Wi-Fi Subsystem

RDK Wi-Fi Sub-system overview

In this section we will get an overview on the kind of Wifi support offered by RDK, how WPA supplicant is playing an important role on managing Wifi Driver, event notification and communication to different applications

In RDK we support integrated Wi-Fi chips as well as USB based Wi-Fi adapters. Implementation differences between on board Wi-Fi & USB Wi-Fi adapter are abstracted from Upper layers i.e. application doesn't know what kind of Wi-Fi device or connection they are accessing.

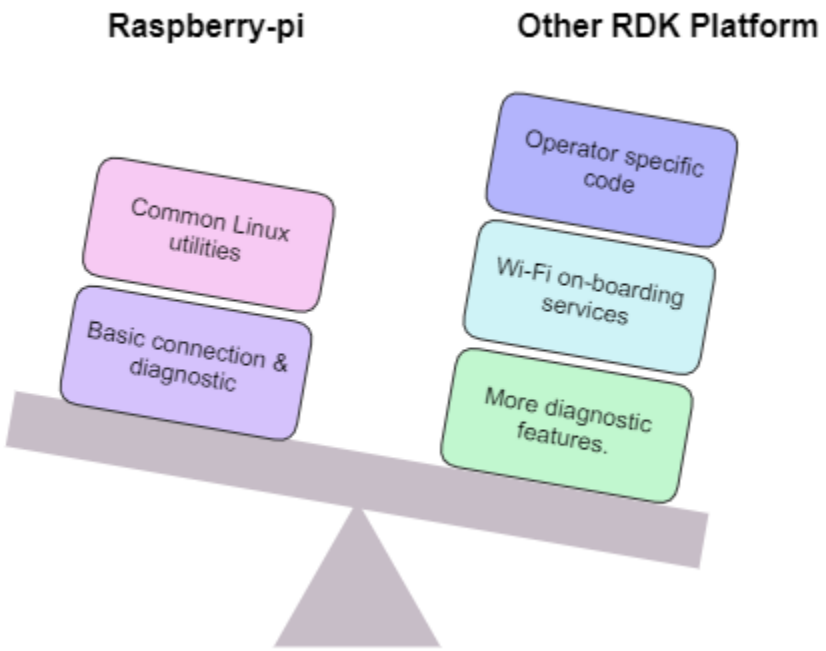
RDK Wifi uses wpa_supplicant wireless daemon for connection management with the Wi-Fi driver. wpa_supplicant is designed to be a "daemon" that runs in the background and acts as the backend component controlling the wireless connection. wpa_supplicant also offers a control and monitoring interface to handle different wireless commands. Wi-Fi subsystem uses generic drivers such as NL80211 brings in wide range of vendor equipment's under coverage.

RDK Wi-Fi stack extensively uses commonly available Linux wireless utilities which brings most of the USB based and on-chip wireless equipment under our coverage. It provides support for diagnostics and connection management from remote and native applications. It uses IARM, which is a Linux D-BUS based communication protocol for managing Wi-Fi event notification and communication across different applications.

WiFi is a new features that is added in RDK video community devices. Recently Wi-Fi features is added to a Hardware Development (Raspberry Pi) video platform. As of now feature wise almost covered the client and access point related things, and we are continuing to work on remote management and diagnostics functionalities to be included into the Wi-Fi code.

Where does Raspberry-Pi Wi-Fi stand?

1.Basic Feature wise R-PI Wi-Fi is almost at the same level as that of other RDK platforms.



Main difference that we can derive are:

In RPI:

