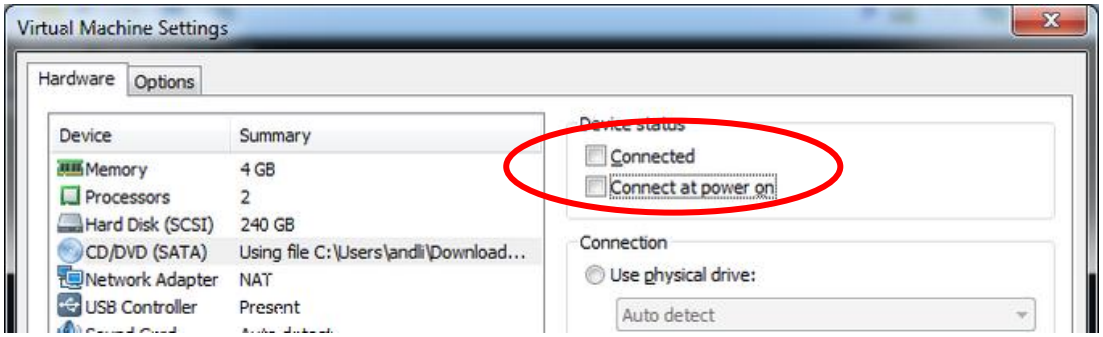


Figure 21 – Installing - Disconnect CD

Copyright 2021 © Embedded Artists AB

After removing the CD and the booting has completed the virtual machine should boot into the desktop (possibly asking for login information depending on the options selected during installation).

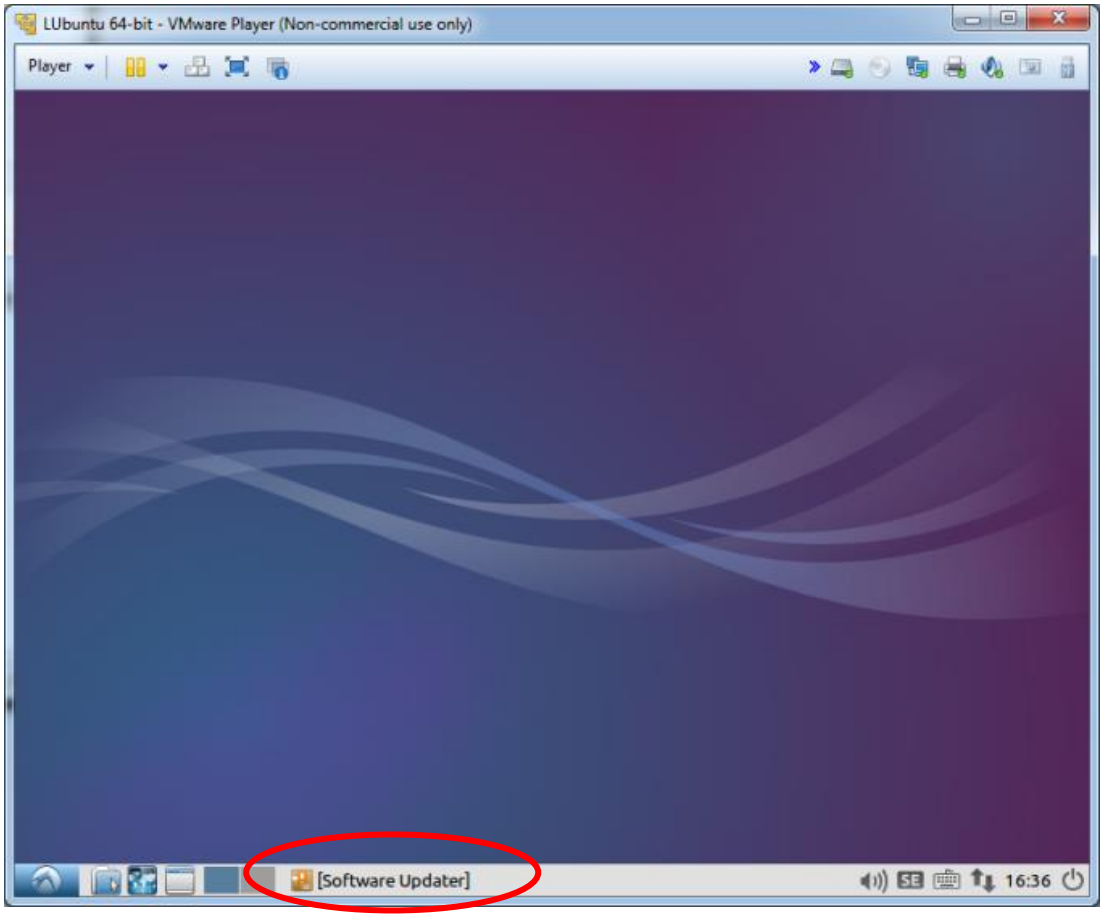


Figure 22 – Desktop

It is recommended to run the Software Updater to get all the fixes that have been made. Click the icon on the bottom of the screen and follow the instructions.

