

For OEM

This page provides the step by step procedures for OEMs to adopt RDK. OEM layer sits on top of SoC layer and later adds support for software for boot-up, image updates, and APIs to handle custom drivers. These could be specializations to the generic or SoC components or complementary software components provided by the OEM to create a fully functional set-top device.

On this Page:

[Before You Begin](#)[Bringing up RDK by OEM- Approach](#)[Procedure for OEM porting of RDK](#)

Before You Begin

RDK License

OEMs are advised to get into an agreement with RDK Management LLC to obtain the free license so as to use the complete RDK Code base in their platform. More details about license is available at <https://rdkcentral.com/licenses/> . Please email info@rdkcentral.com if you have additional questions about licenses or membership

Bringing up RDK by OEM- Approach

This section will detail the recommended step by step procedure of adopting RDK by OEM.

Product Specifications

The first step to get a fully functional product is to define the product features and see if they meet the standard requirements. See [here](#) to know what are all the features available in RDK-V and can implement based on your requirement. OEM can cross check the expected features/specifications with the capabilities of the SoC platform being used and can finalize the features supported by the product.

RDKM On-boarding

RDKM provides a collaboration zone facility for OEMs to facilitate easier engineering of RDK based devices. The collaboration zone will help OEMs to work with SoCs, RDKM and any 3rd party along with a common space to develop & integrate, manage and verify the device. The zone includes facilities for Code management, a confluence based RDK Wiki for knowledge management & sharing, a JIRA for tracking activity progress, issues as well as to manage the activities, a test setup to validate devices. The access restrictions implemented will help the collaboration zone to be accessible only for the authorized personnel thereby guarding any sensitive information related to SoC/OEM/Third party.

Roles & Responsibilities

A table explaining the roles & responsibilities of OEM & RDKM in the collaboration zone is given below:

#	Activity	Owner	Remarks
1	OEM Collaboration zone creation	RDKM	RDKM will setup Collaboration space, access restrictions
2	JIRA Project creation	OEM	JIRA project. OEM will be the owner for the JIRA project.
3	SoC / OEM meta -layer creation in collaboration zone	OEM/RDKM	RDKM will create the space and SoC/OEM push the code changes
4	Device specific HAL repo creation	OEM	Create necessary device specific HAL implementation for porting RDK into Accelerator
5	Share SoC SDK Artifacts	RDKM	Which SDK version to be used. RDKM will support the integration with OEM libraries
6	Manifest creation	OEM	Manifest for building the accelerator
7	Remote control integration (only for RDK licensee)	RDKM/OEM	RDKM approved RCU's are enabled by default
8	UI/UX (only for RDK licensee)	RDKM/OEM	Comes with pre-integrated UI's, OEM and RDKM will discuss on the default UI
9	Create Accelerator build from CMF GIT	OEM / RDKM	Both teams work together to build Accelerator from CMF.
10	Provide Devices to RDKM team	OEM	

11	Device flashing instructions / recovery mechanisms	OEM	OEM should share the device flashing instruction.
12	Sanity, Functionality Testing & automation tests	RDKM/OEM	RDK Certification Suite (only for RDK licensee)
13	Monthly release & tagging	OEM	Monthly tagging and release with stakeholders along with test results

How to create a collaboration zone

It is expected that OEM has already obtained a license to work with RDKM (If not, OEM can send a mail to support@rdkcentral.com to start off with the discussions) .

With this user account an INFRA ticket can be raised at <https://jira.rdkcentral.com> to create a collaboration repo. The ticket should contain the details for

- Location of collaboration zone
- Collaboration zone access groups/members

How to create user accounts for OEM members

OEM users can sign up at <https://wiki.rdkcentral.com/signup.action> to create a user account in RDK. For any issues faced, a mail can be sent to support@rdkcentral.com

How to get access to the collaboration zone/repo

An INFRA ticket needs to be raised at <https://jira.rdkcentral.com> with the below details

- Collaboration zone/repo name
- User name and the mail id's of the members to whom the access is needed
- OEM collaboration zone/repo owner name

For any issues faced, a mail can be sent to support@rdkcentral.com

How to create a JIRA project for OEM

An INFRA ticket needs to be raised at <https://jira.rdkcentral.com> to create a JIRA project for OEM. Once approvals are received along with required access restrictions, the project will be created. . For any issues faced, a mail can be sent to support@rdkcentral.com

How to create a Git/ Github repository for meta layers or manifests or HAL layers

To get a Git repository a request needs to be raised to CMF team using the CMFSUPPORT ticket at <https://jira.rdkcentral.com> . Once approvals are received along with required access restrictions, the repo will be created. Any changes in merge permissions can be requested in same ticket. For creating any specific branches in the repo, another ticket in the same CMFSUPPORT can be raised. For any issues faced, a mail can be sent to support@rdkcentral.com .