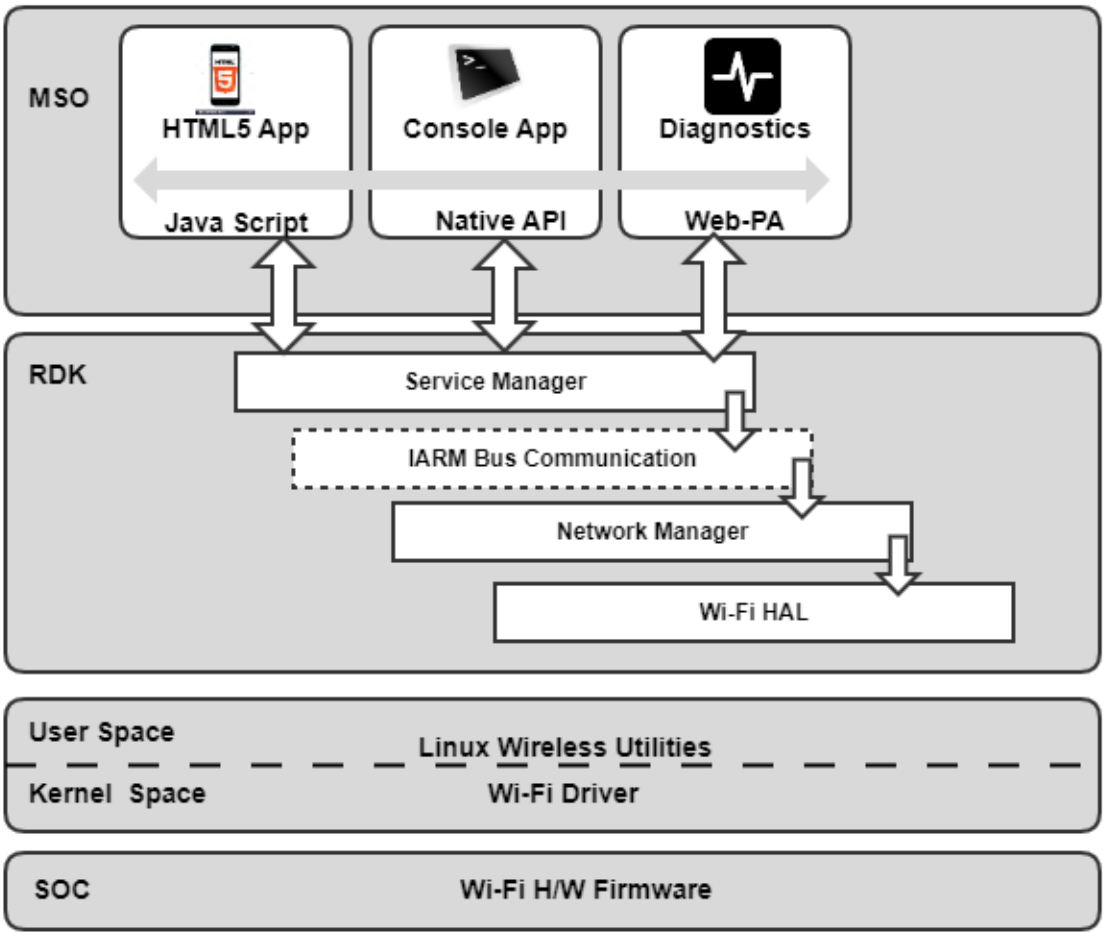
- Raspberry-Pi is light weight and the abstraction layers are mainly written around common linux utilities.
- In R-PI we have a partial set of TR-181 APIs implemented which is needed for basic debugging.

In Other RDK Platform:

- In case of Other commercial RDK platforms, some proprietary code is used along with generic wireless code.
- Network manager can facilitate on-boarding process through a service called as lostfound.
- Also more OEM/SoC calls are used to expose more private features.

Wi-Fi Architecture

In this section, we will get the details on the architecture, layers, and how different RDK components interact with the wifi driver to enable Core wifi functionality.



Application:
In top of the eco-system we have wide range of application which requires wireless network access. This may be a cloud based UI application, a diagnostics webpage or a console application such as test automation kit which will be required to verify readiness of a new RDK box with respect to different component features. This can be a HTML based webpage or a native console application requesting Wi-Fi functionalities.

Service manager:
Service manager is the contact point between external applications and native RDK. It is present in RDK as a library which when plugged in to a browser such as WPE or Qt enhances its capability to make communication from web applications to native RDK components through Java script. This exposes Java Script as well as native APIs, have decision making ability to route the request to appropriate RDK component.

