RDK Video Guide

This document guides users to perform the below activties on a Raspberry Pi (Rpi) platform as well as an Amlogic Reference platform

1. Setup a development environment for an IPSTB target
2. Bring up the device
3. Create & run a sample lightning app in the device

On this Page:

Before you begin

Basic Skills

Though not mandatory, the below skills will help the user to understand RDK, RDK build, and to try out RDK better :

- Familiar with Linux based platforms
- Familiar with Yocto
- Familiar with RDK basics
- Knowledge in using Raspberry Pi
- Experience in setting up boards

Relevant RDK links

RDK Support

- Forum: FORUMS Home
- JIRA: https://jira.rdkcentral.com
- Support: RDK Support

RDK Source Code

- Gerrit: https://code.rdkcentral.com
- Component Owners: RDK Component owners
- Code Contribution Process: How to Contribute

RDK V Releases

- RDK-V Releases

API Documentation

- RDK-V Opensourced Doxygen Documentation : Opensourced API Documentation

FAQ Documetation

- RDK FAQ

ThunderJS documentation

- An elaborate ThunderJS documentation is available here - https://github.com/rdkcentral/ThunderJS

Prerequisites

Requirement	Yocto 3.1 LTS (Dunfell)

Linux PC	64 bit Ubuntu 18.04 LTS Precise supported distributions and versions are here
Free HDD Space	Minimum 100GB Free memory space
Host Tools version	Git 1.8.3.1 or greater tar 1.24 or greater Python 2.7.3
Raspberry Pi development kit	
IPSTB Reference board	Access to repositories hosting code and binaries for reference board
Peripherals	TV, Keyboard

Environment

Rpi & IPSTB similarities:

- Reference devices to try and run RDK
- No need of RDK license to try out IPSTB builds

Difference between Raspberry Pi (Rpi) and IPSTB:

Raspberry Pi (Rpi)	IPSTB
Generic hobby device altered in software for STB capabilities	Hardware specifically made for STB purpose
Low end device capable of only mimicking STB capabilities	Regular hardware used in real STBs
Available in general market	Available for licensed users of SoC vendor
No licenses required to generate and use builds	Agreement with IPSTB SoC vendor required to obtain software licenses (such as SDK, Kernel etc.)
Rpi builds supported in all quarterly releases	IPSTB builds might not be regularly supported in all quarterly releases

Host Setup

Install the following packages for setting up your host VM

The instructions provided below are meant to be executed via the command line on an Ubuntu machine

for yocto 3.1 (dunfell)

```
# essential packages installation
# super user mode is required

# major essential packages
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib g++-multilib build-essential
chrpath socat bison curl cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-
git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3 xterm
```

Configure bash as default command interpreter for shell scripts

```
sudo dpkg-reconfigure dash
```

Select “No”
To choose bash, when the prompt asks if you want to use dash as the default system shell - select “No”

Configure Git

Upgrade your Git version to 1.8.x or higher

Once git is installed, configure your name and email using the below commands

```
# review your existing configuration
git config --list --show-origin

# configure user name and email address
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com

# configure git cookies. Needed for Gerrit to only contact the LDAP backend once.
git config --global http.cookieFile /tmp/gitcookie.txt
git config --global http.saveCookies true
```

Configure repo

In order to use Yocto build system, first you need to make sure that repo is properly installed on the machine:

```
# create a bin directory
mkdir ~/bin
export PATH=~/bin:$PATH

# Download the repo tool and ensure that it is executable
curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

Credential configuration (Only for Amlogic reference board)

Note: it is also recommended to put credentials in .netrc when interacting with the repo.

A sample .netrc file is illustrated below

machine code.rdkcentral.com

login <YOUR_USERNAME>

password <YOUR_PASSWORD>

Build

Build basic image for Rpi

```
# initialize the manifest with repo tool
repo init -u https://code.rdkcentral.com/r/manifests -b dunfell -m rdkv-nosrc.xml
repo sync -j `nproc` --no-clone-bundle --no-tags
MACHINE=raspberrypi-rdk-mc source meta-cmf-raspberrypi/setup-environment
bitbake rdk-generic-mediaclient-wpe-image
```

The generated image resides under the directory `build-<MACHINE>/tmp/deploy/images/<MACHINE>` of the Yocto workspace

Build basic image for Amlogic Reference Board

Enabled the distro(ipclient) for IPSTB as a setup environment option, so during build time we can pass this as an argument.

```
repo init -u https://code.rdkcentral.com/r/rdk/soc/amlogic/aml-manifests -b rdk-next -m aml_ipstb.xml
repo sync -j4 --no-clone-bundle
export LOCAL_BUILD=1
source meta-rdk-aml/set-env.sh mesonsc2-5.4-lib32-ah212 --ipclient --playready --widevine
bitbake lib32-rdk-generic-mediaclient-image
```

The generated image resides under the directory `build-<MACHINE>/tmp/deploy/images/<MACHINE>` of the Yocto workspace

Flash image and bring up

Flash image and bring up Rpi

Flash image

1. Insert an SD card in the SD card port of the USB SD card reader (or Laptop).

   Prefer to use 32gb sd card and there should be minimum 12gb free space available in the device .

2. Verify that the SD card has been detected by executing either of the commands listed below

```
$lsblk
$sudo fdisk –l
```

```
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 931.5G  0 disk
sda1   8:1    0   350M  0 part
sda2   8:2    0     3G  0 part
sda3   8:3    0 896.4G  0 part /
sda4   8:4    0     1K  0 part
sda5   8:5    0  31.8G  0 part [SWAP]
sdb      8:16   1  14.9G  0 disk
sdb1   8:17   1    40M  0 part /media/raspberrypi
sdb2   8:18   1   552M  0 part /media/dd5efb34-1d40-4e50-bbc2-a75d3e02af97
sr0     11:0    1  1024M  0 rom
```

3. Type the following command to ensure that the partitions, if present, on the SD card are not mounted

```
$mount
```

```
$ mount
/dev/sda3 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,noexec,nosuid,nodev)
rpc_pipefs on /run/rpc_pipefs type rpc_pipefs (rw)
nfsd on /proc/fs/nfsd type nfsd (rw)
none on /tmp/guest-zdrO76 type tmpfs (rw,mode=700)
gvfs-fuse-daemon on /var/lib/lightdm/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,nodev,user=lightdm)
/dev/sdb1 on /media/raspberrypi type vfat (rw,nosuid,nodev,uid=136,gid=148,shortname=mixed,dmask=0077,utf8=1,
showexec,flush,uhelper=udisks)
/dev/sdb2 on /media/dd5efb34-1d40-4e50-bbc2-a75d3e02af97 type ext3 (rw,nosuid,nodev,uhelper=udisks)
```

4. Repeat the below command to unmount all the mounted partition present on the SD card.

```
$umount <partition-mountpoint>
```

```
$ sudo umount /dev/sdb1
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 931.5G  0 disk
sda1   8:1    0   350M  0 part
sda2   8:2    0     3G  0 part
sda3   8:3    0 896.4G  0 part /
sda4   8:4    0     1K  0 part
sda5   8:5    0  31.8G  0 part [SWAP]
sdb      8:16   1  14.9G  0 disk
sdb1   8:17   1    40M  0 part
sdb2   8:18   1   552M  0 part /media/dd5efb34-1d40-4e50-bbc2-a75d3e02af97
sr0     11:0    1  1024M  0 rom
$ sudo umount /dev/sdb2
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 931.5G  0 disk
sda1   8:1    0   350M  0 part
sda2   8:2    0     3G  0 part
sda3   8:3    0 896.4G  0 part /
sda4   8:4    0     1K  0 part
sda5   8:5    0  31.8G  0 part [SWAP]
sdb      8:16   1  14.9G  0 disk
sdb1   8:17   1    40M  0 part
sdb2   8:18   1   552M  0 part
sr0     11:0    1  1024M  0 rom
```

5. Execute the following command to flash the image on the SD card

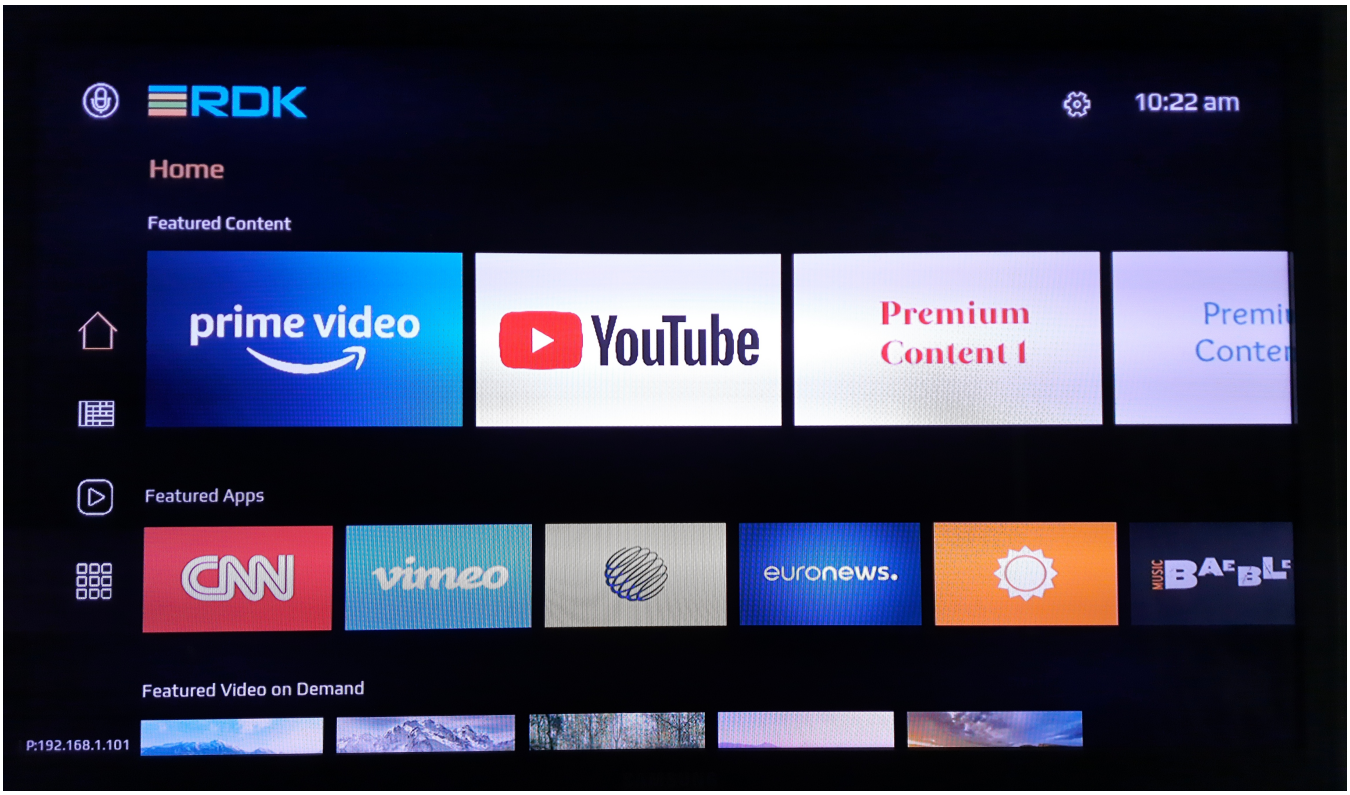
Flash Command