Once the git repo is created, it can be accessed at <https://code.rdkcentral.com>

How to get access to SoC SDK artifacts

An INFRA ticket needs to be raised at <https://jira.rdkcentral.com> to get access to SDK artifacts. Once approvals are received along with required access restrictions, the access should be in place . For any issues faced, a mail can be sent to support@rdkcentral.com

Product Engineering

Once the product features are decided, the device engineering can be started. OEM needs to decide on the hardware layout that incorporates OEM components to the SoC board. Device will be categorized as Low Profile and High Profile device based on the hardware capabilities. In a low profile device 4k support might be optional but it is expected to have 4k support in high profile device.

Sample Reference Flash Layout (Optional) :

Reference Flash Layout	
flash0.macadr	EMMC flash Data : 1024B
flash0.nvram	EMMC flash Data : 64KB
flash0.recovery	EMMC flash Data : 256MB
flash0.vendor	EMMC flash Data : 128MB
flash0.kernel_a	EMMC flash Data : 128MB
flash0.kernel_b	EMMC flash Data : 128MB
flash0.rootfs_a	EMMC flash Data : 1024MB
flash0.rootfs_b	EMMC flash Data : 1024MB
flash0.rootdata	EMMC flash Data : 2048MB
flash0.rsvd	EMMC flash Data : 10174MB

Device Firmware

OEM can make use of the below details to start developing a Yocto build to engineer the device firmware builds based on RDK Yocto build setup.

Yocto based RDK builds are flexible enough to easily accommodate SoC & OEM changes. The starting point for the Yocto builds are a manifest file. The manifest file is an xml file that contains details of the different open embedded Yocto build layers, meta layers , rdk as well as open source components that needs to be fetched during initial stages (than during bitbake time) as well as the URL locations from where the data can be pulled. A set of sample manifests that can be used as a template for developing OEM specific manifests are given below

#	File	Remarks
1	Device Specific Manifest	This is the starting point of the build and is specific to a device. Usually it contains references to other manifest files which will be having specific set of repos
2	SoC manifest	This manifest contains meta layer details of SoC and , optionally, some SoC specific repos
3	OEM manifest	This manifest contains meta layer details of OEM(which might or might not be generic across OEMs multiple devices) and , optionally, some OEM specific repos
4	OE layers manifest	This manifest has details of the basic Yocto Open Embedded layers
5	RDK-V manifest	This manifest can contain meta layers specific to RDK & RDK-V and , for EXTERNALSRC cases, the RDK & RDK-V component repo details too. It might be supplemented with a conf file too
6	Third party apps	This manifest can contain meta layers related to Premium streaming applications which are chosen by OEM

For more details on getting RDK-V ported to an OEM device, please refer the links [Add New SoC and OEM to RDK](#) , [RDK-V OEM Specific Porting](#) and [Guideline for SoC and OEM Vendors](#)

SoC/OEM meta-layer creation

To match the layered structure of Yocto builds, a OEM specific layer is used to include OEM changes and additions. Use the yocto-layer create sub-command to create a new general layer.

```
$ yocto-layer create mylayer
```

There shall be separate device (machine) configuration file (.conf) for each device for the particular OEM for which the layer is intended for. The general naming terminology for OEM layer is meta-rdk-oem-<oem name>

The device (machine) configuration file shall include corresponding include (.inc) file to get machine configuration details.

Adding the Machine Configuration File for the new OEM

Each meta-* layer should have a conf folder containing the machine configuration we can select during 'source setup-environment' . Optionally it can also have a class/classes folder for keeping information that is useful to share between metadata files.

To add a machine configuration, you need to add a .conf file with details of the device being added to the conf/machine/ file. In the machine conf file the basic machine configuration should be defined.

OEM can have several meta-rdk-<oem> layers containing chip specific implementation.

Generally conf file name will be SoC name. If both OEM and SoC are present then machine configuration should be coming from master i.e. OEM . It means whatever image that gets created comes from meta-rdk-oem

The most important variables to set in this file are as follows :

- Include .inc's ,basically these inc's are from SoC layer and these .inc's are platform dependent.
- Fill the MACHINEOVERRIDES (Eg: If it is media client, give client name)

```
MACHINEOVERRIDES= " "
```

- For particular SoC build , if they are overwriting any kernel version or gcc compiler version or whatever preferred should be defined in machine conf file.

For SoC, The most important variables to set in this conf file are as follows :

```
TARGET_ARCH (e.g. "arm")
PREFERRED_PROVIDER_virtual/kernel (Eg:"linux-<chip set name>")
MACHINE_FEATURES (e.g. "apm screen wifi")
KERNEL_IMAGETYPE (e.g. "uImage")
IMAGE_FSTYPES (e.g. "tar.gz ext4")
```


Basic configuration of building kernel

Standard meta-rdk/recipe-core/images will have different kinds of images which one can use and build.

