Intel® Media SDK for Intel Atom® Processor E3900 Series/Intel® Celeron® Processor N3350/Intel® Pentium® Processor N4200/Intel® Celeron® Processor J3355 and J3455 on Yocto Project*-based BSP for Linux* OS (MR4-B-02)

Getting Started Guide

May 2020

Revision 003

Document Number: 600664-003

Contents

Figures

Revision History

Date	Revision	Description
May 2020	003	Updated for MR4-B-02 release
October 2018	002	Updated for MR4 release.
August 2018	001	Initial Release for Kernel 4.14.

§

1.0 Introduction

Intel® Media Software Development Kit for Embedded Linux* (Intel® Media SDK) is a software development library that exposes the media acceleration capabilities of Intel® platforms for decoding, encoding, and video preprocessing.

Unlike software-only packages that can be expected to work across a wide variety of platforms and environments, Intel® Media SDK is a combination of driver, library, and graphics stack components requiring specific hardware, Linux distributions, kernel levels as described here.

This document covers the basics of installing, compiling, and validating the correct operation of Intel® Media SDK, using Intel® Media SDK sample applications.

For more information on the sample source code provided, refer to "`/opt/intel/mediasdk/doc/Media Samples Guide.pdf.`"

Another set of simplified examples can be found under the "tutorials" tab at https://software.intel.com/en-us/intel-media-server-studio-support.

This document describes the following two methods for the installation of Internet of Things (IoT) Media stack on a Yocto Project*-based Linux system. For more information on the IoT Media stack, refer to the "White paper: IoT Media Stack on Yocto Project-Based Metadata and Source Code for Intel Atom® Processor E3900 Series" at this location: http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html.

1. Manually install the IoT Media stack (recommended for first-time users): Build the Yocto-based Linux bootable image with the base Board Support Package (BSP) and manually install IoT Media stack at runtime described in Section 2.0.
2. Pre-install the IoT Media stack (recommended for advanced users): Build the Yocto-based Linux bootable image comprising the base BSP and IoT Media stack at Compile time described in Section 3.0.

The Operating System (OS) image can be built on multiple Linux distributions. However, only Ubuntu* is validated.