IARM Message Bus:
RDK provides a common message and event notification mechanism known as IARM which passes the calls from upper layer i.e. service manager to actual network manager. A D-Bus based event and messaging mechanism propagates requests from application layer to lower layers. Sends event notification from Lower (network or HAL) layer to application layer.

Wi-Fi Network Manager:
Wi-Fi network manager is a daemon which handles network states and network interfaces. It handles Wireless initialization and management. This maintains the Wi-Fi state machine, initializes Wi-Fi subsystem & manages connection & disconnection events.

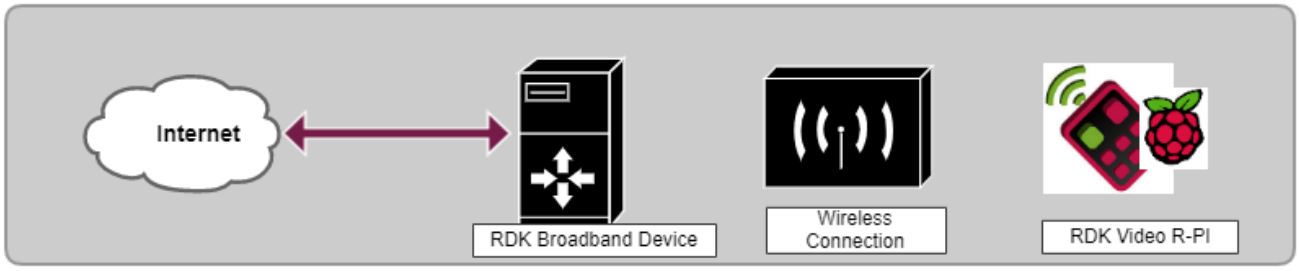
Wi-Fi generic HAL:
It is an Abstraction of Wireless driver calls and various linux wireless utilities to present a set of APIs for common wireless operation. It provides a set of APIs as per RDK specification for connection, management & diagnostic related activities. Abstracts implementation details and SoC dependencies from network management layer.

Linux wireless utility:
Tools such as wpa_supplicant, wereless-tools, net-link library, etc. which provides support for most of the common Wi-Fi chips.

Driver & Firmware:
Provided by SoC/OEM manufacturer. The kernel space driver and firmware binaries will be provided by Wi-FI SOC or OEM and it should be present in the defined path in the RDK box.

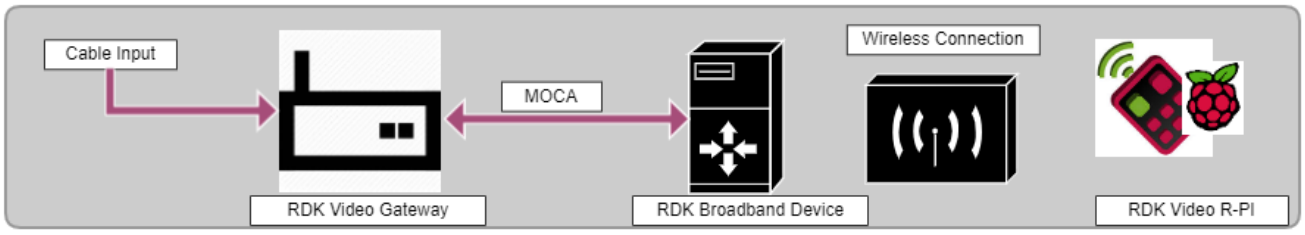
Use Cases

Let us see in what types of network topology one wireless enabled RDK box can operate. We can see two use cases here,
In first case we will see a straight forward network where we will have an IP headend for TV channels, VOD, etc.



- In the above picture, the data will be received by a RDK Broadband device which will be our entry point to the home network. This will act as a video gateway and an internet router also. The network connectivity from RDK broadband device to IP headend will be through a high speed network such as fiber optics, etc.
- From the RDK broadband device, all clients in a home network will receive internet and video data through the wireless network. The RDK broadband device will be the access point and video devices such as Raspberry Pi will act as wifi clients.

In second use case, we wanted to show how a legacy network can be extended to operate with wifi enable devices,
In the figure, we use a RDK Video gateway to receive data from Cable media and relay to a RDK broadband device through MoCA network. In this network topology,



RDK client devices will be able to access the QAM channel from the Video gateway and internet through the broadband device. Video and internet data will be received by the Broadband Device and sent to Wireless clients.