```
$sudo dd if=<path to ImageName.Rpi-sdimg> of=<path to SD card space> bs=4M
Example:
$sudo dd if=rdk-generic-mediaclient-wpe-image.Rpi-sdimg of=/dev/sdb bs=4M
149+0 records in
149+0 records out
624951296 bytes (625 MB) copied, 39.7752 s, 15.7 MB/s
```

6. Remove the SD card and insert it to the Raspberry Pi SD card slot

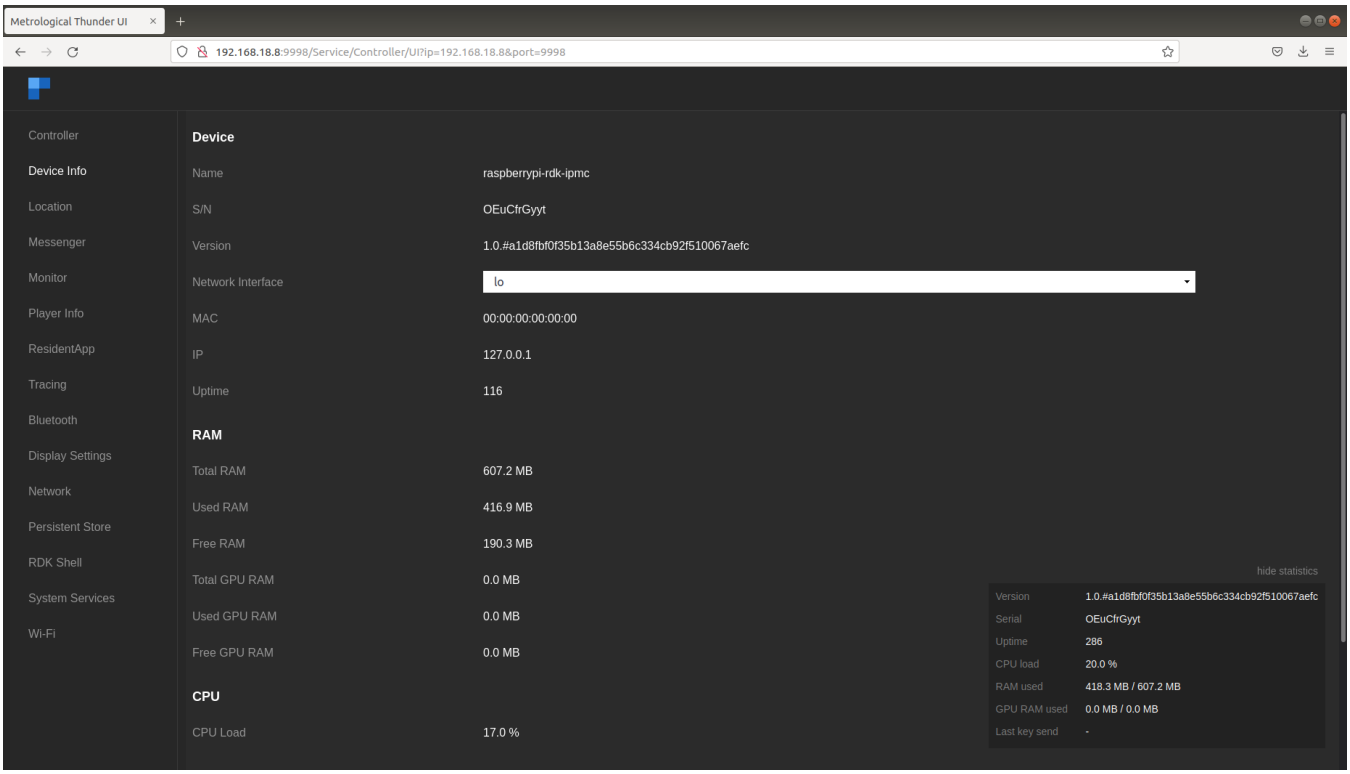
7. Power on the Rpi and Bring up the device

- TV screen will display the Raspberry Pi's IP address(referred as machineIP from now) with default RDK UI as shown below.

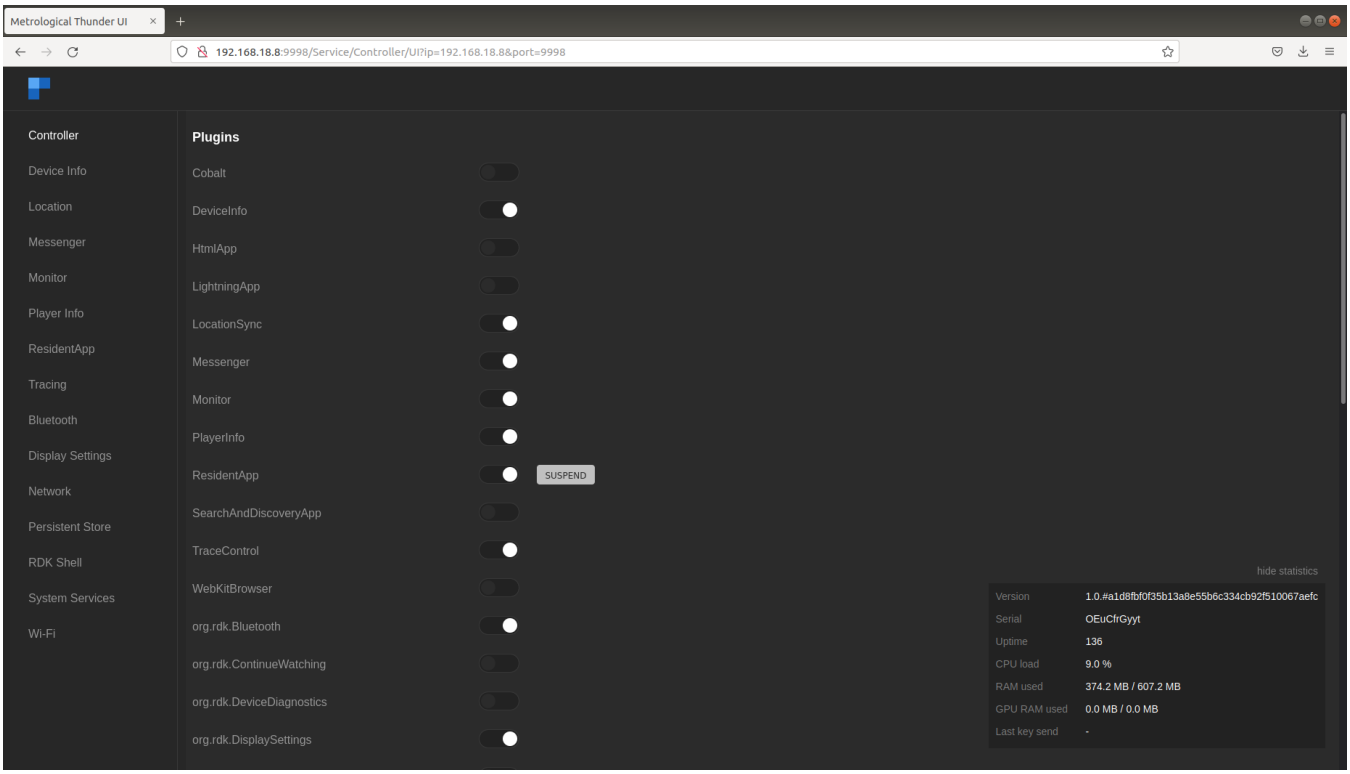


Accessing Raspberry Pi

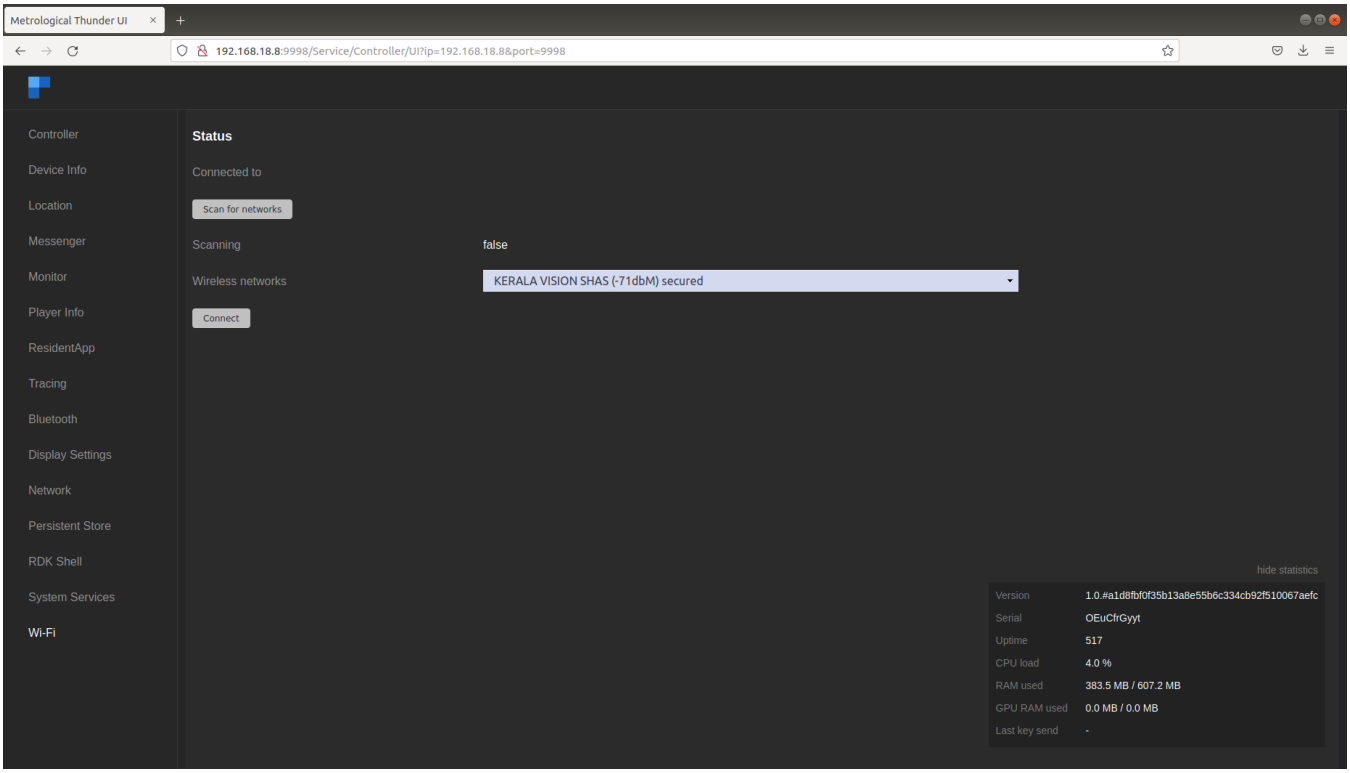
- For connecting Controller UI, use URL: http://<machineIP>:9998



- Plugins can be enabled or disabled from controller UI.



- Wifi related services can be triggered from Wi-Fi tab in controller UI. We can scan and select from available networks.



- For ssh, we can use ssh root@machineip
- For verifying the image details, we can use cat /version.txt command.