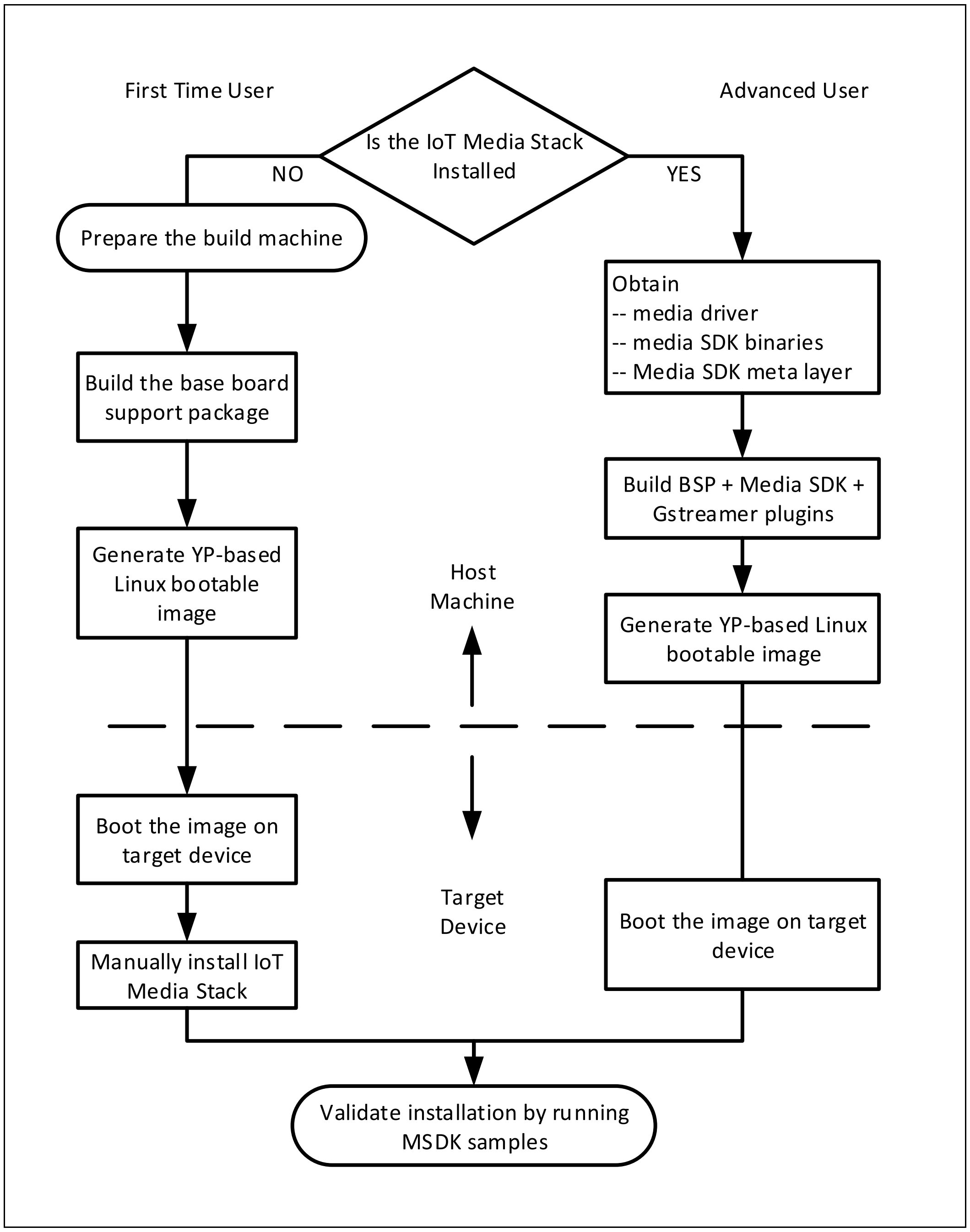
1.1 Terminology

Table 1. Terminology

Term	Description
BIOS	Basic Input/Output System
BSP	Board Support Package
CRB	Customer Reference Board
EFI	Extensible Firmware Interface
GNU GRUB *	GRand Unified Bootloader*
GST	GStreamer
HDD	Hard Disk Drive
Intel® Media SDK	Intel® Media Software Development Kit
IoT	Internet of Things
MSDK	Media Software Development Kit
OS	Operating System
SSD	Solid-state drive
USB	Universal Serial Bus

The following flowchart illustrates installation process steps:

Figure 1. Installation Process


First Time User
Is the IoT Media Stack Installed
Advanced User
NO
YES
Prepare the build machine
Obtain
-- media driver
-- media SDK binaries
-- Media SDK meta layer
Build the base board support package
Build BSP + Media SDK + Gstreamer plugins
Generate YP-based Linux bootable image
Host Machine
Generate YP-based Linux bootable image
Boot the image on target device
Target Device
Boot the image on target device
Manually install IoT Media Stack
Validate installation by running MSDK samples

NOTES:

1. Base BSP – the Yocto Project* BSP Base meta layer tarball for Intel Atom® E3900 SoC Family
2. Intel® Media Package includes iHD VA driver and LibVA binaries.
3. Intel® Media Software Development Kit (MSDK) Package – Intel® Media SDK package containing libraries, plug-ins, samples, tools and so on.

2.0 First-time User

This section is meant for **first-time users** who are completely new to the Yocto Project* and Intel® Media SDK. This section describes the step-by-step procedure to build a Yocto Project-based Linux* machine and run Intel® Media SDK on the "Intel Atom® E3900 SoC Family.

For the **advanced users** who are already familiar with Yocto Project and Intel® Media SDK, go to Section 3.0.

Note: Instructions are limited to specific platforms and packages tested for reference, but are intended to be general.

2.1 Prepare the Build Machine

The generic procedure to prepare the build machine can be found at https://github.com/intel/iotg-yocto-bsp-public/tree/e3900/master

For easy reference, we have provided the procedure used during internal testing in the following sections.

Customers are only to use the information in the following sections for reference because the configurations, settings, Package names, links, and so on, may change in the future.

2.1.1 Check the Prerequisites

- Host/Build Machine Hardware:
  - Processor: Intel® Core™ i7-4770R @3.20 GHz
  - Platform: GB M4HM87P-00
  - Memory: 16 GB DDR3 @1600 MHz
  - Disk: > 200GB (50 GB if the INHERIT variable is used to remove the unused work space)
- Operating System (OS): Ubuntu* 14.04.5 was tested with this release.
- Download the following software packages:
  - Intel® Media SDK binary tarball i.e `IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz` from https://software.intel.com/en-us/media-sdk
  - The LibVA driver (iHD driver binary) is packaged in Intel® media binary tarball). Refer to this link- http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html, to obtain the latest media binary tarball i.e: `intel-linux-media-yocto_bxt-<version>-64bit.tar.gz`

2.1.2 Configure the Build Machine

The build script assumes it is run connected to a VPN, this means some proxy server information must be set. Additional steps are added to prepare the git environment since the proxy for the git client is different from the host environment.

Note: If the proxy settings are not setup correctly, you could experience build issues. So to save time, it is recommended to avoid the proxy settings and directly use the public network.

