Build a Smarter World

EC200U-CN QuecOpen Quick Start Guide

LTE Standard Module Series

Version: 1.0.0

Date: 2021-02-10

Status: Preliminary

www.quectel.com

About the Document

Revision History

Version	Date	Author	Description
-	2021-02-01	Jensen FANG	Creation of the document
1.0.0	2021-02-10	Jensen FANG	Preliminary

Contents

Table Index

Figure Index

1 Introduction

Quectel EC200U-CN module supports QuecOpen® solutions. This document mainly introduces SDK directory structure, compilation environment, compilation process and the general flow of APP development of Quectel EC200U-CN QuecOpen module and how to pack the pre-set file in the APP firmware package.

2 SDK Directory Structure

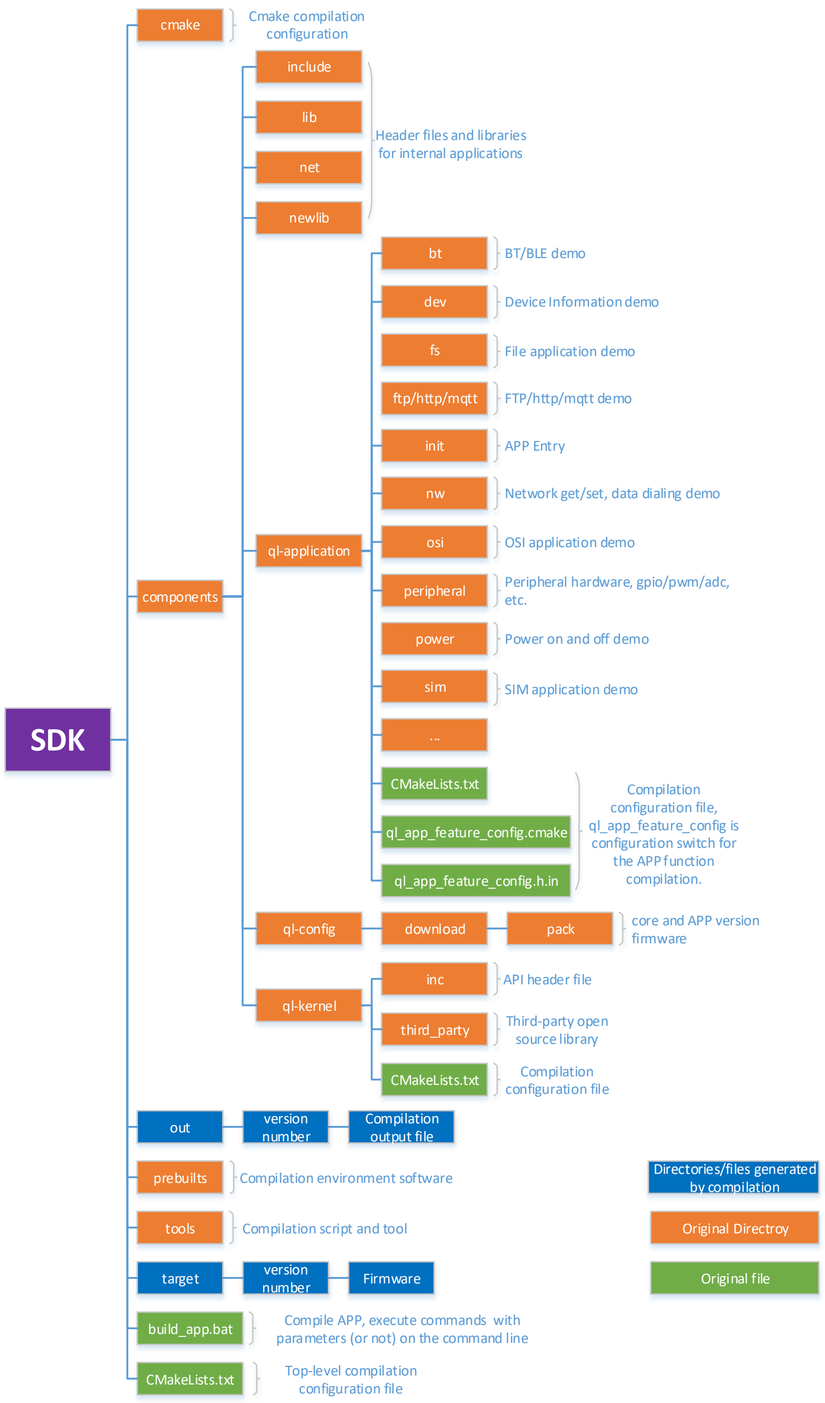

SDK
cmake
Cmake compilation configuration
include
lib
net
newlib
Header files and libraries for internal applications
components
ql-application
bt
BT/BLE demo
dev
Device Information demo
fs
File application demo
ftp/http/mqtt
FTP/http/mqtt demo
init
APP Entry
nw
Network get/set, data dialing demo
osi
OSI application demo
peripheral
Peripheral hardware, gpio/pwm/adc, etc.
power
Power on and off demo
sim
SIM application demo
...
CMakeLists.txt
ql_app_feature_config.cmake
ql_app_feature_config.h.in
Compilation configuration file, ql_app_feature_config is configuration switch for the APP function compilation.
ql-config
download
pack
core and APP version firmware
ql-kernel
inc
API header file
third_party
Third-party open source library
CMakeLists.txt
Compilation configuration file
out
version number
Compilation output file
prebuilts
Compilation environment software
tools
Compilation script and tool
target
version number
Firmware
build_app.bat
Compile APP, execute commands with parameters (or not) on the command line
CMakeLists.txt
Top-level compilation configuration file
Directories/files generated by compilation
Original Directroy
Original file

Figure 1: SDK Directory Structure

The above figure shows the SDK directory structure of EC200U-CN QuecOpen module, the details are as follows:

Table 1: SDK Directory Structure

Directory Name	Description