Copyright 2021 © Embedded Artists AB

The last thing to do is to install VMware Tools. This is optional but it adds a couple of useful features including

- Move the mouse pointer out of the VMware Player window without having to press Ctrl+Alt
- Copy text between the virtual machine and the host machine

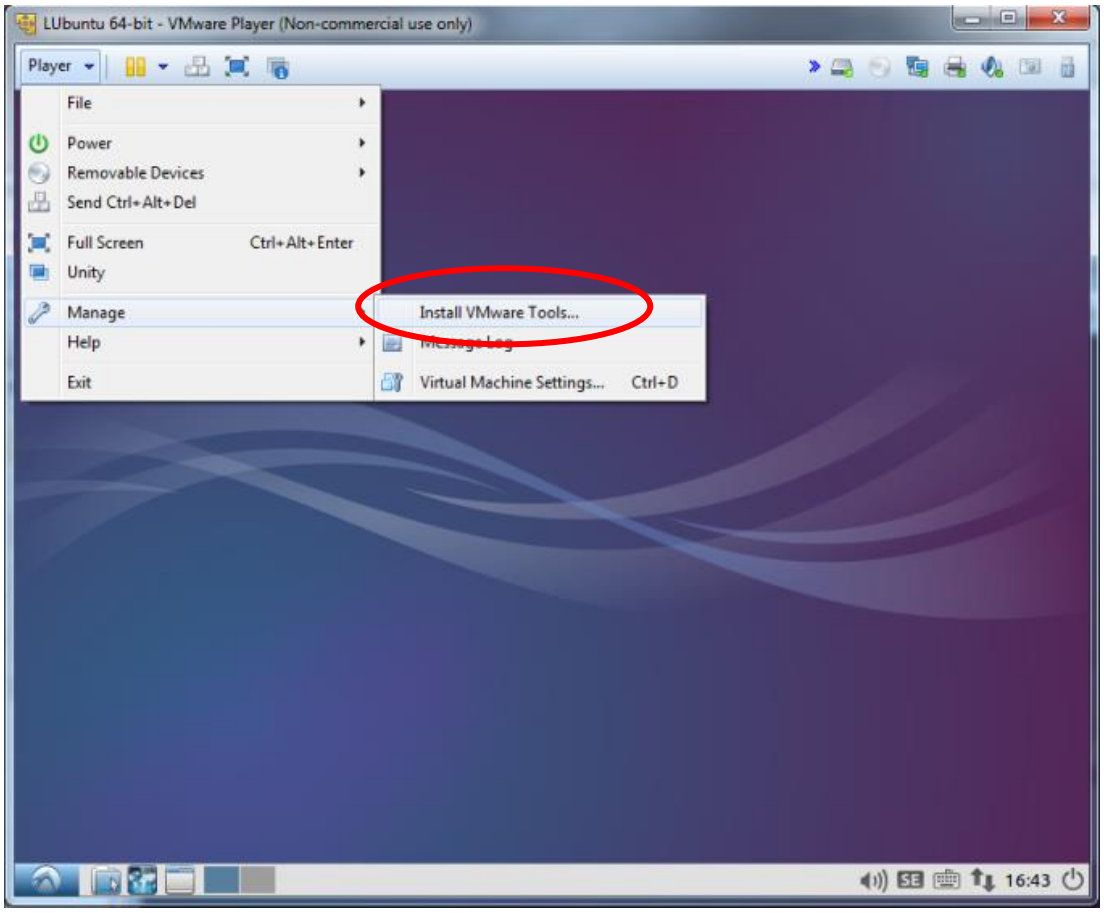


Figure 23 – VMware Tools – Start Installation

Copyright 2021 © Embedded Artists AB

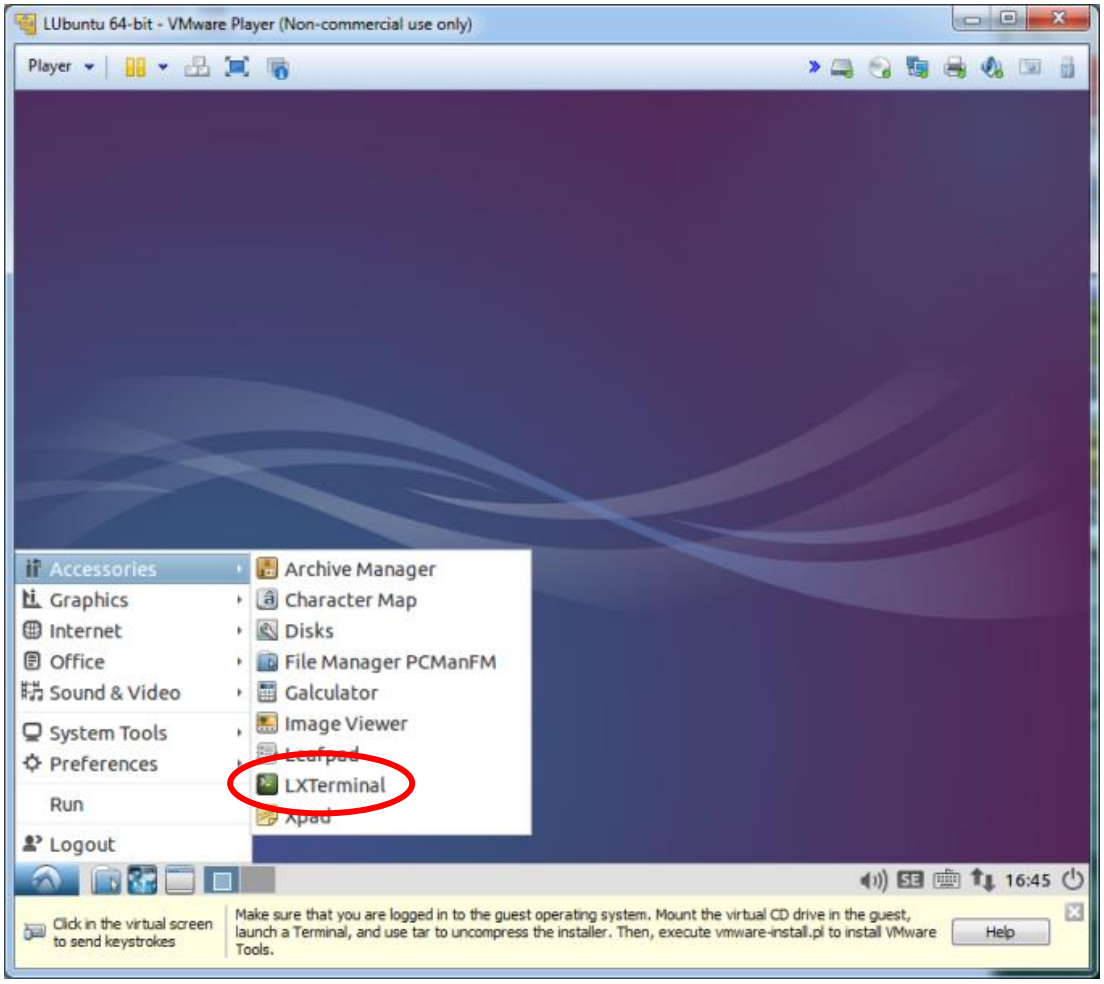


Figure 24 – VMware Tools – Start LXTerminal

Start the LXTerminal and run the following commands in the terminal window that appear:

```
$ cd /tmp/
$ tar -xf /media/user/VMware\ Tools/VMwareTools*.tar.gz
$ cd vmware-tools-distrib
$ sudo ./wmware-install.pl
```

Note that the path above assumes that the login name that you have is “user”. Replace that part with the name used to login to the virtual machine.

The wmware-intall.pl script will ask a lot of questions but always select the default answer (shown as e.g. [yes]) unless you are sure you know what you are doing.

After the last question is answered the VMware tools will be installed. Reboot when the installation program completes.

You now have your own virtual machine!

Copyright 2021 © Embedded Artists AB

8 Yocto Images

8.1 meta-toolchain

To be able to build your own application or, for example, u-boot and the Linux kernel outside of Yocto you need a toolchain. The toolchain consists of cross-compiler, linker, and necessary libraries. As mentioned in section 4.1 there is an image named **meta-toolchain** that will create the necessary toolchain.

1. Build the image

```
$ bitbake meta-toolchain
```

2. The build will result in a file located at `<build dir>/tmp/deploy/sdk`. The exact name of the file depends on several factors, but in our example, it is called:

   `fsl-imx-fb-glibc-x86_64-meta-toolchain-cortexa9hf-neon-toolchain-4.14.78-sumo.sh`

3. Install the toolchain

```
$ cd <build dir>/tmp/deploy/sdk
$ sudo ./fsl-imx-fb-glibc-x86_64-meta-toolchain-cortexa9hf-neon-toolchain-4.14.78-sumo.sh
```

4. If you select the default settings the toolchain will in this example be installed in `/opt/fsl-imx-fb/4.14-sumo/`

5. Before building an application run the command below to setup environment variables

```
$ source /opt/fsl-imx-fb/4.14-sumo/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