Following are the steps to set up the proxy server and it assumes the user ID is "aplbuild":

1. Open "Network setting" and set the proxy server.
2. Click the "apply to system wise" button to apply the new settings to the whole system.
3. Set the environment variables.

```
$ sudo vi /etc/sudoers

    add: aplbuild        ALL=(ALL:ALL) ALL

$ vi ~/.bashrc

    add:
      export SOCKS_SERVER=$socks_proxy
      export HTTP_PROXY=$http_proxy
      export HTTPS_PROXY=$https_proxy
      export FTP_PROXY=$ftp_proxy
      export SOCKS_DIRECT=$HTTP_DIRECT
      export NO_PROXY=$HTTP_DIRECT
      export ALL_PROXY=$HTTP_DIRECT
      export http_direct=$HTTP_DIRECT
      export socks_direct=$SOCKS_DIRECT
      export socks_server=$SOCKS_SERVER
      export no_proxy=$NO_PROXY

$ vi /etc/apt/apt.conf
    confirm:
      Acquire::http::proxy "http://proxy.aplbuild.com:911/";
      Acquire::https::proxy "https://proxy.aplbuild.com:911/";
      Acquire::ftp::proxy "ftp://proxy.aplbuild.com:911/";
      Acquire::socks::proxy "socks://proxy.aplbuild.com:911/";
```

4. Install the build tools.

```
$ sudo apt-get update

$ 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

$ sudo apt-get install make xsltproc docbook-utils fop dblatex
xmlto

$ sudo apt-get install autoconf automake libtool libglib2.0-dev
$ sudo apt-get install xutils-dev nfs-common
```

5. Customize the Git environment.

```
$ vi ~/.gitconfig
    add:
     [user]
     mail = aplbuild.user@aplbuild.com
     name = Aplbuild User

     [sendemail]
     smtpserver = smtp.aplbuild.com
     signedoffcc = false
     suppresscc = all
     chainreplyto = false
     assume8bitEncoding = utf-8
     from = < Aplbuild User> <aplbuild.user@aplbuild.com>
     confirm = always

     [color "grep"]
     match = red

     [color]
     diff = auto
     ui = auto
     interactive = auto
     grep = always

     [alias]
     co = checkout
     br = branch
     ci = commit
     st = status
     ol = log -oneline

     [core]
     editor = gedit OR vi
     gitproxy = /home/aplbuild/bin/gitproxy
$ mkdir ~/bin
$ vi ~/bin/gitproxy
    add:
     #!/bin/bash

exec socat stdio SOCKS:proxy.aplbuild.com:$1:$2
$ chmod +x /home/aplbuild/bin/gitproxy
```

6. Register your SSH key at GitHub.

   a. Confirm a GitHub account exists.

```
$ ssh-keygen -t rsa -b 4096 -C "aplbuild.user@xyz.com"
$ eval "$(ssh-agent -s)"
$ ssh-add ~/.ssh/id_rsa
$ sudo apt-get install xclip
$ xclip -sel clip < ~/.ssh/id_rsa.pub
```

   b. Generate a local SSH key.

   c. Open the browser to website https://github.com.

   d. Login to github account, find the icon on the right upper corner.

   e. Click the icon for the pull down menu, then `Settings>SSH and GPG keys>New SSH key`.

   f. Input the name of the new key and paste the key string from xclip.

   g. Click “Add SSH Key” button.

7. Add following settings to the `.ssh/config` file:

```
$ vi .ssh/config

    add:
      host github.com
      user git
      hostname ssh.github.com
      identityfile ~/.ssh/id_rsa
      port 443

      tcpkeepalive yes
      compression yes
      connectionattempts 3
```

8. Reboot the machine.

9. In a temporary folder, perform the following to register your SSH key. Otherwise the install may be blocked by Yes/No questions during SSH key registration.

```
$ cd ~/tmp
$ git clone git@github.com:01org/iotg-yocto-bsp-public (this step
makes sure the '[ssh.github.com]:443' RSA key to the host)
Cloning into ‘iotg-yocto-bsp-public’...
The authenticity of host '[ssh.github.com]:443 (<no hostip for
proxy command>)' can't be established.
RSA key fingerprint is
16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[ssh.github.com]:443' (RSA) to the list
of known hosts.
remote: Counting objects: 1115, done.
```

```
remote: Total 1115 (delta 0), reused 0 (delta 0), pack-reused 1115
Receiving objects: 100% (1115/1115), 42.48 MiB | 3.57 MiB/s, done.
Resolving deltas: 100% (413/413), done.
Checking connectivity... done.
```

2.2 Build the Base BSP Image

1. On the host machine, create a working directory. For example, let this directory be `/home/user/development.`
2. Download the BSP for Yocto Project* from GitHub to your host machine
   - HTTPS directly from https://github.com/intel/iotg-yocto-bsp-public by selecting the appropriate tag version, for example, MR4-B-02, from the top left menu or
   - SSH using the following commands:

```
$ cd /home/user/development

$ git clone https://github.com/intel/iotg-yocto-bsp-public.git -b
e3900/master
```

This Git tree is maintained as single product branch. To get code base from the previous release, for example the MR4-B-02 release, check out to the specific tag.

```
$ git checkout MR4-B-02
```

3. Run the `setup.sh` script in `iotg-yocto-bsp-public` to prepare the combo-layer build environment.

Note: Driver Option: Select 1. Build Kernel image with CAVS HD Audio driver (Default).

```
$ cd /home/user/development/iotg-yocto-bsp-public/

$ ./setup.sh

Select an option:
1. Build kernel image with CAVS HD Audio driver (Default)
2. Build kernel image with CAVS SSP Audio driver
Default option is build kernel image with CAVS HD Audio driver. If
no input is received within 20 secs, default will be used.

Select an option:
1. core-image-sato-sdk (Default)
2. core-image-sato
3. linux-kernel
4. custom
Default build target is core-image-sato-sdk. If no input is
received within 20 secs, default target will be built.
......
```