```
root@raspberrypi-rdk-ipmc:~# cat /version.txt

imagename:rdk-generic-ip-stb-client_rdk-next_20210902101930

BRANCH=rdk-next

YOCTO_VERSION=dunfell

VERSION=4.09.02.21

SPIN=0

BUILD_TIME="2021-09-02 10:19:30"

Generated on Thu Sep 02  10:19:30 UTC 2021

root@raspberrypi-rdk-ipmc:~#
```

- For playing a video using aamp-cli, launch aamp-cli from terminal, cd /usr/bin;aamp-cli.

```
root@raspberrypi-rdk-mc:/usr/bin# aamp-cli
************************************************************************
** ADVANCED ADAPTIVE MICRO PLAYER (AAMP) - COMMAND LINE INTERFACE (CLI) **
************************************************************************
1644303267:124 : [AAMP-PLAYER][-1][WARN][PlayerInstanceAAMP][144][AAMP_JS][0x104ae00]Creating GlobalConfig
Instance[0x104ff30]
1644303267:124 : [AAMP-PLAYER]ReadAampCfgTxtFile:INFO opened aamp.cfg

1644303267:124 : [AAMP-PLAYER]Parsed value for dev cfg property useWesterosSink - 1
1644303267:124 : [AAMP-PLAYER]SetValue: useWesterosSink New Owner[5]
1644303267:124 : [AAMP-PLAYER]AAMP_ENABLE_WESTEROS_SINK present, Value = 1
1644303267:124 : [AAMP-PLAYER]AAMP_ENABLE_WESTEROS_SINK present: Enabling westeros-sink.
1644303267:124 : [AAMP-PLAYER]SetValue: useWesterosSink Owner[1] not allowed to Set ,current Owner[5]
1644303267:124 : [AAMP-PLAYER]////////////////// AAMP Cfg Override Configuration  //////////
1644303267:124 : [AAMP-PLAYER]Cfg [53 ][useWesterosSink     ][cfg  ][true]
1644303267:124 : [AAMP-PLAYER]////////////////////////////////////////////////
1644303267:124 : [AAMP-PLAYER]////////////////// AAMP Config (Operator Set) //////////
1644303267:124 : [AAMP-PLAYER]////////////////////////////////////////////////
1644303267:125 : [AAMP-PLAYER]SetValue: userAgent New Owner[3]
1644303267:125 : [AAMP-PLAYER][0][WARN][CreatePipeline][1580]Creating gstreamer pipeline
1644303267:125 : [AAMP-PLAYER][0][WARN][CreatePipeline][1607]AAMPGstPlayerPipeline buffering_enabled 1
[AAMPCLI] type 'help' for list of available commands
[AAMPCLI] aamp-cli>
```

- For playing a video using gstreamer, use gst-launch-1.0.

eg :- gst-launch-1.0 playbin uri=aamp://bitdash-a.akamaihd.net/content/MI201109210084_1/m3u8s/f08e80da-bf1d-4e3d-8899-f0f6155f6efa.m3u8 video-sink=westerossink

```
root@raspberrypi-rdk-mc:~# gst-launch-1.0 playbin uri=aamp://bitdash-a.akamaihd.net/content/MI201109210084_1
/m3u8s/f08e80da-bf1d-4e3d-8899-f0f6155f6efa.m3u8 video-sink=westerossink
Setting pipeline to PAUSED ...
westeros (sink) version 1.01.28
Pipeline is PREROLLING ...

^Chandling interrupt.
Interrupt: Stopping pipeline ...
ERROR: pipeline doesn't want to preroll.
Setting pipeline to NULL ...
Freeing pipeline ...
Number of active jsmediaplayer instances: 0
root@raspberrypi-rdk-mc:~#
```

Flash image and bring Amlogic Reference Platform

Flash image

There are multiple methods to flash/upgrade image in an Amlogic reference board. For the ease of process, we will use the USB based method, which is easy as well as helps to recover even the devices that went into a bad state. The method is as simple as copying a configuration file and the image to flash in a USB and restart the device and run a command

1. Copy the below text to a file called aml_sdc_burn.ini

```
;
;Amlogic sdcard burning configure script
;This card burning script support both dos and unix file format, but don't edit in windows if not dos format
;Except comment, all must readable ASCII letters
;
[common]
erase_bootloader    = 1
erase_flash         = 0
reboot              = 0
;package will filled by sdacard burning tool
[burn_ex]
package     = aml_upgrade_package.img
;media       =
```

2. Edit the aml_sdc_burn.ini file and add the image name in it against the key 'package'. In the example, the name 'aml_upgrade_package.img' is used

```
package     = aml_upgrade_package.img
```

3. Copy the ini file and image name to a USB
4. Plug the USB to reference board. Reboot the device. Press Ctrl+C via UART until the bootloader prompt comes as shown below