cmake/prebuilts/tools	Stores the environment, configuration and scripts related to compilation.
components/ql-config	Currently stores core version firmware and default APP firmware.
components/ql-kernel	Stores all open API header files, and third-party open source libraries.
components/ql-application	Stores APP reference demos to implement demo applications of various functional components.
components/include components/lib components/net components/newlib	Stores various header files and library files that the APP needs
build_app.bat	Execute the command as the format below to compile the APP in the command line under Windows, as follows: **build_app r/new/clean/h/-h QuecProj AppTarget [release/debug]** If there is no parameter in the command, it will compile as follows by default: **build_app new EC200UCN_AA appimage release**. See ***Chapter 4*** for details.

NOTE

The directory in SDK may differ depending on the actual released versions.

3 Compilation Environment

3.1. Host Operating System Version

The host operating system version for compilation should be 64-bit Windows 7 or Windows 10. The Host should install the software below:

- Visual C++ Redistributable for Visual Studio 2015 x86 or above
- Visual C++ Redistributable for Visual Studio 2015 x64 or above

NOTE

It is not recommended to use Windows XP and Windows 8.x for their reliability has not been verified.

3.2. APP Compilation Environment

3.2.1. Tool Chain

The APP compilation tool chain uses gcc-arm-none-eabi, that is, GCC tool chain for ARM bare host system.

3.2.2. Build Environment

APP compilation environment only uses gcc-arm-none-eabi with version 7.2.1, located in the directory of *prebuilts*, integrated in SDK. The gcc-arm-none-eabi tool chain installed on the host system is currently not supported.

3.2.3. Other Tools

Such as cmake, python3, located in the *prebuilts* directory.

4 APP Compilation Outputs

In the *components\ql-config\download\pack* directory, kernel firmware files, the default APP firmware files, the corresponding *.map* and *.elf* files, named after the project model, and the firmware files merged by kernel and APP are saved.