Wi-Fi Service Manager

Service Manager can be used for a wide range of services but in this context we wlll focus on the wifi service only.

- ServiceManager provides a uniform mechanism for discovering and consuming Wifi service on a target device.
- It is a Bridge between browser based application and the native RDK.
- It Provides capability for applications to use wifi services from other applications such as HTML 5, Java scripts, etc.
- All the service registrations are received by Service Manager and after registration, it announces the service availability to interested applications.
- It is one of the well-known facility for cloud-based applications to gain access to device vended functionality.
- Here the advantage is , applications need not know which other component are providing which type of services.
- Service manager provides some callback method which is basically for handling asynchronous events that is sent through other applications.
- for example events such as connecting to AP or disconnected from AP are generated by wifi Manager and sent to service manager.

Some of major API details are provided which is available under the header include/services/wifimanagerservice.h. Generally in case of each service, there are two common API for registering and un-registering particular service. When we register a service, we are making it available to the other application. And when we un-register it will no longer be accessible to the upper layer application.

Refer to Wifi Manager APIs for a complete list of APIs provided by Wifi Service manager.

Wi-Fi Network Manager

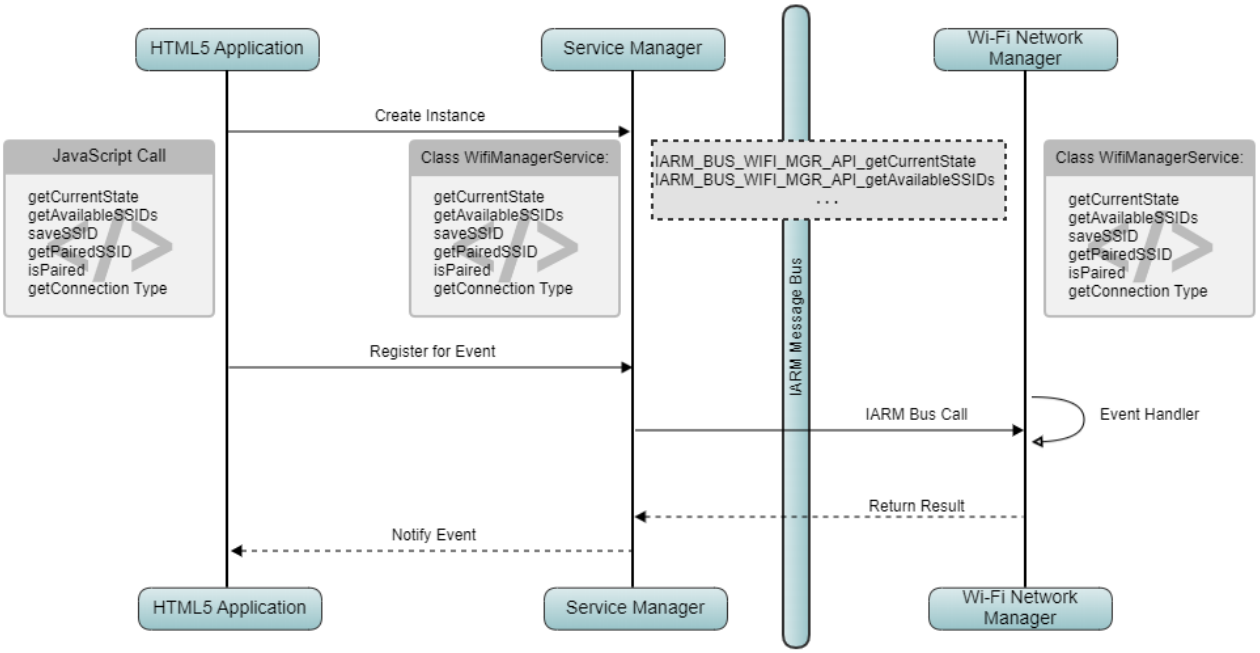
Wifi- network manager, which takes several responsibility for managing Wifi in Video device

- It dynamically detect the Network interfaces, send notifications to subscribers
- Supports diagnostics both local and cloud based applications
- Setup and update the network routes based on priorities, wifi connection status, availability, etc
- It also Listen to home networking updates to determine the routes for which communication to be established in outside world
- It provide Modular approach to add/remove new network interface types and also provide support for configuring routing rules

Refer to netsrvmgr for more detail such as APIs, Events and Error codes.