Note: Image Option: Select 1. core-image-sato-sdk (Default)

4. The build is successful if the message "`BSP Build: PASSED!!!`" appears, and a bootable image i.e `core-image-sato-sdk-intel-corei7-64-*.hddimg`, is available under `yocto_build/build/tmp/deploy/images/intel-corei7-64/`

2.3 Boot the Target Machine

1. Insert the USB flash drive to the USB port of the host machine and burn the live image using the "`dd`" command.

```
$ sudo dd if=tmp/deploy/images/intel-corei7-64-cavs-hda/core-image-
sato-sdk-intel-corei7-64-cavs-hda.hddimg of=/dev/sdc && sync
```

2. Remove the USB flash drive from the host machine.
3. Connect the keyboard, DisplayPort* to the monitor, hard disk drive (HDD)/Solid State Drive (SSD), USB flash drive, and power cable to the target device.
4. Click the power button to start the target device.

Note: Steps 5–8 are applicable only to Intel Customer Reference Board (CRB).

5. Watch the monitor, when the Intel logo shows on the screen, click F2 key to enter the BIOS menu.
6. Select the Boot Manager. There should be two Extensible Firmware Interface (EFI) Hard Drive entries. Select the entry labeled "EFI USB Device".
7. When the GRand Unified Bootloader (GNU* GRUB) menu appears, select "install" to start the installation process.
8. Follow the on-screen instructions to install the image to the target correct storage device

2.4 Install the Intel® Media SDK on the Target Machine

Note: The following steps do not include the installation of GStreamer-MSDK plug-ins. For the installation of GStreamer-MSDK plug-ins, refer to https://github.com/01org/gstreamer-media-SDK.

1. On the host machine, extract the media binary tarball (`intel-linux-media-yocto_bxt-<version>-64bit.tar.gz`) to obtain `iHD_drv_video.so` and copy it to a USB thumb driver. Also copy MSDK tar ball (`IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz`) into the USB thumb drive.
2. On the host machine, clone the libva-staging source files to a local folder and copy it to the USB flash drive. (for example: sdb1)

```
$ git clone git://github.com/01org/iotg-lin-gfx-libva.git
$ cd iotg-lin-gfx-libva
$ git checkout a5156e951e2b7a301c00eced2799ab9f3c59d332 -b
release/mr4_staging
```

```
$ cd ..
$ cp -R iotg-lin-gfx-libva /dev/sdb1/.
```

3. Insert the USB flash drive into the target machine. Transfer the Media SDK tarball and libVA-staging source files from the USB flash drive to a local folder (for example: downloads) on the target machine. Also, transfer the iHD VA driver to `/usr/lib/dri` as shown below:

```
$ mkdir downloads.
$ cd downloads
$ cp /run/media/sda1/IntelMediaSDK2017forEmbeddedLinux-
<version>.tar.gz
$ cp -R /run/media/sda1/iotg-lin-gfx-libva/
$ cp /run/media/user/lib/iHD_drv_video.so /usr/lib/dri/
```

4. Add environment variables.

```
$ vi /etc/environment

Insert env variables below:

export LIBVA_DRIVER_NAME=iHD
export LIBVA_DRIVERS_PATH=/usr/lib/dri
export LD_LIBRARY_PATH=/usr/lib/media-
libva:/opt/intel/mediasdk/samples/_bin/x64:/opt/intel/mediasdk/lib6
4/
export XDG_RUNTIME_DIR=/run/wayland
```

5. On the target machine, build and install the libva library.

```
$ cd ~/downloads/iotg-lin-gfx-libva/
$ chmod +x autogen.sh
$ ./autogen.sh --prefix=/usr --libdir=/usr/lib --enable-wayland --
enable-x11
$ make -j 4
$ make install
```

6. Unzip the tar ball and install the Intel® Media SDK package.

```
$ tar -xvzf IntelMediaSDK2017forEmbeddedLinux-<version>tar.gz
$ cd IntelMediaSDK2017forEmbeddedLinux-<version>
$ rpm -ivh intel-linux-mediasdk-<version>.yocto.x86_64.rpm
$ rpm -ivh --nodeps intel-linux-mediasdk-devel-
<version>.yocto.x86_64.rpm
```

7. Ensure that the Intel® Media SDK library can be found. By default, the dispatcher searches in `/opt/intel/mediasdk/lib64/`. The `libmfxhw64-p.so.<version>` and `libmfxhw64.so` files can be located in library `/opt/intel/mediasdk/lib64/`.