This recipe-core/images will have bbappend or main image name. If meta-rdk is used(i.e. meta-rdk image) then bbappend of that image client is required

For example,

<image name>.bbappend

 Define "bbappend of images" and "conf file" as required.

Adding Recipe for SoC/OEM

The following kind of recipes can be added to SoC/OEM layer. The recipes shall be grouped as described in "BSP Reference Layer".

- recipes (.bb) to build Kernel
- recipes(.bb) to build SDK
- Kernel patches (SoC/OEM specific - if any)
- SDK patches (SoC/OEM specific - if any)
- Any SoC/OEM specific scripts or files which need to be installed in RF

TDK and RDK Certification Suite Package

RDKM offers an in-house Test & certification suite that facilitates OEMs to get their IP based product certified as RDK Compliant device.

Certification program includes testing which validates the RDK stack on the device with defined test suite called as RDK Certification Test Suite. It is mandatory to go through this program in order to brand user's platform as RDK compliant product.

Certification suite is available at [RDK IP Set-top Product Certification](#)(only for RDK licensee) and for TDK test app please refer [TDK-V Documentation](#).

SDK Releases

Once the RDK bring-up in SoC is completed, the software is delivered to OEM by the SoC vendors. This usually happens in 2 ways:

HAL + SDK binary

In this approach will make use of the RDK Artifactory server. Artifactory server is a Repository Manager that functions as a single access point organizing all the binary resources including proprietary libraries, remote artifacts and other 3rd party resources. It is a secure and restricted server, only collaboration members will have access to this server. OEM secure information can be hosted in Artifactory server.

OEM vendors will work in collaboration with the SoC vendor. SoC vendor can define a HAL layer, share the source of HAL & yocto meta layer that can be stored in RDK CMF Git repository, share the SDK binary that can be stored in RDK Artifactory (Shared only from authorized SoC vendors who will work in collaboration with the OEM vendor) and then publish necessary documentation on how to build the OEM SDK. OEM vendor can use the git/ Artifactory for periodic updates (for releases) or for bug fixes. All the source code, binary and documentation will be strictly access restricted and access will be allowed only for authorized personnel by OEM vendor.

Artifactory server can be accessed by adding the Artifactory details and login credentials in the .netrc file, just like it is done for normal git repositories. A sample is given below:

```
machine your.artifactory.host
login YOUR_ARTIFACTORY_USERNAME
password YOUR_PASSWORD_OR_KEY_OR_TOKEN
```

Complete source code

In this approach, OEM vendors will work in collaboration with the SoC vendor. SoC vendor can define a HAL layer, share the source of HAL & yocto meta layer that can be stored in RDK CMF Git repository and then publish necessary documentation on how to build the OEM SDK. OEM vendor can use the git/ Artifactory for periodic updates (for releases) or for bug fixes. All the source code, binary and documentation will be strictly access restricted and access will be allowed only for authorized personnel by OEM vendor.

For both approaches, the RDKM collaboration zone will be used with strict access restrictions.

Collaboration with SoC

After a successful bring up of RDK on SoC platform, the next step will be to allow SoCs to work with OEMs to get a device based on the SoC platform. RDKM offers collaboration space for OEMs which would help OEMs to collaborate with SoC and RDK teams (as well as any 3rd party) in their journey to engineer a successful RDK product. RDKM collaboration zone includes features like (but not limited to) CMF facility to maintain build manifests as well as SoC/OEM specific code, SoC SDK artifact storage facility, JIRA & RDK Wiki spaces, integration with Test & Certification suites, monthly & release tagging etc.