```
sc2_hp44h#
```

5. Give the command 'usb_burn aml_sdc_burn.ini' in the terminal. Wait for the flashing to complete

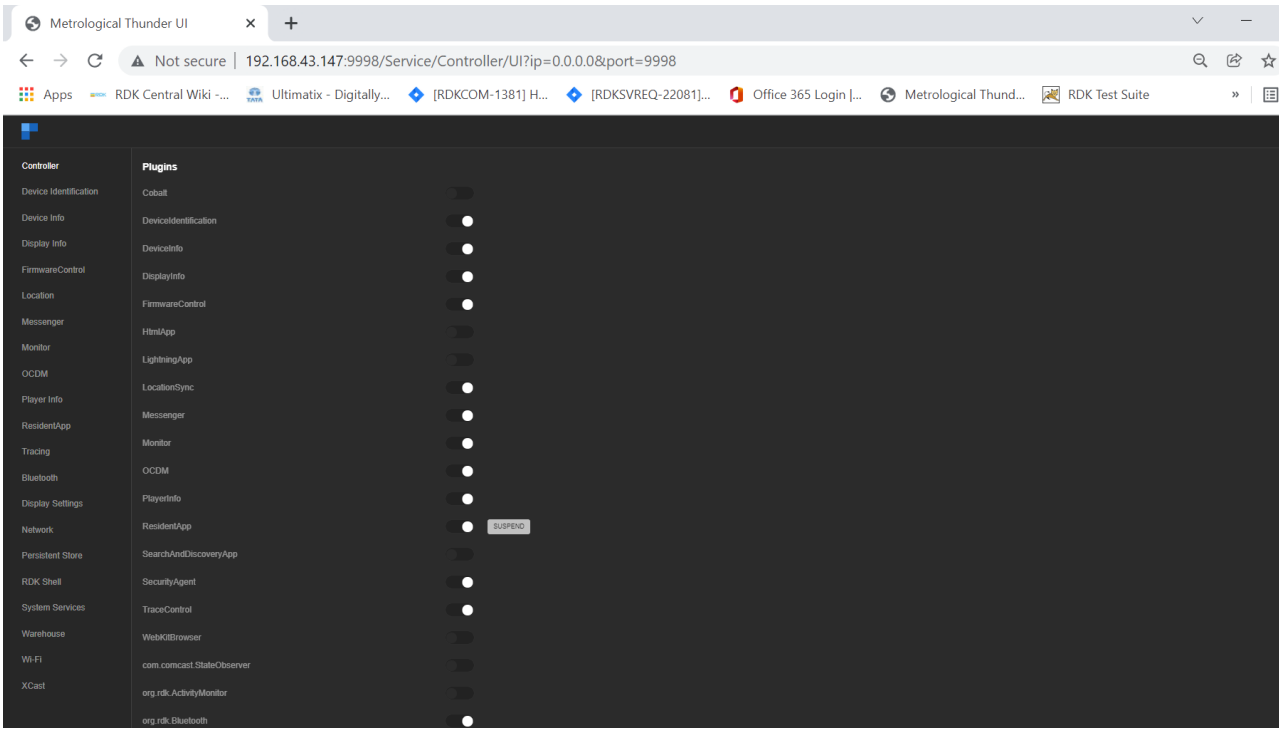
```
sc2_hp44h#usb_burn aml_sdc_burn.ini
```

6. Reboot the device upon prompted so. Wait for the device to come up

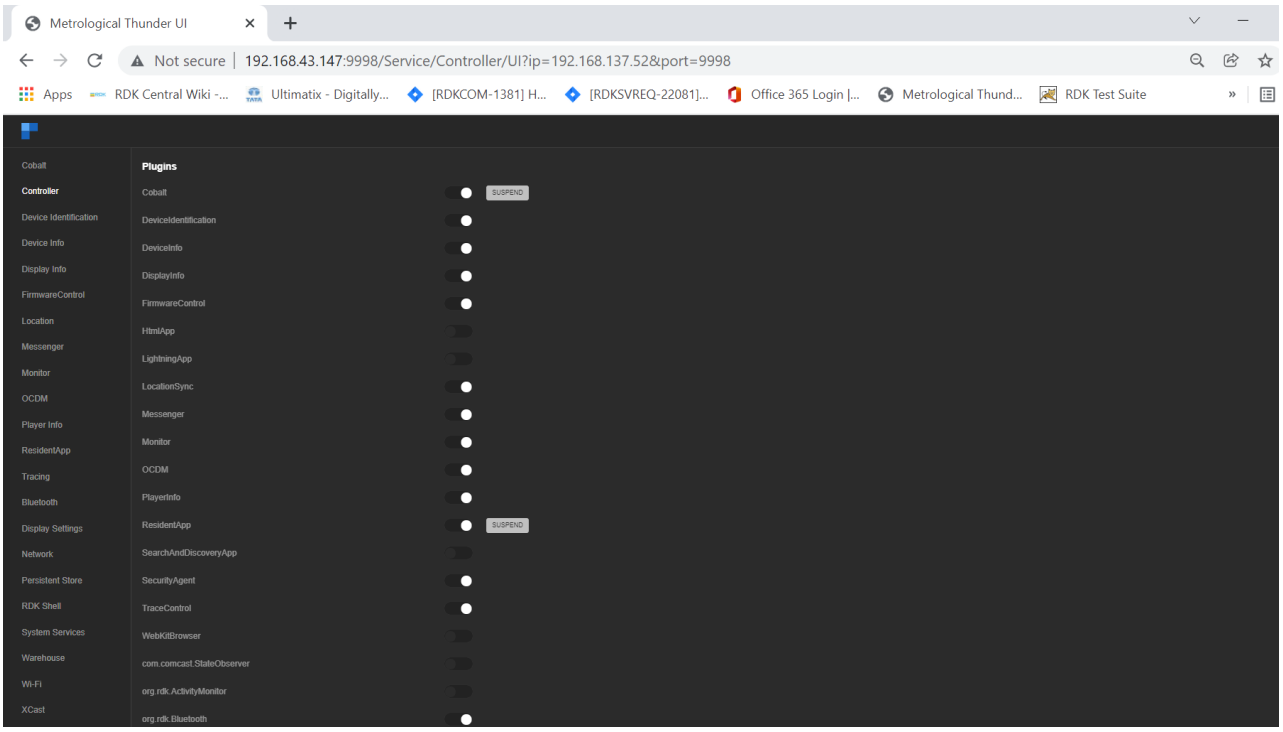
There are other image upgrade methods also available. Please refer to https://wiki.rdkcentral.com/display/RDK/Build+and+Flashing+Steps for more details

Accessing Amlogic STB

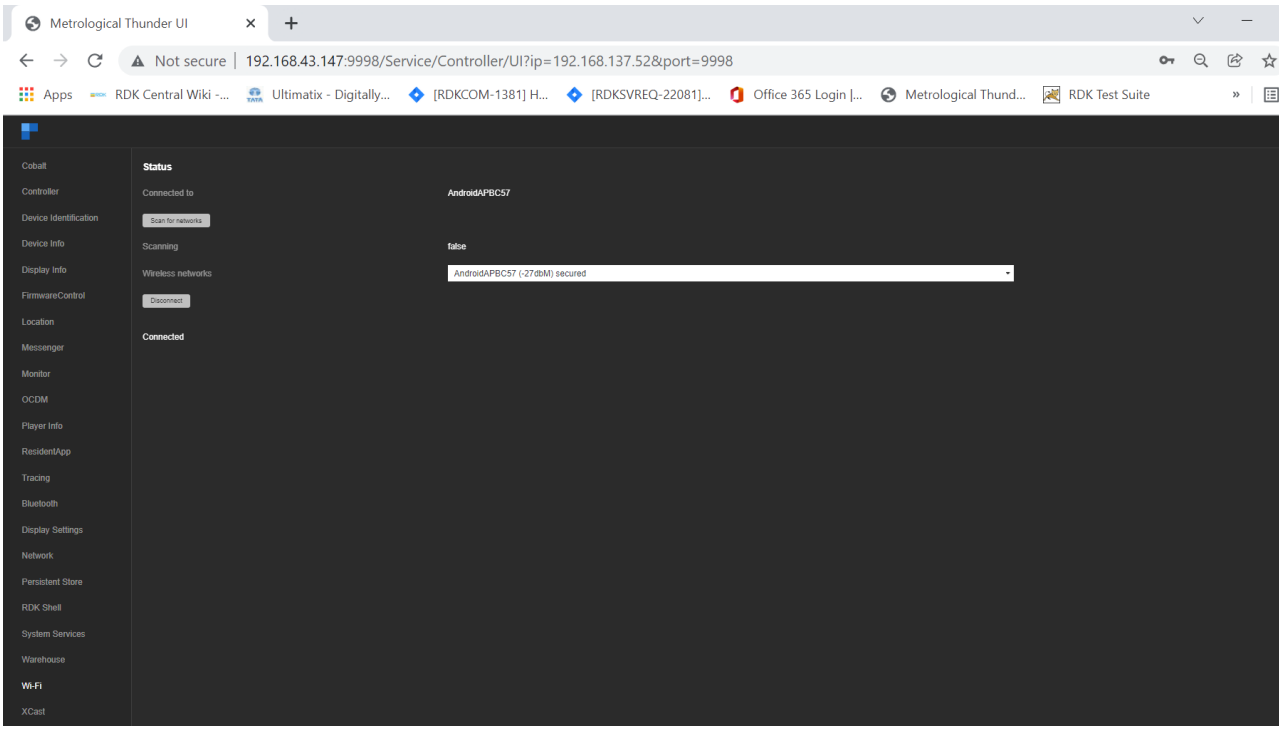
- **Controller UI :** For connecting Controller UI, use URL: http://<machineip>:9998



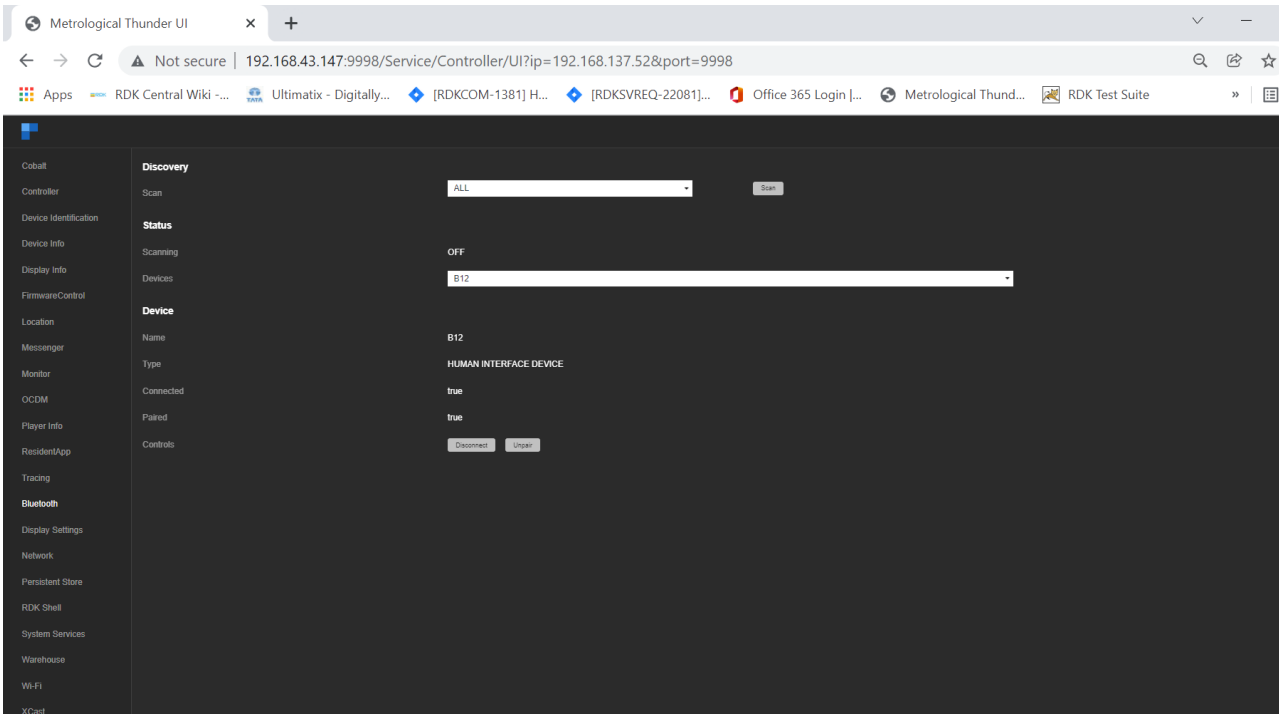
- **Aamp Playback:** aamp-cli http://rdmedia.bbc.co.uk/dash/ondemand/elephants_dream/1/client_manifest-all.mpd
- **Gstreamer Playback :** gst-launch-1.0 playbin uri=http://rdmedia.bbc.co.uk/dash/ondemand/elephants_dream/1/client_manifest-all.mpd video-sink=westerossink
- **Playback through Youtube Application:** For launching youtube from controller UI, enable cobalt plugin, youtube page will be loaded in TV UI.



- **WiFi Service**

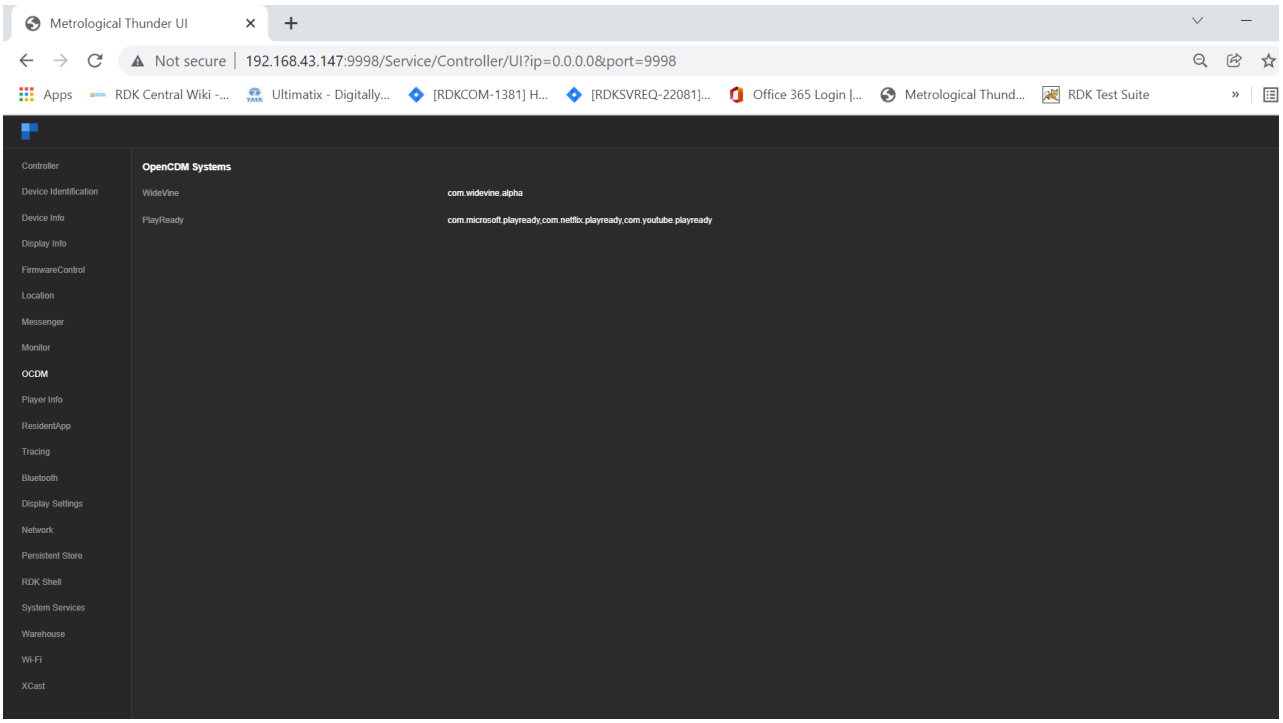