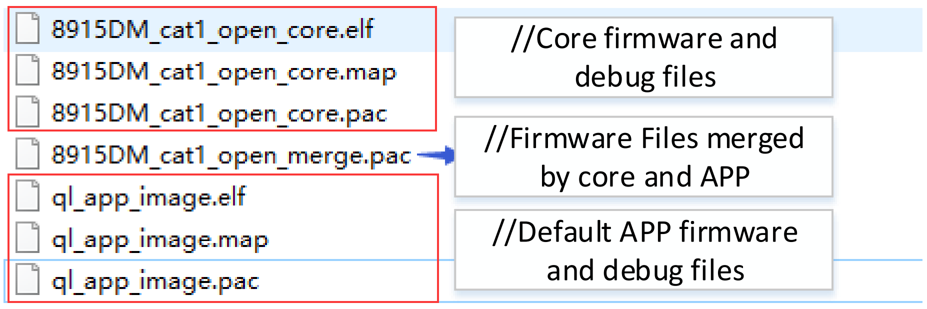

8915DM_cat1_open_core.elf
8915DM_cat1_open_core.map
8915DM_cat1_open_core.pac
8915DM_cat1_open_merge.pac
ql_app_image.elf
ql_app_image.map
ql_app_image.pac
//Core firmware and debug files
//Firmware Files merged by core and APP
//Default APP firmware and debug files

Figure 2: Basic Firmware in SDK

When APP is compiled via **build_app** scripts, the second parameter checks whether there is a kernel firmware of the corresponding project model in the *components\ql-config\download\pack* directory, if there is, it will continue to compile. After the compilation is complete, a new APP firmware will be generated, and the core firmware and the new APP file will be merged into a new file in the *target* directory for further download.

4.1. Compilation Method

The APP code is located in the *components\ql-application* directory, and the compilation script is **build_app.bat** in the *components* directory. You can use **build_app.bat -h** to get the usage method, as follows:

```
Usage: build_app r/new QuecProj AppTarget [release/debug]
        build_app clean
        build_app h/-h
Example:
        build_app new EC200UCN_AA appimage release
```

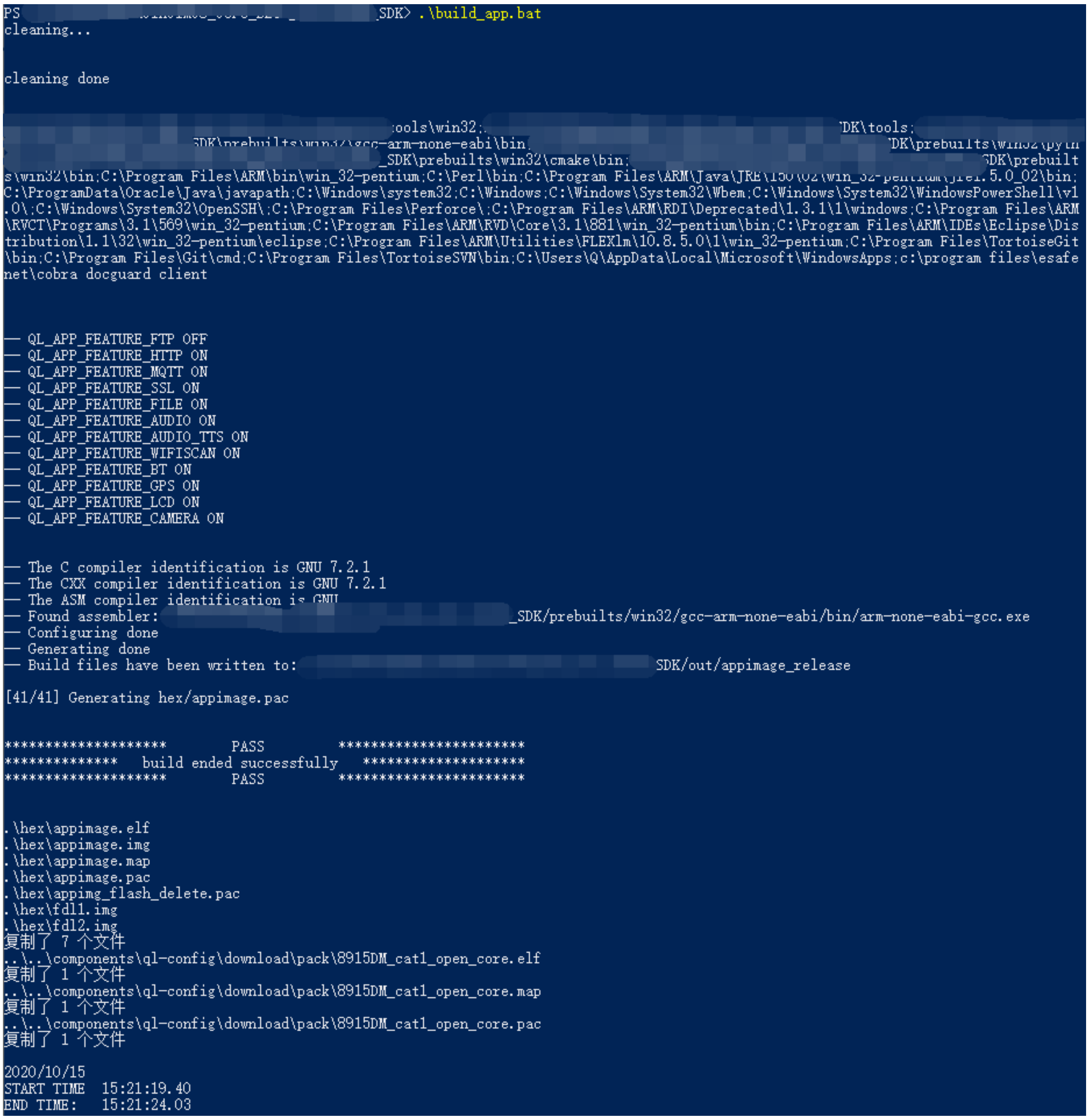
In the command **build_app r/new/clean/h/-h QuecProj AppTarget [release/debug]**:
the first parameter is the compilation type, including incremental compilation **r**, new compilation **new**,