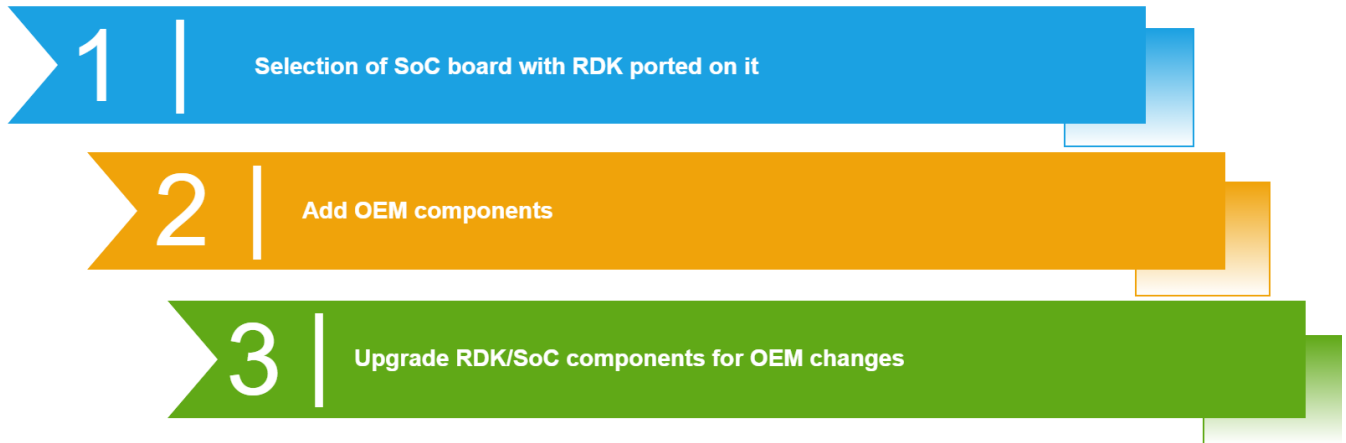
Please refer [RDKM On-boarding](#) for more details on facilities available for SoCs and OEMs as part of collaboration zone. In short, it will include:

- Access restricted Git repositories and Artifactory servers
- Access restricted Confluence and JIRA spaces for Management and Documentation
- Access to RDKM support as well as extended documentation
- Access to test & certification support

Procedure for OEM porting of RDK

The aim of this document is to explain the procedure for OEM porting of RDK.

Procedure for OEM porting of RDK



Step 1 : Selection of SoC board with RDK ported on it

Refer this page [SoC Platform Firmware](#) to know the details about Yocto manifests, SoC meta-layer creation includes adding the Machine Configuration File for the new SoC .

Step 2 : Add OEM components

OEM needs to add OEM specific components like Firmware Upgrade, Secure Boot Loader, MFR libraries, Vendor Specific Information, NVRAM files and partition, Provisioning, OEM Specific drivers, STB Utilities, RDK Device-Specific Patches, Image Generation Utilities etc. as well as interfacing layers to the generic RDK for relevant OEM code modules (see below)

Step 3 : Upgrade RDK/SoC components for OEM changes

Any Revision change in SoC layer is usually done by SoC's build environment and the new SDK or revision is updated in recipe. If a new recipe is added for any update in SoC software, then can be handled using PREFERRED_VERSION Yocto flag in meta layer

Components of OEM Interface

Bluetooth

Bluetooth Manager implements the Bluetooth HAL i.e. Bluetooth Core (BTRCore) API. Bluetooth HAL interface provides a software abstraction layer that interfaces with the actual Bluetooth implementation and/or drivers. RDK Bluetooth HAL layer enables projects to pick any Bluetooth profiles as per their requirements. Bluetooth HAL uses BlueZ5.42 stack which is a quite popular Linux Bluetooth library.

- Bluetooth HAL - Provides APIs to perform Bluetooth operations by abstracting and simplifying the complexities of Bt-Ifce (& BLuez)
- Bt-Ifce - Abstracts Bluez versions and serves as a HAL for other Bluetooth stacks - Interacts with Bluez over Dbus
- Bluez - Interacts with kernel layer bluetooth modules

Modules

- [Bluetooth Core APIs](#)
- [Bluetooth Core Types](#)

Crash Upload

- Uploads core dumps to a FTP server if there are any
- This interface is optional, OEM may implement a customized script for uploading the crash dump files to a server using specific certificate files

Device Settings

- OEM implementation includes device specific configuration parameters & customizing data structures for front panel setting and so on

DTCP

- Integrates the SoC provided DTCP library with DTCP/IP manager Interface implementation which Manages source/sink DTCP/IP sessions and performs the encryption/decryption
- Provides setup scripts to initialize the default DTCP data to device specific persistent partition

hwselftest

Provides platform specific configuration options for Hardware test. Which will run periodically in background to check attached hardware health.

- OEM dependency is limited to providing a configuration file hwselftest.conf which defines the hardware interfaces that should be tested e. g. partition name, HDMI, IR, MOCA and so on

LED Manager

LED Manager is used to control the LED patterns during different system events.

- OEM should implement the template code to ensure that it matches with their LED coloring scheme and other configuration parameters such as brightness level and number of LED types, multi-color or single color LEDs

tenableHDCP

This handles the HDCP service operations such as enable or disable the HDCP.

- OEM interface includes implementing manufacturer specific calls to read the keys and so on

Wi-Fi

- OEM specific Wi-Fi driver configuration files and utility library can be added optionally



For details on the Wi-Fi HAL Public APIs and Data Types, please refer: https://wiki.rdkcentral.com/doxygen/rdkv-opensourced/df/dce/group__w_i_f_i__h_a_l.html