RDK-V Getting Source Code and Build

Access to CMF repository

Contact the support team at *support@rdkcentral.com* to get access to CMF repository

Download source code

The RDK code can be obtained from the CMF Gerrit instance using the `repo` commands below :

```
$ repo init -u https://YOUR_USERNAME@code.rdkcentral.com/r/manifests -b <release>
$ repo sync --no-clone-bundle
```

Note : The above commands illustrate how to access the specific CMF *iteration* (snapshot). Substitute the latest iteration currently available in RDKV-Releases page. If you want to get the very **latest** state of the code (i.e. current development), do not supply a branch, and `master` will be fetched:

```
$ repo init -u https://YOUR_USERNAME@code.rdkcentral.com/r/manifests
$ repo sync --no-clone-bundle
```

The first command will download the manifest, while the other command will fetch and checkout all the relevant git trees. Now you have a complete folder structure ready to build the RDK OpenEmbedded distro. The Manifest file(manifest.xml or default.xml) defines which repositories the project uses and links to appropriate revisions of each git repository. After completing "repo init", manifest files will be downloaded in ".repo/manifests/" path.

repo init command for RDK Emulator:

repo init -u https://code.rdkcentral.com/r/manifests -b <release> -m emulator.xml

Note : - Repo commands will not fetch any of the source code. Instead fetch the recipes which will have the URIs of source code repositories.

- Cloning the code before login once to code.rdkcentral.com, user would get the Authentication error, even though the account is in good standing and has all the required access.
- Please login to code.rdkcentral.com before attempting to clone.

Initializing the Build Environment

To build OpenEmbedded we first need to initialize the build environment and to generate the build configuration files, namely:

- <BUILDDIR>/conf/local.conf which contains build parameters, such as distro name, options to control build parallelism, ..
- <BUILDDIR>/conf/bblayers.conf which contains the list of "OE Layers" that we want to use for the build.

A "helper" script has been created to facilitate the creation of these files. You can initialize the build environment with:

```
$ cd <workspace dir>
```

```
$ source meta-cmf/setup-environment
```

The above step configures and sets up your directory to start an appropriate build for hybrid or media client. There are different kinds of builds listed. Please read the options and select the number of the build you need. It moves the user to a new build folder from where the bitbake command is executed. setup-enviornment is the script used to prepare the host for a bitbake build.

The build/ directory structure:

- conf/ : Configuration files - image specific and layer configuration.
- downloads/ : This folder stores the downloaded upstream tarballs of the packages used in the builds, facilitating fast rebuilds. If the content of this folder is deleted, the builds will go and refetch the source tars again.
- sstate-cache/ : Shared state cache, it is the local prebuilt store used by all builds. It will be populated when you do the builds. It is important to keep this directory safe for sstate reuse.
- tmp/ : Holds all the build.
- tmp/buildstats/ : Build statistics for all packages built (CPU usage, elapsed time, host, timestamps).
- tmp/deploy/ : Final output of the build.
- tmp/deploy/images/ : Contains the complete images built by the OpenEmbedded build system. These images are used to flash the target.
- tmp/work/ : Set of specific work directories, split by architecture. They are used to unpack, configure and build the packages. Contains the patched sources, generated objects and logs.
- tmp/sysroots/ : Shared libraries and headers used to compile packages for the target but also for the host.

Note: build-<machine> (e.g. build-oem-platform) - This is the object/build directory, all objects and intermediate files for all components are stored in this folder. if you want to do a clean build you can delete this folder, then run ./meta-cmf/setup-environment and rebuild. The build will be very fast since it will reuse the sstate (prebuilts) during this build, assuming the sstate-cache directory was populated with previous builds already.

The Kernel image and root filesystem will be created under ../build_xxx/tmp/deploy/images folder.

Building the OEM Platform Image

The repo tool is used to set up the workspace and bitbake is used to facilitate builds. The following commands serve to illustrate the steps involved in building for a platform

```
$ repo init -u https://<URL>/r/<platform manifests> -b master -m oem-soc-platform.xml --no-repo-verify
$ repo sync
 (patch qtwebsockets, jquery in generic subdirectory where externalsrc source trees are located)
$ MACHINE=oem-platform . ./meta-cmf/setup-environment
$ bitbake cmf-generic-mediaclient-image
```

Summary:

- **repo init** downloads the repo manifest and initializes it.
- **platform manifests** is the repository that contains all manifests.
- **oem-soc-platform.xml** is the OEM's specific platform manifest, containing URLs used to build the platform. A snippet of this manifest is illustrated below.
- **repo sync** downloads all the meta data and the source code required to build the platform image.
- **bitbake** uses recipes in order to figure out how to build each component. There are several images supported.

In a normal Yocto build build, the repo sync command would download the metadata and then bitbake, upon execution, would download the code and build. In this instance, the repo sync command downloads not just the recipes but also the code for individual components using a Yocto/OE feature called EXTERNALSRC.

Note: The RDKM CMF Gerrit does not distribute OEM/SoC specific meta layers or platform manifests. It is necessary to contact the SoC vendor for these packages and supporting information.

Example manifest file snippet

The following sample manifest file is taken from CMF stack :

```
<!-- Yocto OpenEmbedded build environment -->

<project path="openembedded-core/bitbake" remote="github" name="openembedded/bitbake" revision="1.28" />
<project path="meta-browser" remote="github" name="OSSystems/meta-browser" revision="
63963cc56c8d0291779693e62b66cb16e5c86883" upstream="master" />
<project path="meta-java" remote="yocto" name="meta-java" revision="66c97ae7461f4c1a13917131787bb76dc34e6b6f"
upstream="master" />
<project path="meta-linaro" remote="linaro" name="openembedded/meta-linaro" revision="
06008235ca752fea678953e85adaa29a491d246b" upstream="daisy" />
<project path="meta-openembedded" remote="github" name="openembedded/meta-openembedded" revision="
d3d14d3fcca7fcde362cf0b31411dc4eea6d20aa" upstream="daisy" />
<project path="meta-qt5" remote="github" name="meta-qt5/meta-qt5" revision="
f6f2d16260ae5b35828264a553c05f065aa9d7e2" upstream="daisy" />
<project path="meta-raspberrypi" remote="yocto" name="meta-raspberrypi" revision="
68634deeed2a930a15eedd02e6b1de7d1e014d3b" upstream="master" />
<project path="meta-virtualization" remote="yocto" name="meta-virtualization" revision="
a89c11a3d89601f6d8499bd7d0f265cf4512d772" upstream="master" />
<project path="openembedded-core" remote="github" name="openembedded/openembedded-core" revision="
e1a2540227250d854d5bba278634bcc9e7572cda" upstream="daisy" />

<!-- Yocto OpenEmbedded patches for CMF Yocto builds-->

<project name="components/opensource/patches/rdk-oe" path="patches/rdk-oe" />

<!-- Yocto RDK build environment -->

<project name="components/generic/rdk-oe/meta-rdk" path="meta-rdk" />
<project name="components/generic/rdk-oe/meta-rdk-bsp-raspberrypi" />
<project name="components/generic/rdk-oe/meta-rdk-video" path="meta-rdk-video" />


<!-- Yocto CMF build environment -->

<project name="components/generic/rdk-oe/meta-cmf" path="meta-cmf" />
<project name="components/generic/rdk-oe/meta-cmf-raspberrypi" path="meta-cmf-raspberrypi"/>
<project name="components/generic/rdk-oe/meta-cmf-video" path="meta-cmf-video" />
```