Wi-Fi Communication Workflow

In the below sequence diagram, we will see how an HTML application will interact with Wifi manager through Service manager. We can see that for each functionality we have an uniform API name across different component, which enable simplicity in development and the set off APIs can be mapped easily. For example, If we take one API that is getConnectionType(), which we can use to get the active interface type which may be Wifi or LAN interface.



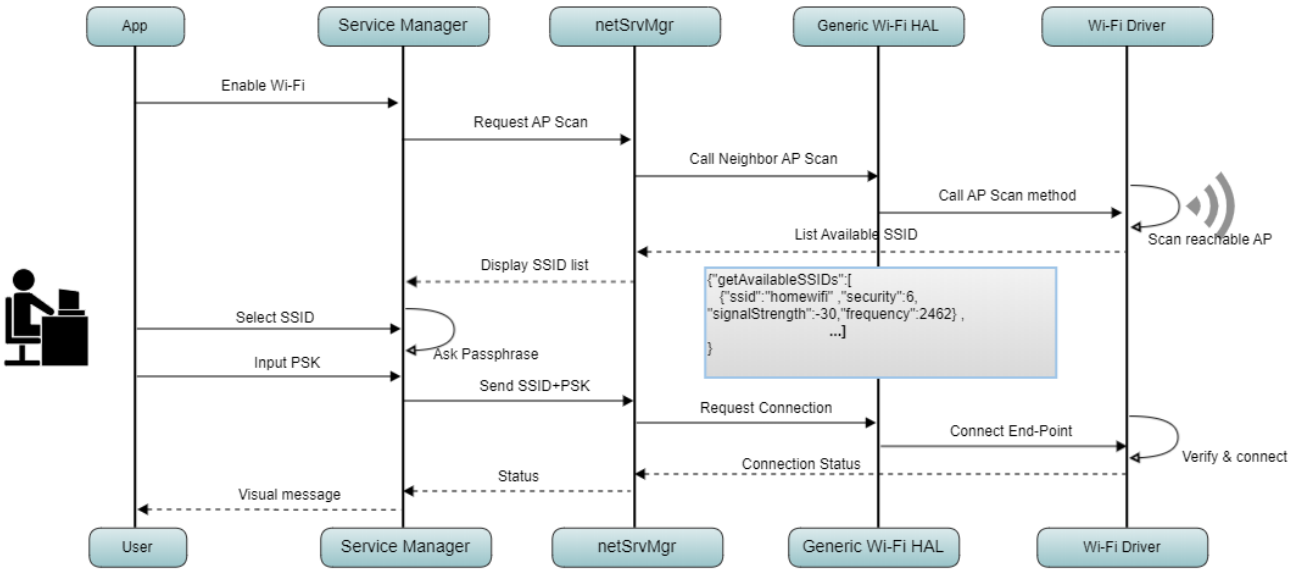
In first place, we will have Java Script functions registered with Service manager. Then Service manager will have an internal implementation for that API with error handling and managing result.
When the call is propagated, to the actual handler daemon (Wifi manager) through message bus, it will be translated to a IARM RPC call.

On the receiving side, Wifi manager will invoke actual Wi-Fi HAL API to get the result.So, we have seen a similar set of API are presented by different components to achieve the task.

An example, we have an user interface application where we want to show an icon based on whether Wi-Fi network is up. In that case we can take the help of service manager API to get the current state. Here service manager will internally call an RPC call with the Wifi manager to invoke the specific function. Then the wifi manager will invoke the actual HAL API to retrieve all the details and returns to the caller in a JSON formatted string. This status string will be then return to the application for displaying the appropriate icon based on UP/DN status.