```

6. If you are compiling for an iMX8 target then run the following command to avoid linker error(s)

```
$ unset LDFLAGS
```

7. You can verify that the environment variables has been correctly setup by running the command below that will show the version of the GCC compiler used.

```
$ $CC --version
arm-poky-linux-gnueabi-gcc (GCC) 7.3.0
...
```

NOTE 1: Setting up environment variables in step 5 may overwrite other variables you already have in your environment. It is, for example, not recommended to do this in the same terminal where you run bitbake to build Yocto images.

Copyright 2021 © Embedded Artists AB

9 Customization

9.1 Create a layer

In Yocto a layer can be seen as a collection of related recipes (instructions) that tell the build system what to do. As an example, Embedded Artists created the layer `meta-ea` to contain all recipes needed to build an image for one of our COM boards.

For these instructions you should first have built an image as described in chapter 4 We are using the name `meta-example` in these examples. Change this to the name you want to use for your layer.

Create the layer

The instruction is run from the build directory and the layer will be created in the `sources` directory.

```
$ bitbake-layers create-layer ../sources/meta-example
```

The meta-example layer will be created with a number of files including an example recipe. The directory structure will look like below.

```
meta-example
| -- conf
|    | -- layer.conf
| -- COPYING.MIT
| -- README
| -- recipes-example
|    | -- example
|    |    | -- example_0.1.bb
```

Add the layer

The previous instruction only created the layer, but you also need to add it to the build. You can do this by either manually editing <build-dir>/conf/bblayers.bb or running the command below.

```
$ bitbake-layers add-layer ../sources/meta-example
```

The disadvantage of running `bitbake-layers` is that it will add the layer with an absolute path and not a relative path as the other layers in `bblayers.conf`. Below is an example of `bblayers.conf` were `meta-example` has been added using `bitbake-layers`.

```
...
BBLAYERS += " ${BSPDIR}/sources/meta-qt5 "
#Embedded Artists Yocto layer
BBLAYERS += " ${BSPDIR}/sources/meta-ea "
BBLAYERS += " ${BSPDIR}/sources/meta-murata-wireless "
BBLAYERS += "/home/user/builds/build-dir/sources/meta-example"
```