8. Reboot and make sure `$MFX_HOME` is set.

9. Go to Chapter 4.0 to verify the installation.

3.0 *Advanced Users*

This section is for advanced users who are already familiar with setting up Yocto Project*-based build machines (detailed steps to setup a Yocto* build machine are not covered here), and describes the procedure to add all the necessary recipes during the build process so the generated BSP image contains the pre-installed IoT Media stack. Therefore, there is no need to install any of the components of the SW stack manually at runtime. Upon booting the target machine with this BSP image containing the pre-installed IoT Media stack, users can directly run Intel® Media SDK samples.

3.1 Prerequisites

- Set up a build machine in order to build the Yocto Project* BSP image for "Intel Atom® E3900 SoC Family". Refer to https://github.com/intel/iotg-yocto-bsp-public/tree/e3900/master for the instructions to set up the build machine.

Note: The IoT Media stack comprises of the Intel® Media SDK binaries, User Mode VAAPI driver (referred to as iHD-va driver in this document), libva-staging, GStreamer* (GST), and Intel® Media SDK plug-ins.

- The Yocto Project*-based BSP Base meta layer tarball i.e., iotg-yocto-bsp-public is available at https://github.com/intel/iotg-yocto-bsp-public/tree/e3900/master.
- Refer to https://software.intel.com/en-us/media-sdk to obtain the Intel® Media SDK binary tarball, i.e `IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz`, and http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html to obtain Yocto recipes for IoT Media SDK stack i.e `meta-intel-msdk.tar.bz2`. (or `meta-intel-proprietary-msdk.tar.bz2`).
- The iHD driver binary is packaged in Intel® Media binary tarball (referred to as media binary tarball). Refer to http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html to obtain the media binary tarball i.e `intel-linux-media-yocto_bxt-<version>-64bit.tar.gz`

3.2 Generating Bootable Image with Pre-Installed Intel® Media SDK

1. Create a working directory on the build machine. For instance, let this directory be `/home/user/development`.
2. Download the BSP for Yocto Project* from GitHub to your host machine
   - HTTPS directly from https://github.com/intel/iotg-yocto-bsp-public/tree/e3900/master by selecting the appropriate tag version, for example, MR4-B-02, from the top left menu or

- SSH using the following commands:

```
$ cd /home/user/development
$ https://github.com/intel/iotg-yoctobsp-public.git -b
e3900/master
```

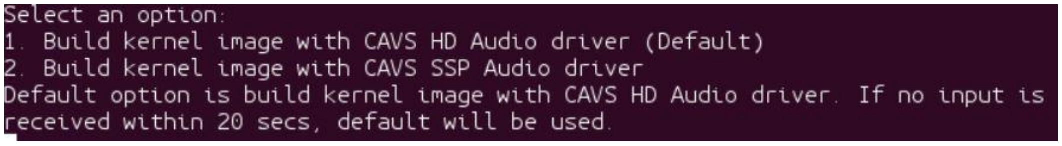
This Git tree is maintained as single product branch. To get code base from the previous release, for example the MR4-B-02 release, check out to the specific tag.

```
$ git checkout MR4-B-02
```

3. Run `setup.sh` script in `iotg-yocto-bsp-public` to prepare the combo-layer build environment.

```
$ cd /home/user/development/iotg-yocto-bsp-public
$ ./setup.sh
```

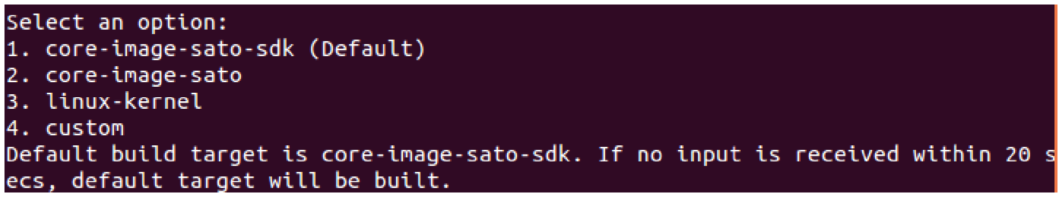
4. First, you are prompted to select the Audio driver:



```
Select an option:
1. Build kernel image with CAVS HD Audio driver (Default)
2. Build kernel image with CAVS SSP Audio driver
Default option is build kernel image with CAVS HD Audio driver. If no input is
received within 20 secs, default will be used.
```

Select option `1. Build Kernel image with CAVS HD Audio driver (Default).`

5. Next you are prompted to choose the image type. Select option `4. Custom`



```
Select an option:
1. core-image-sato-sdk (Default)
2. core-image-sato
3. linux-kernel
4. custom
Default build target is core-image-sato-sdk. If no input is received within 20 s
ecs, default target will be built.
```

6. Upon selecting option 4, configure the appropriate meta layers needed to pre-install the IOTG media stack during BitBake.
7. Once `setup.sh` script has run, there is a new folder named yocto_build created under `/home/user/development/`.
8. Create another directory in the build machine to place all the binary tarballs. For instance, let this directory be `/home/user/rpm-binary`.
9. Place Media SDK binary tarball i.e `IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz` & media binary tarball i.e `intel-linux-media-yocto_bxt-<version>-64bit.tar.gz` in this directory.
10. Extract the packages.

```
$ cd /home/user/rpm-binary/
$ tar xf IntelMediaSDK2017forEmbeddedLinux-<version>.tar.gz --
strip 1
```

```
$ tar xf intel-linux-media-yocto_bxt-<version>-64bit.tar.gz
```