- **Bluetooth Service**



- **DRM:**

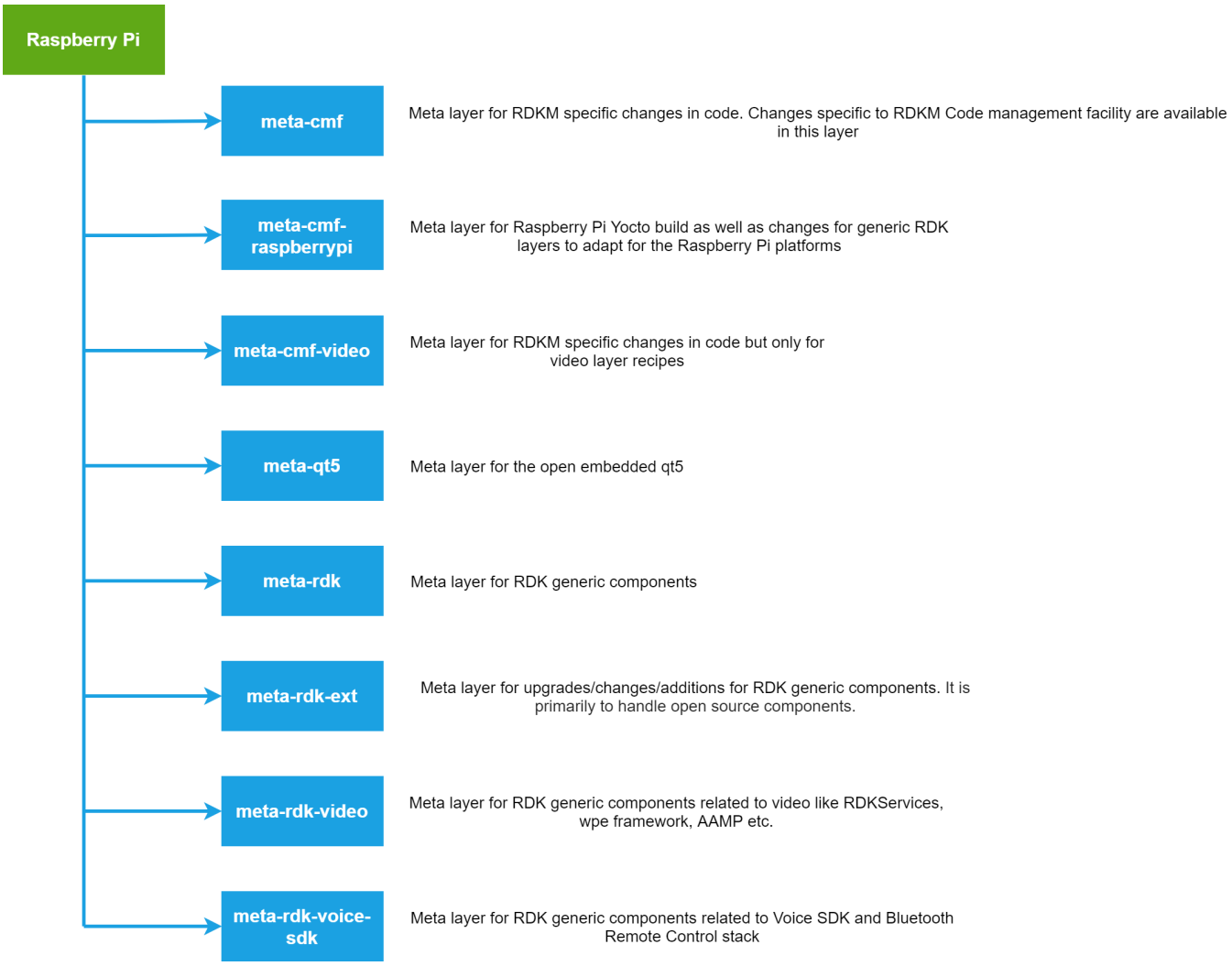
aamp-cli https://demo.unified-streaming.com/k8s/features/stable/video/tears-of-steel/tears-of-steel-dash-playready.ism/.mpd
aamp-cli https://wowzaec2demo.streamlock.net/live/bigbuckbunny-enc-wv.stream/manifest_mvlist.mpd



Yocto recipe structure of relevant components

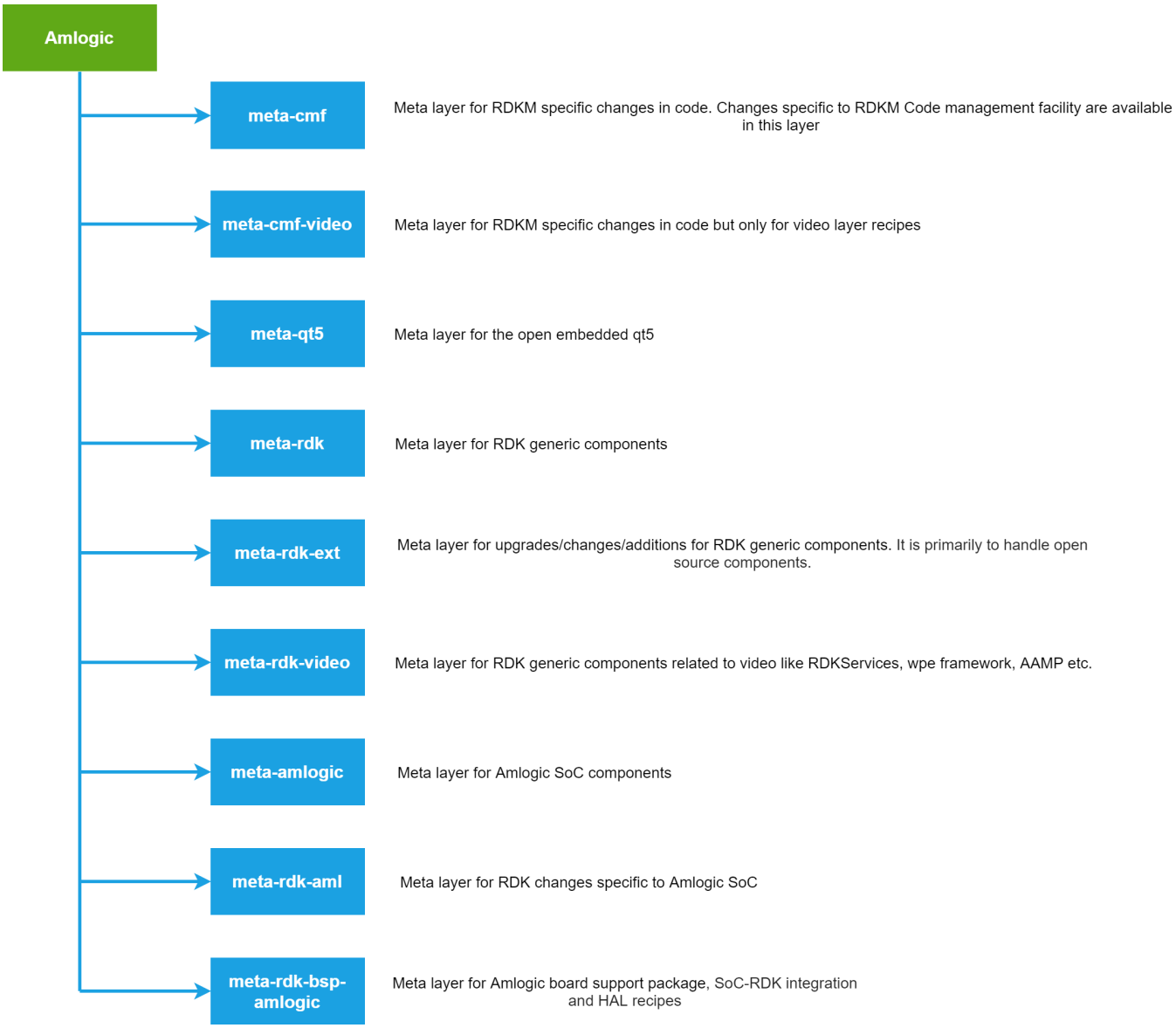
Below is the major meta-layers specific to Raspberry Pi.

Raspberry Pi major meta-layers structure:



Below is the major meta-layers specific to Amlogic IPSTB

Amlogic major meta-layers structure:



Setup and Develop Thunder plugin

Steps involved in implementing new RDK services Plug-In

RDK components implemented as Thunder plugins are called as RDKServices. it is developed based on the Thunder (WPE) Framework. Services each other or a particular service can be COMRPC for (communication between plugins) or JSONRPC (for external communication). It has a web-based controller UI.

Reference : https://github.com/rdkcentral/rdkservices/pull/960

In RDK services -plugins workspace:

Cloned from https://github.com/rdkcentral/rdkservices