Version control

It is recommended that you add your new layer to a version-controlled repository such as GitHub. It is out of the scope of this document to describe how you do this, but there are many guides online such as the link below that describes how to create a new repository on GitHub.

https://help.github.com/en/github/getting-started-with-github/create-a-repo

Copyright 2021 © Embedded Artists AB

By adding the layer to a repository, you can also add it to a repo manifest that will keep track of all repositories needed to do a full Yocto image. Embedded Artists has created the repository `ea-yocto-base` that contains a repo manifest. See an example by following the link below.

https://github.com/embeddedartists/ea-yocto-base/blob/ea-4.14.98/default.xml#L34

9.2 Create a recipe

It is the recipe that contains the settings and tasks needed to build a package. The recipe contains the information about where the source code can be found, which version to use and if any patches should be applied. It can also specify dependencies to other recipes.

When a new layer was created in section 9.1 an example recipe was also created. This example only contains one .bb file which will print a message during the build.

`example_0.1.bb`

```
SUMMARY = "bitbake-layers recipe"
DESCRIPTION = "Recipe created by bitbake-layers"
LICENSE = "MIT"

python do_build() {
    bb.plain("**********************************************");
    bb.plain("*                                            *");
    bb.plain("*  Example recipe created by bitbake-layers  *");
    bb.plain("*                                            *");
    bb.plain("**********************************************");
}
```

A better example is to create a recipe that will build and install a hello world application. The instructions below are based on the example described in Yocto's official documentation.

https://www.yoctoproject.org/docs/2.5.3/mega-manual/mega-manual.html#new-recipe-single-c-file-package-hello-world

Create recipe directory

Within the meta-example layer create the directories needed for the new recipe. The first directory, `recipes-example`, is a container directory where you can put several recipes (using the same directory as was created when creating the layer). The directory `hello-world` is the actual recipe directory for this example. The sub-directory `files` will contain the source code, `helloworld.c` in this example.

```
$ cd ../sources/meta-example
$ mkdir -p recipes-example/hello-world/files
```

Create recipe file

Create a file called `hello-world.bb` within the `hello-world` directory. The content of the file should be as shown below.

```
SUMMARY = "Simple helloworld application"
SECTION = "examples"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4
f302"

SRC_URI = "file://helloworld.c"
```

Copyright 2021 © Embedded Artists AB

```
S = "${WORKDIR}"

do_compile() {
  ${CC} helloworld.c -o helloworld
}

do_install() {
  install -d ${D}${bindir}
  install -m 0755 helloworld ${D}${bindir}
}

INSANE_SKIP_${PN} = "ldflags"
```

The recipe file begins with short description of the recipe and also specifies which license that will apply.

The file continues with specifying the path to the source code (SRC_URI). It only lists the name of the file, but Yocto will know to look for this file in the sub-directory `files`.

Then comes the compile and install steps. In `do_compile` the file will be compiled into an executable program and in `do_install` the program will be installed into the file system at the location defined by `${bindir}` (typically `/usr/bin`).

The last line (`INSANE_SKIP_${PN}`) was added to skip ldflags. Without this line, QA errors were generated (like below).

```
ERROR: hello-world-1.0-r0 do_package_qa: QA Issue: No GNU_HASH in
the elf binary:
```

Create helloworld.c

Within hello-world/files create the file helloworld.c and add the content below to that file.

```
#include <stdio.h>

int main() {
    printf("Hello world\n");
    return 0;
}
```

Build the recipe

You can now try to build the recipe by running the command below from your build directory.

```
$ cd ../../<build-dir>
$ bitbake hello-world
```

To make sure the hello world application is included in the image you are building add the following to `conf/local.conf`.

```
IMAGE_INSTALL_append = " hello-world"
```

Then build the image (in this example we are using `ea-image-base`, but you can change to the image you are using).

```
$ bitbake ea-image-base
```

Copyright 2021 © Embedded Artists AB

9.3 Add content or change behavior of existing recipe

It is possible to modify an existing recipe by using a `bbappend` file. This avoids duplication of functionality and allows for modifying a recipe from another layer. One common usage of `bbappend` files is to apply patches to existing recipes without having to modify the original source.

In the example below we will modify the content of the file `asound.conf` being installed by the recipe `alsa-state`.

A file called `alsa-state.bbappend` is created in a `recipes-bsp/alsa-state`. We are keeping the same directory structure as for the original file which is located in the `poky/meta` layer. Below you can see the content of this file.

```
# Change the default configuration from 44100Hz to 48000Hz for iMX6 UltraLite

do_install_append_mx6ul () {
  if [ -e ${D}/${sysconfdir}/asound.conf ]; then
    sed -e "s:playback\.pcm.*:playback.pcm \"dmix_48000\":g" \
        -e "s:capture\.pcm.*:capture.pcm \"dsnoop_48000\":g" \
        -i ${D}/${sysconfdir}/asound.conf
  fi
}
```

What this file do is to append to the `do_install` task. When this task is run it will look for the `asound.conf` file and modify it to use 48000 Hz instead of 44100 Hz using the `sed` command. Please note that this change will only be applied for the `mx6ul` target (i.e., iMX6 UltraLite). The reason to limit the change to the iMX6 UltraLite is because for this target it wasn't possible to generate a 44100 Hz clock given the way the audio related clocks are setup for the processor.