11. Place the Intel® Media SDK meta layer tarball i.e `meta-intel-msdk.tar.bz2` in `/home/user/development`.

12. Extract the package.

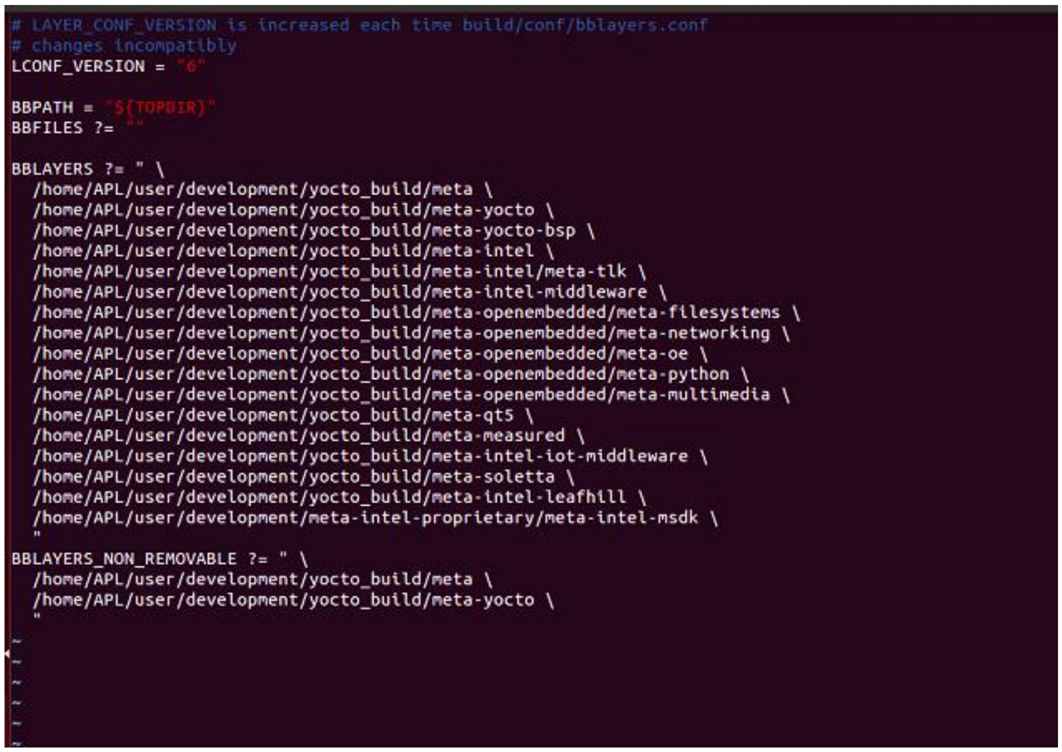
    ```
    $ cd /home/user/development
    ```

    ```
    $ tar xvjf meta-intel-msdk.tar.bz2
    ```

13. Edit `bblayer.conf` under`/home/user/development/yocto_build/build/conf` to include `meta-intel-msdk` layer

    ```
    $ vim bblayers.conf
    ```

    Add the paths of `meta-intel-msdk` as shown in the screenshot below.

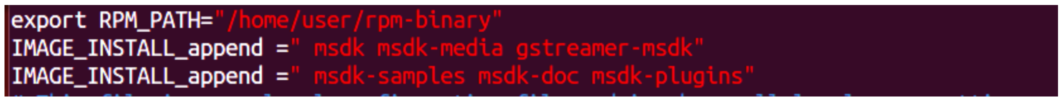


```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/APL/user/development/yocto_build/meta \
  /home/APL/user/development/yocto_build/meta-yocto \
  /home/APL/user/development/yocto_build/meta-yocto-bsp \
  /home/APL/user/development/yocto_build/meta-intel \
  /home/APL/user/development/yocto_build/meta-intel/meta-tlk \
  /home/APL/user/development/yocto_build/meta-intel-middleware \
  /home/APL/user/development/yocto_build/meta-openembedded/meta-filesystems \
  /home/APL/user/development/yocto_build/meta-openembedded/meta-networking \
  /home/APL/user/development/yocto_build/meta-openembedded/meta-oe \
  /home/APL/user/development/yocto_build/meta-openembedded/meta-python \
  /home/APL/user/development/yocto_build/meta-openembedded/meta-multimedia \
  /home/APL/user/development/yocto_build/meta-qt5 \
  /home/APL/user/development/yocto_build/meta-measured \
  /home/APL/user/development/yocto_build/meta-intel-iot-middleware \
  /home/APL/user/development/yocto_build/meta-soletta \
  /home/APL/user/development/yocto_build/meta-intel-leafhill \
  /home/APL/user/development/meta-intel-proprietary/meta-intel-msdk \
  "

BBLAYERS_NON_REMOVABLE ?= " \
  /home/APL/user/development/yocto_build/meta \
  /home/APL/user/development/yocto_build/meta-yocto \
  "
~
~
~
~
~
~
```

14. Edit the `local.conf`

    ```
    $ vim local.conf
    ```

    Export the path to all the binary tarballs i.e `/home/user/rpm-binary`, and also include recipes for the Media SDK components (binaries, samples, documentation and so on), iHD va-driver, and GStreamer-MSDK plug-ins as shown below.