Use Case: AP Discovery & Establishing Wi-Fi connection with AP

Here we will walkthrough the different RDK component involved in the process and starting from Application which is the imitator, ending with Wi-Fi HAL which is the provider of the functionality. We will also see the messaging formats in which the RDK components communicate.



Initiating the connection: Here application may need to enable the WiFi if required. In which case he has call the initialization routine and network manager will handle the further operation

Getting a list of connection: When the WiFi system is up, application can request for a scan operation and then the neighbor scan API will be called by network manager. Which intern calls a Wi-Fi HAL API and return the result in JSON formatted string.

Selecting a preferred network: User can select a network from the list of Available AP.

Providing credential for the preferred network: Application has to handle the prompts for passphrase, network security Key, etc. and initiate the connection process.

Validation of the credential: Wifi manager will take the parameter and select the appropriate method to pass credential to WPA supplicant via an Wi-Fi HAL function. WPA supplicant will perform the actual sequence of operation to validate and establish the connection which will be an asynchronous process. i.e whenever the connection is established an appropriate event will be generated and notify to the caller.

Results: Application when receive the connection establish/failed event can show a visual confirmation.

Wi-Fi Integration Requirement

For integrating Wifi feature in a new RDK platform, following basic requirement can be kept in mind,

- First we need to have the OEM or SOC provider driver and firmware available with us so that we can integrate with our platform.
- There are Generic linux wireless packages need to be integrated with the platform.
- Most importantly, all the compatibility issues regarding Wifi driver and platform need to be addressed before hand, so that there is no issues regarding performance or connection glitches.
- We will need build support files for integration.
- A new HAL has to be written for the platform.

1.Enable OEM/SoC recipes
. To build Wi-Fi driver & firmware installation

2.Select generic Linux wireless utilities
. wireless-tools
.wpa-supplicant
.net-link library etc.

3.RDK provides a reference Wi-Fi HAL
.wifi-hal-noop

4.Generic Wi-Fi HAL code for the new platform.

5.New HAL recipe to define build rules.

6.Customize virtual/wifi-hal to set preferred HAL

7.Machine configuration & image recipes

RDK Wifi Specification mentions, all communication from network manager to the Wifi Driver has to be through WPA supplicant. So the basic requirement is to add WPA supplicant and its related packages such as netlink library and wireless tools. When all the dependency are added to the platform, we have to write a Wi-Fi HAL customized in our platform. Then we have to add the Wifi support in build framework via adding appropriate packages in image recipe as well as machine configuration file.

Code Walk-through in RPI Platform

Code Walk-through: Folder structure

Generic (Reference) Wi-Fi code : rdk/components/generic/wifi/

```
rdk
`-- components
    `-- generic
        `-- wifi
            |-- cfg
            |   `-- TODO
            |-- configure.ac
            |-- CONTRIBUTING.md
            |-- COPYING -> LICENSE
            |-- include
            |   |-- wifi_ap_hal.h
            |   |-- wifi_client_hal.h
            |   `-- wifi_common_hal.h
            |-- LICENSE
            |-- Makefile.am
            |-- NOTICE
            |-- README
            |-- src
            |   |-- Makefile.am
            |   |-- wifi_client_hal.c
            |   `-- wifi_common_hal.c
            `-- test
                |-- Makefile.am
                `-- testwifi.c
```

Device specific Wi-Fi code : devices/raspberrypi/wifi/

```
devices/
|-- intel-x86-pc
`-- raspberrypi
    |-- devicesettings
    |-- dvr
    |-- iarmmgrs
    |-- mediaframework
    |-- mfrlibs
    |-- sysint
    `-- wifi
        |-- cfg
        |-- configure.ac
        |-- COPYING -> LICENSE
        |-- LICENSE
        |-- Makefile.am
        |-- NOTICE
        |-- README
        `-- src
            |-- Makefile.am
            |-- wifi_client_hal.c
            |-- wifi_common_hal.c
            |-- wifi_hal_priv.h
            `-- wifi_hal_test.c
```

Looking at the below folder structure of the wifi HAL code in reference wifi & device specific wifi, We can see both follow similar naming conventions. We have 2 common files: wifi_common_hal.c and wifi_client_hal.c

In wifi_common_hal.c:

- Here we have defined most of the diagnostic APIs and utility functions.