The original file can be found on GitHub.

https://github.com/embeddedartists/meta-ea/blob/ea-4.14.98/recipes-bsp/alsa-state/alsa-state.bbappend

More information in the Yocto documentation:

https://www.yoctoproject.org/docs/2.5.3/mega-manual/mega-manual.html#using-bbappend-files

Copyright 2021 © Embedded Artists AB

10 Miscellaneous

10.1 Root file system on SD card

By default, u-boot, kernel, and root file system are stored on the onboard eMMC flash. The instructions in this section show how to put the root file system on an external SD card. The u-boot, kernel and dtb files are still stored on the eMMC.

Detect which device file the SD card is available on

1. Boot into Linux without having an SD card inserted in the SD card slot
2. Insert the SD card and you will see output similar to below in the terminal. In this example the device file to use will be `/dev/mmcblk1p1`.

```
mmc1: new SD card at address 9047
mmcblk1: mmc1:9047 SU01G 968 MiB
 mmcblk1: p1
```

Put the root file system on the SD card

1. Make sure you have downloaded and unpacked the UUU bundle (from imx.embeddedartists.com) for the board you are using.
2. Open the file `<mfgtool dir>/full_tar.uuu`
3. At the end of this file you will see a list of commands used to burn the file system to the target. The commands use the variable mmc (${mmc}) to indicate which mmc device to use. Instead of using a variable you can change to the device previously retrieved (`mmcblk1p1` in this example). See the excerpt below for how you could change the file.

```
#FBK: ucmd mmc=`cat /tmp/mmcdev`; mkfs.ext3 -F -E nodiscard -j /dev/mmcblk${mmc}p2
#FBK: ucmd mkdir -p /mnt/ext3
#FBK: ucmd mmc=`cat /tmp/mmcdev`; mount -t ext3 /dev/mmcblk${mmc}p2 /mnt/ext3

FBK: ucmd mkfs.ext3 -F -E nodiscard -j /dev/mmcblk1p2
FBK: ucmd mkdir -p /mnt/ext3
FBK: ucmd mount -t ext3 /dev/mmcblk1p2 /mnt/ext3

FBK: acmd export EXTRACT_UNSAFE_SYMLINKS=1; tar -jx -C /mnt/ext3
FBK: ucp  files/ea-image-base-imx8mmea-ucom.tar.bz2 t:-
FBK: Sync
FBK: ucmd umount /mnt/ext3
```

4. Now you can run `uuu` with full.tar to update the target. See chapter 5 for more information about deploying images.

NOTE: In this section we are modifying an existing configuration file. An alternative is to copy the configuration file and renaming it instead of modifying it.

Update u-boot variables:

1. Boot into u-boot
2. Set `mmcautodetect` to `no`. If you don't do this u-boot will detect and set the value of `mmcroot` by itself, that is, overwriting any value you set.

```
# setenv mmcautodetect no
```

Copyright 2021 © Embedded Artists AB

3. Set mmcroot to the device file for the SD card

```
# setenv mmcroot '/dev/mmcblk1p1 rootwait rw'
```

4. Save the changes

```
# saveenv
```

5. When you reset/boot the board it will use the root file system stored on the SD card.

10.2 Build Linux kernel from source code

You can build the Linux kernel outside of Yocto by following the instructions in this section. Please note that it is recommended that the kernel is built by Yocto when you are generating your final distribution images since there can be dependencies between the root file system and the kernel.

The instructions in this section assume that you have built and installed the toolchain as described in section 8.1 above.

Setup the **environment variables** for the toolchain. We are using the same installation path as described in section 8.1 above. If you have installed the toolchain in a different path use that path in the instructions below.

```
$ source /opt/fsl-imx-fb/4.14-sumo/environment-setup-cortexa9hf-
neon-poky-linux-gnueabi
```

If you are compiling Linux for an iMX8 based board then run the following command to avoid linker error(s):

```
$ unset LDFLAGS
```

Get the **source code** from the Embedded Artists GitHub repository. In this example we are checking out branch **ea_4.14.78**.

```
$ git clone https://github.com/embeddedartists/linux-imx.git
$ cd linux-imx
$ git checkout ea_4.14.78
```

Use Embedded Artists **kernel configurations** (use `ea_imx8_defconfig` instead if compiling for an **iMX8M** target).

```
$ make ea_imx_defconfig
```

(Optional) If you want to change kernel configurations you can at this point run the **menuconfig** tool.

```
$ make menuconfig
```

Build the kernel.

```
$ make
```

Copyright 2021 © Embedded Artists AB

When the build process has finished the kernel will be available here: **`arch/arm/boot/zImage`**. Device tree files are available in the following directory: `arch/arm/boot/dts`. A compiled device tree file has the file extension `dtb`.

Updating the system

You have several options to update the system with the new kernel. The first option is to use the manufacturing tool as described in section 5.1 above. The other option is to update the boot partition from within Linux as described in section 5.4.1 .

10.3 Build u-boot from source code

You can build u-boot outside of Yocto by following the instructions in this section.

The instructions in this section assume that you have built and installed the toolchain as described in section 8.1 above.