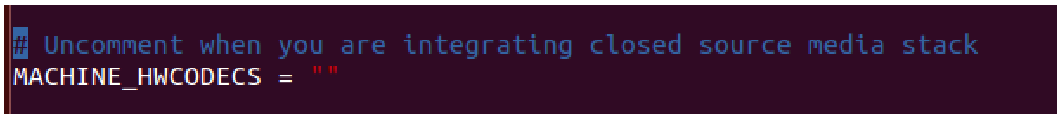


```
export RPM_PATH="/home/user/rpm-binary"
IMAGE_INSTALL_append =" msdk msdk-media gstreamer-msdk"
IMAGE_INSTALL_append =" msdk-samples msdk-doc msdk-plugins"
```

Note: There is a <space> before MSDK as shown in screenshot above.

15. Edit `local.conf` to uncomment the line as shown in the picture below.

    ```
    $ vim local.conf
    ```



```
# Uncomment when you are integrating closed source media stack
MACHINE_HWCODECS = ""
```

16. You are now ready to build the image containing the pre-installed IoT media stack.
17. Go up one level above to build/.

    ```
    $ cd ..
    ```

    Prepare the environment to run the BitBake command.

    ```
    $ source ../oe-init-build-env
    ```

18. Start the image compilation.

    ```
    $ bitbake core-image-sato-sdk
    ```

Note: Recommend starting with a brand-new image build. However, if reusing the previous build, execute the following steps before image compilation.

```
$ bitbake libva -c cleanall && bb virtual/libva -c cleanall

$ bitbake libva -c cleansstate && bb virtual/libva -c
cleansstate

$ bitbake linux-firmware -c cleanall
```

19. Once the build is successfully complete, the bootable image is available at tmp/deploy/images/intel-corei7-64 under the build directory.
20. Boot the Target Machine. Refer to Section 2.3 for booting up the image.

§

4.0 Verifying Correct Installation

1. Once the system has booted, confirm that an Intel VGA adapter can be found.

```
$ lspci -nn
...
00:02.0 VGA compatible controller [0300]: Intel Corporation Device
[8086:5a84] (rev 0a)
...
```

2. Add the user(s) who run Intel® Media SDK applications to the video group.

```
$ sudo usermod -a -G video $USER
```

3. Confirm that the i915 module is loaded correctly.

```
$ lsmod | grep 'i915'
i915                 1138688 2
drm_kms_helper       126976  1      i915
drm                  352256  2      i915,drm_kms_helper
i2c_algo_bit         13564   1      i915
video                24576   1      i915
```

4. Communication with the DRM library occurs via `/dev/dri/cardx` handler (usually `/dev/dri/card0`, though there can be more entries if there are more graphics adapters). Quite often permissions are set so that root access is required for users not in the video group. Add Intel® Media SDK application users to the video group and/or ensure users have permissions to work as a regular user.

```
$ sudo chmod 666 /dev/dri/card0
```

5. Ensure that the Intel® Media SDK library can be found. By default, the dispatcher searches in `<sdk-install-dir>/lib64/`. The `libmfxhw64-p.so.<version>` and `libmfxhw64.so` files can be located in library `<sdk-install-dir>/lib64/`. For example, the library search path can be adjusted by the LD_LIBRARY_PATH variable.

```
$ export LD_LIBRARY_PATH=$MEDIASDK_INSTALL_FOLDER/lib64
```

6. Make sure the following environment variables are set.

```
$ export LIBVA_DRIVER_NAME=iHD
```

```
$ export LIBVA_DRIVERS_PATH=/usr/lib/dri
```

Or

```
$ export LIBVA_DRIVERS_PATH=/usr/lib64 (if multilib is turned
on in the user environment
```