```
$ git clone https://github.com/rdkcentral/rdkservices
```

Inside PluginName directory

1. <PluginName>.json : This file contains the plugin's information like schema, information and interface json file.

    a. PluginTemplate.json
2. CmakeLists.txt: CMAKE based configuration file which contains a set of directives and instructions describing the project's source files and targets. This is used to compile the Plug-in code to generate the plugin library(Shared library by default; ".so"). External dependencies can be included/linked to the target with the help of CMakeLists.txt configurations.
3. Module.h: This header file includes the support for JSON request, response, logging etc...
4. Module.cpp: This file is used to declare the module name for the Plug-in. This file contains the plugin's information like schema, information and interface json file (defined earlier).
5. <PluginName>.config: This file is used to set configurations of the Plug-in . Ex:- set (autostart true) - Used to make the Plug-in to start automatically along with wpeframework daemon
6. <PluginName>.h :Declare the plugin class in this which should contains all the structures, variables and methods which are needed for plugin implementation. The interface header auto-generated earlier will be used here,
7. <PluginName>.cpp: This class does contains all the definitions for the methods declared in the Plugin.h and those definitions should be defined inside the below namespace.
8. Cmake / (directory) :

```
PluginTemplate/
  CMakeLists.txt
  PluginTemplate.config
  PluginTemplate.cpp
  PluginTemplate.h
  PluginTemplate.json
  Module.cpp
  Module.h
  README.md
  cmake
  |   FindDS.cmake
     FindIARMBus.cmake
  doc
   PluginTemplate.md
```

<PluginName>.json

This file contains the plugin's information like schema, information and interface json file.

Syntax :

```
  {
   {
   "$schema": "plugin.schema.json",
   "info": {
   "title": "Plugin Name Plugin",
   "callsign": "PluginName",
   "locator": "libWPEFrameworkPluginName.so",
   "status": "production",
   "description": "The PluginName plugin allows retrieving of various plugin-related information.",
   "version": "1.0"
    },
    "interface": {
   "$ref": "{interfacedir}/PluginName.json#"
    }
  }
```

eg: PluginTemplate.json

```
{
    "locator":"libWPEFrameworkPluginTemplate.so",
    "classname":"PluginTemplate",
    "precondition":[
     "Platform"
    ],
    "callsign":"org.rdk.PluginTemplate",
    "autostart":false
}
```

<PluginName>.config

.config files are files used to configure the parameters and initial settings for some computer programs.

```
set (autostart false)                       #we are setting autostart condition disable
set (preconditions Platform)
set (callsign "org.rdk.PluginTemplate")         #The callsign name was given to an instance of a plugin.

#One plugin can be instantiated multiple times. but each instance, the instance-name "callsign" must be
unique. here we using org.rdk.PluginTemplate.
```

<PluginName>.h

Declare the plugin class in this which should contain all the structures, variables, and methods which are needed for plugin implementation.

```
 namespace WPEFramework {
  namespace Plugin {

   class PluginName : public PluginHost::IPlugin, public PluginHost::IWeb, public PluginHost::JSONRPC {
   public:
     PluginName()
      : _skipURL(0)
      , _service(nullptr)
      , _subSystem(nullptr)
     {
      RegisterAll();
     }

     virtual ~PluginName()
     {
      UnregisterAll();
     }
   }
   ---------------------------------------
   ---------------------------------------
  }
}
```

eg: PluginTemplate.h

for more information refer PluginTemplate.h

<PluginName>.cpp

This class does contain all the definitions for the methods declared in the PluginTemplate.h and those definitions should be defined inside the below namespace.

The plugin should register using service registration MACRO as declared below :

```
  namespace WPEFramework {
   namespace Plugin {
    SERVICE_REGISTRATION(Plugin, 1, 0);
    ---------------------------------------
    ---------------------------------------
    ---------------------------------------
   }
  }
```

To initialize and deinitialize or activate or deactivate handler for the plugin services :

```
const string PluginTemplate::Initialize(PluginHost::IShell* /* service */)
        {
        //shared pointer initialized
        //initialize external library
        LOGINFO();
        return (string());
        }
void PluginTemplate::Deinitialize(PluginHost::IShell* /* service */)
        {
         //shared pointer deinitialized
        //deinitialize external library
        LOGINFO();
        }
```

eg:

```
namespace WPEFramework {
    namespace Plugin {
         SERVICE_REGISTRATION(Plugin, 1, 0);

          //registration
          //All the methods declared in Plugin.h should be registered here

          //initialize and deinitialize the handlers for the plug-in service

          //All the methods declared in Plugin.h should be defined here

    }
}
```

CMakeLists.txt

Using the CMake utility this file contains the task needed to be done to make a plug-in. Also contains packages, libraries needed to compile, its path, and other plugin-in configuration option.

This file contains a set of directives and instructions describing the project's source files and targets (executable, library, or both).

```
set(PLUGIN_NAME PluginTemplate)                         # to set a environment variable set(<variable>
<value>)
set(MODULE_NAME ${NAMESPACE}${PLUGIN_NAME})
find_package(${NAMESPACE}Plugins REQUIRED)              # to Finds and loads settings from an external
project.

#Adds a library target called <name> to be built from the source files listed in the command invocation. The
<name> corresponds to the logical target name and must be globally unique within a project.
add_library(${MODULE_NAME} SHARED
        PluginTemplate.cpp
        Module.cpp
        ../helpers/utils.cpp)
```

Code flow

- Enable or disable the plug-in flag in the recipe file.
- Add this flag into the main CMakeLists.txt file present in the rdkservice.
- It will invoke CMakeLists file present in the <plugin name>/ (eg: PluginTemplate/CMakeLists.txt).
- When this file started to execute it finds dependencies, packages. it compiled and generate .so file.

To add plugin in rdkservices CMakeLists.txt

In rdkservices directory open CmakeLists.txt :

```
$ vi CmakeLists.txt
```

add these lines (by default its disabled) :

```
    if(PLUGIN_PLUGINTEMPLATE)
            add_subdirectory(PluginTemplate)
    endif()
```

it will invoke your CMakeLists.txt file present in your plugin directory.

Compilation and Install

To include ***plugintemplate*** plugin in build sequence, Open rdkservices recipe file and add below line. By default; its configured to be disabled while building `rdkservices.`

```
$ vi meta-rdk-video/recipes-extended/rdkservices/rdkservices_git.bb
```

```
PACKAGECONFIG[plugintemplate]      = " -DPLUGIN_PLUGINTEMPLATE=OFF,-DPLUGIN_PLUGINTEMPLATE=ON, "
```

To include the plugin in rdkservises build; add the same in packageconfig in rdkservices recipe:

```
PACKAGECONFIG += " plugintemplate"
```

to compile and install in build directory :

```
$ bitbake -c compile -f rdkservices
```

once build complete copy .json, .so file into raspberry Pi.

Copy the Plugin.json (eg: PluginTemplate.json) file to “/etc/WPEFramework/plugins” in raspberry Pi

Copy the plugin library (libWPEFrameworkPluginTemplate.so) to “/usr/lib/wpeframework/plugins”

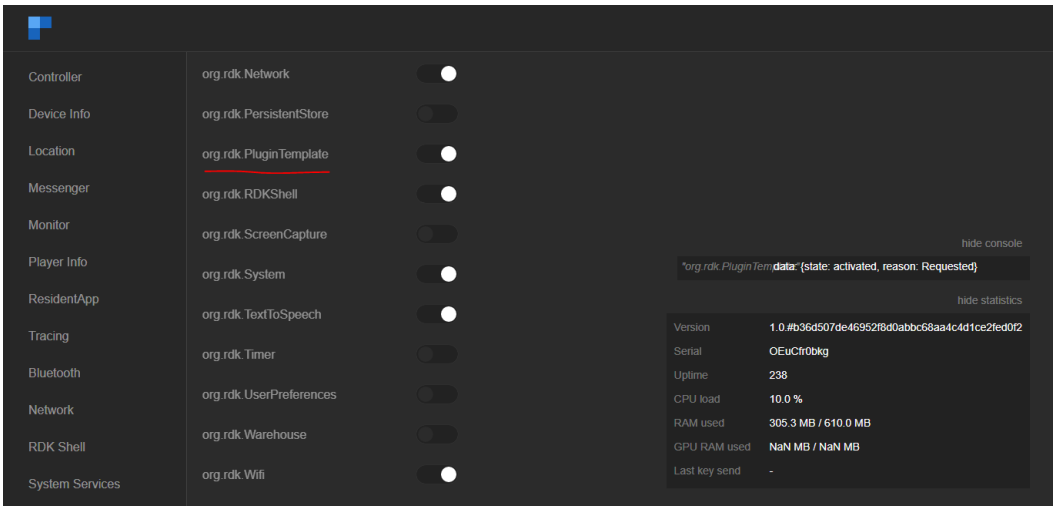
so that the controller plugin identify it and list it in the WebUI (controller UI).

Controller UI

Controller UI is a web UI that can be launched from a host machine's (machine under the same network where Rpi resides) browser. This UI can be loaded with the Rpi box's IP address with Thunder's port number configured (here). RDKServices uses 9998 as port.