Setup the **environment variables** for the toolchain. We are using the same installation path as described in section 8.1 above. If you have installed the toolchain in a different path use that path in the instructions below.

```
$ source /opt/fsl-imx-fb/4.14-sumo/environment-setup-cortexa9hf-
neon-poky-linux-gnueabi
```

If you are compiling the u-boot for an iMX8 based board then run the following command to avoid linker error(s):

```
$ unset LDFLAGS
```

Get the **source code** from the Embedded Artists GitHub repository. In this example we are checking out branch **ea_v2018.03**.

```
$ git clone https://github.com/embeddedartists/uboot-imx.git
$ cd uboot-imx
$ git checkout ea_v2018.03
```

Use the Embedded Artists **configuration** for the COM board you are using. In the example below the configuration for the iMX6 SoloX COM board is used.

```
$ make mx6sxea-com_config
```

Build the bootloader.

```
$ make
```

When the build process has finished the u-boot image will be available directly in the `uboot-imx` directory.

Updating the system

Use the manufacturing tool as described in section 5.1 to update the system with the new u-boot image.

Copyright 2021 © Embedded Artists AB

10.3.1 Extra steps for iMX8

> **Note**: Instead of running these commands manually you can use `devtool` as described in section 10.4

For the iMX8 it is not enough to compile the u-boot - it must also be packaged together with these components to get something that can be flashed:

- U-boot (including SPL)
- ARM Trusted Firmware
- DDR Firmware
- HDMI Firmware

This is normally handled in yocto automatically but if you want to do it outside of yocto these are the steps:

1. Build the u-boot as explained above. These steps will assume that the folder `~/work/uboot-imx/` was used - otherwise change accordingly below
2. Download and build the ARM Trusted Firmware

```
$ cd ~/work/
$ git clone https://source.codeaurora.org/external/imx/imx-atf
$ cd imx-atf
$ git checkout imx_4.14.78_1.0.0_ga
$ source /opt/fsl-imx-wayland/4.14-sumo/environment-setup-aarch64-
poky-linux
$ unset LDFLAGS
```

For iMX8M Mini uCOM:

```
$ make PLAT=imx8mm bl31
```

For iMX8M Quad COM:

```
$ make PLAT=imx8mq bl31
```

3. Download and extract the DDR/HDMI Firmware and answer yes to the end user license agreement

```
$ cd ~/work/
$ wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-
8.0.bin
$ chmod +x firmware-imx-8.0.bin
$ ./firmware-imx-8.0.bin
```

4. Download the tool to create the final image

Copyright 2021 © Embedded Artists AB

```
$ cd ~/work/
$ git clone https://source.codeaurora.org/external/imx/imx-mkimage
$ cd imx-mkimage
$ git checkout imx_4.14.78_1.0.0_ga
```

5. Gather all common files

```
$ cp ../firmware-imx-8.0/firmware/ddr/synopsys/lpddr4_pmu_* iMX8M/
$ cp ../firmware-imx-8.0/firmware/hdmi/cadence/signed_* iMX8M/
$ cp ../uboot-imx/u-boot.bin iMX8M/
$ cp ../uboot-imx/u-boot-nodtb.bin iMX8M/
$ cp ../uboot-imx/spl/u-boot-spl.bin iMX8M/
$ cp ../uboot-imx/tools/mkimage iMX8M/mkimage_uboot
```

6. Gather all target specific files

   For iMX8M Mini uCOM:

```
$ cp ../imx-atf/build/imx8mm/release/bl31.bin iMX8M/
$ cp ../uboot-imx/arch/arm/dts/fsl-imx8mm-ea-ucom-kit_v2.dtb
iMX8M/fsl-imx8mm-evk.dtb
```

   For iMX8M Quad COM:

```
$ cp ../imx-atf/build/imx8mq/release/bl31.bin iMX8M/
$ cp ../uboot-imx/arch/arm/dts/fsl-imx8mq-ea-com-kit_v2.dtb
iMX8M/fsl-imx8mq-evk.dtb
```

7. Build to get `iMX8M/flash.bin` and then rename it to match the file name used in UUU.

   For iMX8M Mini uCOM:

```
$ make SOC=iMX8MM flash_evk
$ mv iMX8M/flash.bin iMX8M/imx-boot-imx8mmea-ucom-sd.bin
```

   For iMX8M Quad COM:

```
$ make SOC=iMX8MQ flash_evk
$ mv iMX8M/flash.bin iMX8M/imx-boot-imx8mqea-com-sd.bin
```

8. Use the UUU tool as described in section 5.2 to update the system with the bootloader

10.4 Use devtool to build Linux / u-boot

Section 10.2 and section 10.3 describe how to build Linux and u-boot from source code outside of Yocto. This is in general a fast way to work with Linux / u-boot development compared to re-building inside of Yocto. There is however another alternative that will let you build within Yocto, but with your own Linux / u-boot repositories. This can be suitable when you for example want to test modifications, but make sure to keep the kernel in sync with the root file system. For u-boot and especially u-boot for iMX8 the benefits of using `devtool` is that you don't have to manually re-create the boot image as described in section 10.3.1 .

Run the commands below within your Yocto build directory.

For Linux kernel:

Copyright 2021 © Embedded Artists AB

```
$ devtool modify linux-ea <path to Linux repository>
```

For u-boot:

```
$ devtool modify u-boot-ea <path to u-boot repository>
```

Once you have modified the repository you can build a new Yocto image, for example, ea-image-base.

```
$ bitbake ea-image-base
```

If you want to switch back to using the default repositories you can use `devtool reset` as shown below.

```
$ devtool reset linux-ea
$ devtool reset u-boot-ea
```

10.5 State and download cache in Yocto

If you regularly build Yocto images you can significantly reduce build times by using state and download caches. The download cache will contain downloaded packages so they don't have to be downloaded multiple times for new build directories. The state cache will contain built packages so you these don't have to be re-built between build directories given that the state hasn't changed.