7. Run “vainfo” to verify the correct installation of the iHD-va-driver.

```
$ vainfo
[sudo] password for mediasdk:
```

```
error: can't connect to X server!
libva info: VA-API version 0.35.0
libva info: va_getDriverName() returns 0
libva info: Trying to open /usr/lib/dri/lib64/iHD_drv_video.so
libva info: Found init function __vaDriverInit_0_32
libva info: va_openDriver() returns 0
vainfo: VA-API version: 0.99 (libva 1.67.0.pre1)
vainfo: Driver version: 16.y-xxxx
vainfo: Supported profile and entrypoints
VAProfileNone : VAEntrypointVideoProc
VAProfileNone : <unknown entrypoint>
VAProfileMPEG2Simple : VAEntrypointEncSlice
VAProfileMPEG2Simple : VAEntrypointVLD
VAProfileMPEG2Main : VAEntrypointEncSlice
VAProfileMPEG2Main : VAEntrypointVLD
VAProfileH264Baseline : VAEntrypointEncSlice
VAProfileH264Baseline : <unknown entrypoint>
VAProfileH264Baseline : <unknown entrypoint>
VAProfileH264Main : VAEntrypointVLD
VAProfileH264Main : VAEntrypointEncSlice
VAProfileH264Main : <unknown entrypoint>
VAProfileH264Main : <unknown entrypoint>
VAProfileH264High : VAEntrypointVLD
VAProfileH264High : VAEntrypointEncSlice
VAProfileH264High : <unknown entrypoint>
VAProfileH264High : <unknown entrypoint>
VAProfileVC1Simple : VAEntrypointVLD
VAProfileVC1Main : VAEntrypointVLD
VAProfileVC1Advanced : VAEntrypointVLD
VAProfileJPEGBaseline : VAEntrypointVLD
VAProfileJPEGBaseline : VAEntrypointEncPicture
```

Pre-built samples are included in /opt/intel/mediasdk/samples/_bin/x64.

Note: For Intel® Media SDK 2017 for Embedded Linux*, there is no software implementation. For these samples to work "-hw" must be specified on the command line.

§

5.0 Compiling and Running Intel® Media SDK Samples on the Target Machine

1. Build the Intel® Media SDK samples for testing; ensure `$MFX_HOME` is set to the directory corresponding to your build. The build.pl script only builds samples if the prerequisites can be found. It includes rendering to X11 compositor by default.

```
$ cd /opt/intel/mediasdk/samples/
$ export |grep MFX_HOME
$ perl build.pl --cmake=intel64.make.release -build
```

2. DRM frame buffer rendering is supported by default.

```
$ cd /opt/intel/mediasdk/samples/_bin/x64/sample_decode h264 -hw -
vaapi -i /path/to/h.264 -rgb4 -rdrm
```

3. (Optional) Build by enabling X11 DRI3/present support. For more details, refer to the white paper on DRI3 implementation at this location: http://www.intel.com/content/www/us/en/embedded/products/apollo-lake/technical-library.html

Note: Intel® Media SDK sample binaries included into the package are built with Wayland* support.

If graphic interface support is required, the following steps can be used:

a. Use "–enable-x11-dri3=yes" for build.pl during compilation:

```
$ perl build.pl --cmake=intel64.make.release -enable-x11-dri3=yes -
build
```

b. Create `04-dri3.conf` in `/usr/share/X11/xorg.conf.d` as shown below:

```
$ vi /usr/share/X11/xorg.conf.d/04-dri3.conf
Section "Device"
Identifier        "Card0"
Driver  "intel"
BusID             "0:2:0"
Screen             0
   Option "AccelMethod" "sna"
   Option "DRI" "3"
EndSection
```

c. Run the decode sample by specifying conversion to RGB4 ("-rgb4" since Present Pixmap could only take RGB) and "-r".

```
$
/opt/intel/mediasdk/samples/__cmake/intel64.make.release/__bin/rele
ase
$ ./sample_decode h264 -vaapi -hw -i
/path/to/elementary/AVC/stream/ -rgb4 -r
```

4. To enable Wayland* support:

a. Use "–enable-wayland=yes" for build.pl during compilation:

```
$ perl build.pl --cmake=intel64.make.release -enable-wayland=yes -
build
```

b. Run the following command to load Weston*.

```
$ export XDG_RUNTIME_DIR=/tmp
$ weston
```

c. Run the decode sample (execute commands below from Weston terminal).

```
$ cd
/opt/intel/mediasdk/samples/__cmake/intel64.make.release/__bin/rele
ase
$ ./sample_decode h264 -vaapi -hw -i
/path/to/input/elementary/AVC/stream   -rwld
```

§

6.0 Installing for Non-Supported Configurations

It's possible that the Intel® Media SDK will work in other configurations. For full support, using one of the supported configurations is required. However, you are free use the Intel® Media SDK in other settings if you are willing to go through an extra step in reporting issues. If an issue can be reproduced in one of the supported configurations, it can be addressed, otherwise, you are on your own.

Since Intel® Media SDK is based on hardware access via the video driver, the main concern with alternative installations is making sure that all device IDs and other changes to the kernel are available. Usually patches required to enable working with the hardware are submitted to the kernel repository tip relatively quickly. If an advanced kernel or compile are used from close to the tip its likely to get most, if not all, changes required for Intel® Media SDK to work.

With this approach at this point unfortunately there are no guarantees. Enabling Intel® Media SDK to work on a wider set of configurations is a work in progress.

Important Note: When using your own kernel configuration, make sure `CONFIG_MMU_NOTIFIERS=y` is enabled. It can be disabled implicitly by disabling virtualization support.

§