```
URL: http://<IP address of the Target device>:9998
```

Defalut page of Controller UI shall be loaded on web-browser and that will be of Controller tab. Controller tab allows all available plugins to be enabled or disabled.



PluginTemplate JSON RPC command

Each RDK Service can be validated through JSON RPC Commands through HTTP. It has a request and response in JSN format.

Note: the argument is case sensitive.

"callsign":"org.rdk.PluginTemplate"

Function	Request	Response	Remarks
Activate controller	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{ "jsonrpc":"2.0", "id":3, "method":"Controller.1.activate", "params":{"callsign":"org.rdk.PluginTemplate"} }'	{"jsonrpc":"2.0","id":3,"result":{"success":true}}	
Deactivate controller	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{ "jsonrpc":"2.0", "id":3, "method":"Controller.1.deactivate", "params":{"callsign":"org.rdk.PluginTemplate"} }'	{"jsonrpc":"2.0","id":3,"result":{"success":true}}	
getPluginStatus	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.getPluginTemplateStatus"}'	{"jsonrpc":"2.0","id":3,"result":{"connection status from plugin":["CONNECTED"],"success":true}}	
getPluginTemplateList	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.getPluginTemplateList"}'	{"jsonrpc":"2.0","id":3,"result":{"Supported plugin list":["plug-A","plug-B","plug-C","plug-D","plug-E"],"success":true}}	
getPluginTemplateInfo	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.getPluginTemplateInfo", "params":{"plugin_name":"plug-A"}}'	{"jsonrpc":"2.0","id":3,"result":{"supportedTvResolutions":["xyz-plugin","no:430HT5"],"success":true}}root@raspberrypi-rdk-mc:~#	
event API when hdmi connected	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.getConnectedVideoDisplays"}'	{"jsonrpc":"2.0","id":3,"result":{"connectedVideoDisplays":["HDMI0"],"success":true}}root@raspberrypi-rdk-mc:~#	
event API when hdmi not connected	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.getConnectedVideoDisplays"}'	{"jsonrpc":"2.0","id":3,"result":{"connectedVideoDisplays":[],"success":true}} root@raspberrypi-rdk-mc:~#	

```
root@raspberrypi-rdk-mc:~# curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --
request POST --data '{ "jsonrpc":"2.0", "id":3, "method":"Controller.1.activate", "params":{"callsign":"org.
rdk.PluginTemplate"} }'
{"jsonrpc":"2.0","id":3,"result":{"success":true}}root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~# curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --
request POST --data '{ "jsonrpc":"2.0", "id":3, "method":"Controller.1.deactivate", "params":{"callsign":"
org.rdk.PluginTemplate"} }'
{"jsonrpc":"2.0","id":3,"result":{"success":true}}root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~# curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --
request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.getPluginTemplateStatus"}'
{"jsonrpc":"2.0","id":3,"result":{"connection status from plugin":["CONNECTED"],"success":true}}
root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~# curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --
request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.getPluginTemplateList"}'
{"jsonrpc":"2.0","id":3,"result":{"Supported plugin list":["plug-A","plug-B","plug-C","plug-D","plug-E"],"
success":true}}root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~# curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --
request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.getPluginTemplateInfo",
"params":{"plugin_name":"plug-A"}}'
{"jsonrpc":"2.0","id":3,"result":{"supportedTvResolutions":["xyz-plugin","no:430HT5"],"success":true}}
root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~# curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --
request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.
getConnectedVideoDisplays"}'
{"jsonrpc":"2.0","id":3,"result":{"connectedVideoDisplays":["HDMI0"],"success":true}}root@raspberrypi-rdk-mc:
~#
root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~# curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --
request POST --data '{"jsonrpc":"2.0", "id":3, "method":"org.rdk.PluginTemplate.1.
getConnectedVideoDisplays"}'
{"jsonrpc":"2.0","id":3,"result":{"connectedVideoDisplays":[],"success":true}}root@raspberrypi-rdk-mc:~#
root@raspberrypi-rdk-mc:~#
```

OUT OF PROCESS Plugin

Here the plugin is developed as out of process, which runs as a separate thread from WPEFramework. Services each other or a particular service can be COMRPC (for communication between plugins) or JSONRPC (for external communication). it has a web-based controller UI.

Inside PluginName directory

```
OutOfProcessPlugin/
  CMakeLists.txt
  OutOfProcessPlugin.config
  OutOfProcessPlugin.cpp
  OutOfProcessPlugin.h

  OutOfProcessPluginJsonRpc.cpp
  OutOfProcessPlugin.json
  Module.cpp
  Module.h
  OutOfProcessPlugin.md
```

<PluginName>.json

This file contains the plugin's information like schema, information and interface json file. Here the outofprocess will be true, which indicates that the plugin run as a seperate process.

eg: OutOfProcessPlugin.json

```
{
 "locator":"libWPEFrameworkOutOfProcessPlugin.so",
 "classname":"OutOfProcessPlugin",
 "precondition":[
  "Platform"
 ],
 "autostart":true,
 "configuration":{
  "root":{
   "outofprocess":true
  }
 }
}
```

<PluginName>.config

.config files are files used to configure the parameters and initial settings for some computer programs.

Here outofprocess is set to true, to make plugin as out of process plugin.

```
set (autostart true)
set (preconditions Platform)
map()
    kv(outofprocess true)
end()
ans(rootobject)
```

<PluginName>.h

Declare the plugin class in this which should contain all the structures, variables, and methods which are needed for plugin implementation.

```
namespace WPEFramework {

 namespace Plugin {

  class PluginName : public PluginHost::IPlugin, public PluginHost::IWeb, public PluginHost::JSONRPC {

  public:

    PluginName()

     : _skipURL(0)

     , _service(nullptr)

     , _subSystem(nullptr)

    {

     RegisterAll();

    }

    virtual ~PluginName()

    {

     UnregisterAll();

    }

  }

  ----------------------------------------

  ----------------------------------------

 }

}
```

<PluginName>.cpp

This class does contain all the definitions for the methods declared in the Plugin.h and those definitions should be defined inside the below namespace.

The plugin should register using service registration MACRO as declared below :

```
namespace WPEFramework {

  namespace Plugin {

   SERVICE_REGISTRATION(Plugin, 1, 0);

   ----------------------------------------

   ----------------------------------------

   ----------------------------------------

  }

 }
```

To initialize and deinitialize or activate or deactivate handler for the plugin services :

```
const string OutOfProcessPlugin::Initialize(PluginHost::IShell* /* service */)

    {

    //shared pointer initialized

    //initialize external library

        LOGINFO();

        return (string());

        }

void OutOfProcessPlugin::Deinitialize(PluginHost::IShell* /* service */)

        {

    //shared pointer deinitialized

    //deinitialize external library

        LOGINFO();

        }
```

Process handler plugin services to receive request and sent responses based on the services :

```
Core::ProxyType<Web::Response> OutOfProcessPlugin::Process(const Web::Request &request)

{

  Core::ProxyType<Web::Response> result(PluginHost::IFactories::Instance().Response());



  //Handle the service request and send the responses

  -----------------

  -----------------

  return result;

}
```

eg: refer OutOfProcessPlugin.cpp

<PluginNameJsonRpc>.cpp>

The PluginNameJsonRpc file contains the registration for methods and properties which are declared in PluginName.h

```
namespace WPEFramework {

    namespace Plugin {

          //registration

         void OutOfProcessPlugin::RegisterAll()

          {

               // methods and properties declared in Plugin.h are registered here

                  -------------------

                  -------------------

          }

         void OutOfProcessPlugin::UnregisterAll()

         {

         }

    }

}
```

CMakeLists.txt

Using the CMake utility this file contains the task needed to be done to make a plug-in. Also contains packages, libraries needed to compile, its path, and other plugin-in configuration option.

This file contains a set of directives and instructions describing the project's source files and targets (executable, library, or both).

```
set(PLUGIN_NAME OutOfProcessPlugin)                          # to set a environment variable set(<variable>
<value>)

set(MODULE_NAME ${NAMESPACE}${PLUGIN_NAME})

find_package(${NAMESPACE}Plugins REQUIRED)              # to Finds and loads settings from an external
project.