You can enable the caches by adding the lines below to your `local.conf` file (`<build dir>/conf/local.conf`).

Download cache (change the directory name below to a directory on your computer):

```
DL_DIR="~/downloads"
```

State cache (change the directory name below to a directory on your computer):

```
SSTATE_DIR="~/shared-sstate-cache"
```

You could also create a file called `site.conf` that you copy to your conf directory each time you create a new build directory. Below is an example of how such a file could look like.

```
SSTATE_DIR = "/opt/yocto-cache/shared-sstate-cache"
DL_DIR = "/opt/yocto-cache/downloads"
SSTATE_MIRRORS = " file://.* file:///opt/yocto-cache/shared-sstate-
cachePATH "
```

Copyright 2021 © Embedded Artists AB

11 Frequently Asked Questions

11.1 I want to add package XYZ – how do I do this?

Add the package to `IMAGE_INSTALL_append` as described in section 6.2 above. You can run `bitbake -s` to get a list of all available recipes.

11.2 Which packages are included in my build?

Use the command below to get a list of packages that are included. In this example we are also running the output through sort to get a sorted list of packages.

```
$ oe-pkgdata-util list-pkgs | sort > included_packages.txt
```

11.3 Which recipe generated a specific package?

If you want to know which recipe generated a specific package you can use the command below. Replace <package> with the package you want to look for. It is quite common that the recipe name is the same as the package name.

```
$ oe-pkgdata-util lookup-recipe <package>
```

11.4 Which recipe generated a specific file on the file system?

If you have found a file on the file system that you would like to modify you can run the command below to find which recipe create / installed that file. In this example we look for the recipe that created `/etc/asound.conf`.

```
$ oe-pkgdata-util find-path /etc/asound.conf
alsa-state: /etc/asound.conf
```

Here we can see that the recipe is called `alsa-state`. You can now search in the `sources` directory for this recipe.

```
$ find . -name *alsa-state*.bb
./poky/meta/recipes-bsp/alsa-state/alsa-state.bb
```

If you would like to modify this file you could then create a `bbappend` file an put that in your own layer as described in section 9.3 .

11.5 How do I add my own files to the file system?

If you want to install your own files to the file system you could either add your own recipe that installs the files (see section 9.2 for a description of how to add a recipe) or you can use the Embedded Artists recipe called `ea-files`.

Below you can see an example where we use `ea-files` to add two files to the file system. The first part (`EA_FILES_DIRS`) lists the directories where the files need to be installed. The second part installs the file `crank_8mm_mn.tar.gz` to `/home/root` and `crank_8mm_mn.sh` to `/etc/profile.d`.

These files will be given the permissions 644 (read for all, write only for user).

Copyright 2021 © Embedded Artists AB

```
EA_FILES_DIRS = "\
 /home/root \
 /etc/profile.d \
"

EA_FILES_644 = "\
 crank_8mm_mn.tar.gz:/home/root/crank_8mm_mn.tar.gz \
 crank_8mm_mn.sh:/etc/profile.d/crank_8mm_mn.sh \
"
```

See the `ea-files.bb` file for more information.

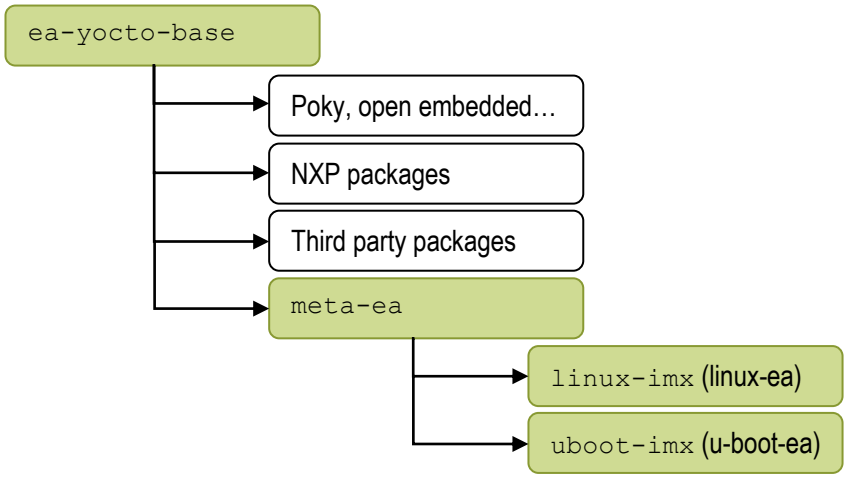
https://github.com/embeddedartists/meta-ea/blob/ea-4.14.98/recipes-ea/ea-files/ea-files.bb#L32

11.6 How do I install my own application to the file system?

You can either follow the instructions in section 11.5 if you have an already compiled application or you can follow the instructions in section 9.2 if you want to create your own recipe that compiles and installs the application.

11.7 Where are the repositories?

Below you can see an overview of how repositories are organized. It all begins with the manifest located in the `ea-yocto-base` repository.