In wifi_client_hal.c:

- It Mainly defines the internal state management functions, threads for monitoring the events coming from wpa_supplicant and connection related functions.
- It also defines callback functions for sending back the state changes and error events to wifi manager
- In generic wifi HAL, we have 3 header functions for defining AP, client & common APIs.
- In Raspberry Pi, the device specific HAL has dependency of WPA client library and libnl.

Code Walk-through: Wi-Fi HAL Recipe

This is a simple recipe which defines build rules for WIFI HAL library code. Here we can see that after defining some common stuff, it defines 2 environment variables i.e. PROVIDES , RPROVIDES_PN, both of which are assigned to virtual/wifi-hal. We can also see that wpa_supplicant and wifi-hal-headers as dependencies in the below code.

$ vi meta-cmf-raspberrypi/recipes-extended/wifi-client-hal/wifi-client-hal-raspberrypi_0.1.bb

```
# ============================================================================
# RDK MANAGEMENT, LLC CONFIDENTIAL AND PROPRIETARY
# ============================================================================
# This file (and its contents) are the intellectual property of RDK Management, LLC.
# It may not be used, copied, distributed or otherwise  disclosed in whole or in
# part without the express written permission of RDK Management, LLC.
# ============================================================================
# Copyright (c) 2017 RDK Management, LLC. All rights reserved.
# ============================================================================
#

SUMMARY = "WiFi Client RDK HAL interface layer library for Raspberry Pi."
SECTION = "console/utils"

LICENSE = "Apache-2.0"
LIC_FILES_CHKSUM = "file://LICENSE;md5=175792518e4ac015ab6696d16c4f607e"
PV = "${RDK_RELEASE}+git${SRCPV}"

PROVIDES = "virtual/wifi-hal"
RPROVIDES_${PN} = "virtual/wifi-hal"

SRC_URI = "${CMF_GIT_ROOT}/devices/raspberrypi/wifi;protocol=${RDK_GIT_PROTOCOL};branch=${RDK_GIT_BRANCH}"
SRCREV = "${AUTOREV}"

S = "${WORKDIR}/git"

DEPENDS="wifi-hal-headers wpa-supplicant"

EXTRA_OECONF += "--disable-static --disable-silent-rules"
EXTRA_OECONF_append = " --enable-wpa-supplicant=yes"

inherit autotools pkgconfig
```

Code Walk-through: Network Manager

Folder structure

```
rdk/components/generic/netsrvmgr/src/services/wifi/
|-- include
|   |-- wifiHalUtiles.h
|   |-- wifiSrvMgr.h
|   `-- wifiSrvMgrIarmIf.h
|-- Makefile.am
`-- src
    |-- wifiHalUtiles.cpp
    `-- wifiSrvMgr.cpp

2 directories, 6 files
```

$vi netsrvmgr/src/services/wifi/src/wifiSrvMgr.cpp

```
int  WiFiNetworkMgr::Start()
{
#ifdef ENABLE_IARM

    IARM_Bus_RegisterCall(IARM_BUS_WIFI_MGR_API_getAvailableSSIDs, getAvailableSSIDs);
    IARM_Bus_RegisterCall(IARM_BUS_WIFI_MGR_API_getCurrentState, getCurrentState);
#endif
#ifdef USE_RDK_WIFI_HAL
    monitor_WiFiStatus();

    if(wifi_init() == RETURN_OK) {
        RDK_LOG( RDK_LOG_INFO, LOG_NMGR, "[%s:%s:%d] Successfully wifi_init() done. \n",
    } else  {
        RDK_LOG( RDK_LOG_ERROR, LOG_NMGR, "[%s:%s:%d] Failed in wifi_init(). \n", MODULE_
    }

    /*Register connect and disconnect call back */
    wifi_connectEndpoint_callback_register(wifi_connect_callback);
    wifi_disconnectEndpoint_callback_register(wifi_disconnect_callback);
#endif
}
```

Wi-Fi HAL APIs

Doxygen documentation of Wifi Hal Api's are in-progress. Will provide the link here once it is completed.