#Adds a library target called <name> to be built from the source files listed in the command invocation. The
<name> corresponds to the logical target name and must be globally unique within a project.

add_library(${MODULE_NAME} SHARED

    OutOfProcessPlugin.cpp

    OutOfProcessPluginJsonRpc.cpp

    Module.cpp)
```

The **Code flow, Compilation and Install** steps are similar to the PluginTemplate.

The last step,

Copy the Plugin.json (eg: OutOfProcessPlugin .json) file to “/etc/WPEFramework/plugins” in Raspberry Pi

Copy the plugin library (libWPEFrameworkOutOfProcessPlugin.so) to “/usr/lib/wpeframework/plugins”

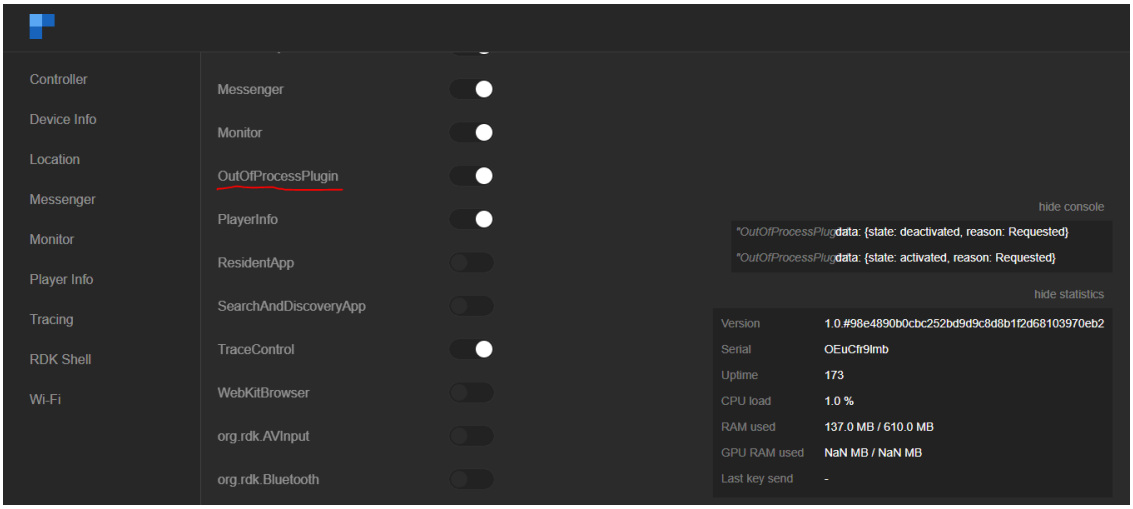
so that the controller plugin identify it and list it in the WebUI (controller UI).

Controller UI

- Inside PluginName directory

Controller UI is a web UI that can be launched from a host machine's (machine under the same network where Rpi resides) browser.

```
URL: http://<IP address of the Target device>:9998
```



OutOfProcessPlugin JSON RPC command

Each RDK Service can be validated through JSON RPC Commands through HTTP. It has a request and response in JSON format.

Note: the argument is case sensitive.

"callsign":"OutOfProcessPlugin"

Function	Request	Response	Remarks
Activate controller	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{ "jsonrpc":"2.0", "id":3, "method":"Controller.1.activate", "params":{"callsign":"OutOfProcessPlugin"} }'	{"jsonrpc":"2.0","id":3," result":{"success":true}}	
Deactivate controller	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{ "jsonrpc":"2.0", "id":3, "method":"Controller.1.deactivate", "params":{"callsign":"OutOfProcessPlugin"} }'	{"jsonrpc":"2.0","id":3," result":{"success":true}}	
Get fps	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{"jsonrpc":"2.0", "id":3, "method":"OutOfProcessPlugin.1.fps"}'	{"jsonrpc":"2.0","id":3," result":32}	
Get plugin id	curl http://127.0.0.1:9998/jsonrpc --header "Content-Type: application/json" --request POST --data '{"jsonrpc":"2.0", "id":3, "method":"OutOfProcessPlugin.1.getpluginid"}'	{"jsonrpc":"2.0","id":3," result":6501}	

Interface with other RDK services

RDK components implemented as Thunder plugins are called as RDKServices. It is developed based on the Thunder (WPE) Framework. Services each other or a particular service can be COMRPC (for communication between plugins) or JSONRPC (for external communication).i.e.

COMRPC is used to communicate between the plugins (out of process) or to communicate for larger data.

JSONRPC is used to fetch/update info to or from plugins externally (most of the plugins provide this in interface, similar to ReST API) also it can be used from applications.

JSONRPC:

For instance please see below Bluetooth plugin's **pair** method with JSONRPC interface.

Events

Event	Description
BluetoothState: PAIRING_CHANGE	Triggers onStatusChanged event when the device gets paired to given device ID.
BluetoothState: PAIRING_FAILED	Triggers onRequestFailed event, when the device is unable to pair.

Parameters

Name	Type	Description
params	object	
params. deviceID	string	ID that is derived from the Bluetooth MAC address. 6 byte MAC value is packed into 8 byte with leading zeros for first 2 bytes

Result

Name	Type	Description
result	object	
result.success	boolean	Whether the request succeeded

Request

```
{
    "jsonrpc": "2.0",
    "id": 42,
    "method": "org.rdk.Bluetooth.1.pair",
    "params": {
        "deviceID": "61579454946360"
    }
}
```

Response

```
{
    "jsonrpc": "2.0",
    "id": 42,
    "result": {
        "success": true
    }
}
```

For more details please refer : https://github.com/rdkcentral/rdkservices/blob/sprint/2107/Bluetooth/doc/BluetoothPlugin.md#pair-method

The corresponding implementation in ThunderJS is given below:

```
   this.thunderJS.call('org.rdk.Bluetooth', 'pair', { deviceID: deviceIDval },

      (err, result) => {

        if (err) {

          Log.info('\n Bluetooth Pair error' + JSON.stringify(err))

        } else {

          Log.info('Pairing success' + JSON.stringify(result))

        }

      }

    )
```

To know more about how to implement JSONRPC inerface using ThunderJS in JS environment please see https://github.com/rdkcentral/ThunderJS/blob/master/readme.md

Interface with Lightning apps

ThunderJS is used to make easy to make API calls to Thunder (WPEframework) over a Websocket connection. ThunderJS can also be used to listen to (and act upon) notifications broadcasted by Thunder. ThunderJS is an *isomorphic* library, which means it can be used in a browser environment as well as a NodeJS environment.

Lightning is a Javascript TV app development framework based on NodeJS environment. So ThunderJS can be easily integrated to the Lightning apps.

Adding ThunderJS dependency to package.json

ThunderJS dependencies can be added manually to the lightning projects by adding '"ThunderJS": "github:rdkcentral/ThunderJS",' to the *package.json* under *'devDependencies'* and then run *'npm install'*.

OR

ThunderJS can be installed into your project via *NPM* command. Then the package.json will be updated with the thunder dependency "npm install github:rdkcentral/ThunderJS"

Snippet of package.json is given below .To use the ES6 syntax, we need add the Babel dependency also.

```
  }
},
"devDependencies": {
 "@babel/core": "^7.7.2",
 "ThunderJS": "github:rdkcentral/ThunderJS",
 "babel-eslint": "^10.0.3",
 "dashjs": "^3.1.3",
 "eslint": "^6.6.0",
 "eslint-config-prettier": "^6.7.0",
 "eslint-plugin-prettier": "3.1.1",
 "hls.js": "^0.13.2",
 "husky": "^3.1.0",
 "lint-staged": "^9.4.3"
 "prettier": "^1.19.1"
  }
}
```

Import ThunderJS dependency in Lightning script

Next you can import the ThunderJS dependency into your own script like given below.

import ThunderJS from 'ThunderJS',

Initializing the library

The library can be initialised by passing the IP, port and other parameters mentioned below

```
const config = {
 host: '192.168.1.100', // defaults to localhost,
 port: 2020, // defaults to 80  endpoint: '/api', // defaults to '/jsonrpc'
 protocol: 'wss://', // defaults to 'ws://'
 subprotocols: 'notification', // WebSocket sub-protocols, defaults to 'notification'
}
 const thunderJS = ThunderJS(config)
```

Example:

```
this.config = {
        host: '127.0.0.1',
        port:'9998'
}
try {
        this.thunderJS = ThunderJS(this.config)
}        catch (err) {
        Log.error('Error in initialising the Thunder JS' , err)
```

Making API Calls and read results

The library supports 2 ways of making API calls, depending on your coding style preferences.

Option 1 - Argument based

Option 1 - Argument based

```
const plugin = 'DeviceInfo'

const method = 'systeminfo'

const params = {

  foo: 'bar'

}

thunderJS.call(plugin, method, params)
```

Option 2 – Object based

```
const params = {
  foo: 'bar'
}
thunderJS.DeviceInfo.systeminfo(params)
```

The result can be processed in two ways also

Option 1 - Promise based

```
thunderJS.DeviceInfo.systeminfo()
  .then(result => {
    console.log('Success', result)
  }).catch(err => {
    console.error('Error', err)
  })
```

Option 2 - Callback based

```
thunderJS.DeviceInfo.systeminfo((err, result) => {
  if(err) {
    console.error('Error', err)
  }
  else {
    console.log('Success!', result)
  }
})
```

Example for Argument based call for Bluetooth plugin is given below where the result is processed in callback based method

Bluetooth plugin pair method - https://github.com/rdkcentral/rdkservices/blob/sprint/2107/Bluetooth/doc/BluetoothPlugin.md#pair-method

Corresponding implementation in ThunderJS

```
  this.thunderJS.call('org.rdk.Bluetooth', 'pair', { deviceID: deviceIDval },

     (err, result) => {

       if (err) {

         Log.info('\n Bluetooth Pair error' + JSON.stringify(err))

       } else {

         Log.info('Pairing success' + JSON.stringify(result))

       }

     }

   )
```

Notifications

Thunder (WPEframework) broadcasts notifications when events ocur in the system. However it will only broadcast those events that the client has subscribed to.

ThunderJS makes it easy to subscribe to specific events, and execute a *callback-function* upon every notification of each event.

Example for notification event subscriptions is given below.

Bluetooth onDiscovered event - https://github.com/rdkcentral/rdkservices/blob/sprint/2107/Bluetooth/doc/BluetoothPlugin.md#ondiscovereddevice-event

Corresponding implementation in the Lightning App

```
/**
* Event listener to listen to device discovered
*/
this.thunderJS.on('org.rdk.Bluetooth','onDiscoveredDevice',notification =>{
        Log.info('<<Device discovered event>>'+JSON.stringify(notification))
})


/**
```