11.7.1 meta-ea

Section 3.4 describes how to download the repositories included in the Yocto build. In that section you can see that a tool called `repo` is used together with a link to a repository called `ea-yocto-base`. This repository contains a manifest (`default.xml`) that tells `repo` which repositories to download, where they are located, and exactly which revision to use. Follow the link below to see how the manifest looks like for the ea-4.14.98 distribution.

https://github.com/embeddedartists/ea-yocto-base/blob/ea-4.14.98/default.xml

Below is an excerpt from the manifest file. Here you can see that the Embedded Artists repositories are located on GitGub and you can also see which revision of the layer `meta-ea` that will be downloaded.

Copyright 2021 © Embedded Artists AB

```
...
<remote fetch="git://github.com/embeddedartists" name="EA"/>
...

<project remote="EA" name="meta-ea" path="sources/meta-ea"
revision="841430dceea43fc3a82683f2a520e4c6cb9775ae">
     <copyfile src="ea-setup-release.sh" dest="ea-setup-
release.sh"/>
</project>
...
```

11.7.2 Linux kernel

The Linux kernel repository is not specified in the manifest file. Instead, the layer `meta-ea` specifies which kernel to use. Below is a direct link to the recipe that specifies which Linux kernel to use for 4.14.98 distribution.

https://github.com/embeddedartists/meta-ea/blob/841430dceea43fc3a82683f2a520e4c6cb9775ae/recipes-kernel/linux/linux-ea_4.14.98.bb

An excerpt from the recipe shows the location of the repository (`SRC_URI`), the branch to use (`SRCBRANCH`) and also the exact revision to use (`SRCREV`).

```
...
SRC_URI = "git://github.com/embeddedartists/linux-
imx.git;protocol=git;branch=${SRCBRANCH}"
...
LOCALVERSION = "-2.2.0"
SRCBRANCH = "ea_4.14.98"
SRCREV = "b095f5fb8a8de39e77f578dcd351783fdcdf1984"
```

11.7.3 U-boot bootloader

The u-boot repository is not specified in the manifest file. Instead, the layer `meta-ea` specifies which u-boot bootloader to use. Below is a direct link to the recipe for the u-boot used in the 4.14.98 distribution.

https://github.com/embeddedartists/meta-ea/blob/841430dceea43fc3a82683f2a520e4c6cb9775ae/recipes-bsp/u-boot/u-boot-ea_2018.03.bb

The location and revision of the u-boot are specified in a separate file called `u-boot-ea-common_2018.03.inc`.

https://github.com/embeddedartists/meta-ea/blob/841430dceea43fc3a82683f2a520e4c6cb9775ae/recipes-bsp/u-boot/u-boot-ea-common_2018.03.inc

An excerpt from `u-boot-ea-common_2018.03.inc`.shows the location of the repository (`SRC_URI`), the branch to use (`SRCBRANCH`) and also the exact revision to use (`SRCREV`).

```
...
SRCBRANCH = "ea_v2018.03"
SRC_URI = "git://github.com/embeddedartists/uboot-
imx.git;branch=${SRCBRANCH} \
           "
SRCREV = "f70b292988ec7bcee7f4b412e62cdd9cd05e6dba"
...
```

Copyright 2021 © Embedded Artists AB

11.8 How do I use my own Linux kernel?

It is quite likely that you need to modify the Linux kernel that we provide. This could for example be to add a new device driver or change the default kernel configuration. What is the recommended way to include these changes in a Yocto build? The answer depends on which step of the development you are at.

In the early stages you would typically clone our repository, apply your changes and then test these by using either `devtool` as described in section 10.4 or compile it outside of Yocto as described in section 10.2 above.

The recommendation is however that you in the end should **use your own repository**. The easiest way is to fork our repository on GitHub and then apply your changes to this new repository. You can still use the instructions in sections 10.3 and 10.4 to build your version of the Linux kernel.

How to handle this in your final product see our recommendations in section 11.10 below.

11.9 How do I use my own u-boot?

The recommendations are the same for the u-boot as for the Linux kernel so see section 11.8 above.

11.10 Should I use my own repo manifest and Yocto layer?

Our recommendation is that you take control over the software you include in your product. This also includes the Linux distribution you provide. To take this control you should setup your own `repo` manifest (you could base it on `ea-yocto-base`) and also your own Yocto layer (you could base it on `meta-ea`). In the Yocto layer you should then use your repositories of Linux kernel and u-boot bootloader.

11.11 How can I reduce the build time?

See section 10.5 for a description of how you can use state and / or download caches to reduce the build time.

11.12 Where is the package manager?

Many users have been using Debian / Ubuntu based distributions where they are used to installing new software using APT (`apt-get`) and want to know if there is something similar on Embedded Artists Yocto images. The short answer is; No there isn't any package manager.

The main reason is that although it is quite convenient to use a package manager during the development cycle it is not that common to use it in a final product. Adding, removing or modifying software on a final product must be handled in a controlled way making sure not to break any functionality. Instead, many products have over-the-air (OTA) / remote update functions that allows it to be updated in a controlled way by the manufacturer.

It is possible to enable package manager(s) in Yocto, but this is out-of-scope for this document. See the link below for some more information.

- https://wiki.yoctoproject.org/wiki/TipsAndTricks/EnablingAPackageFeed

11.13 Which version of Yocto am I using?

Run the command below in the directory containing the Yocto sources.

```
$ grep -E "DISTRO_VERSION|DISTRO_CODENAME" sources/poky/meta-
poky/conf/distro/poky.conf
```

Copyright 2021 © Embedded Artists AB