clear **clean**, and help information **h** or **-h**; the second parameter *{QuecProj}* is the current Quectel module model used by customers currently; the third parameter *{AppTarget}* is the target firmware version, that is, file name of the APP firmware. The fourth parameter is optional.

If there is no parameter in the command, it will compile as follows by default:
build_app new EC200UCN_AA appimage release.

Before compilation, the second parameter will check whether there is a kernel firmware of the corresponding project model in the *components\ql-config\download\pack* directory. If not, it will prompt an error.

Start the command line or PowerShell (only for Win10) to locate the SDK root directory (identified by {ql-sdk-root}), or start the command line or PowerShell directly in the SDK root directory, and then enter the compilation command, Press Enter to compile. An example of the execution result is shown in the figure below:

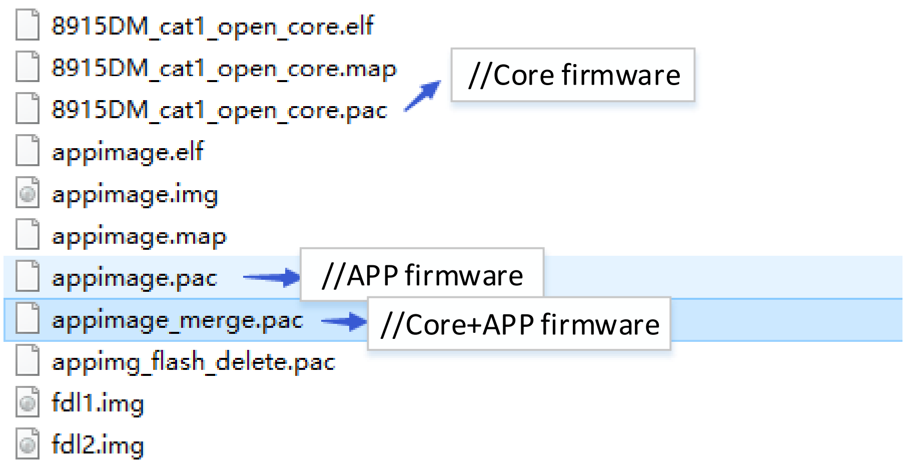


```
PS [redacted]SDK> .\build_app.bat
cleaning...


cleaning done


[redacted]ools\win32;[redacted]DK\tools;[redacted]
[redacted]SDK\prebuilts\win32\gcc-arm-none-eabi\bin;[redacted]DK\prebuilts\win32\pyth
[redacted]_SDK\prebuilts\win32\cmake\bin;[redacted]SDK\prebuilt
s\win32\bin;C:\Program Files\ARM\bin\win_32-pentium;C:\Perl\bin;C:\Program Files\ARM\Java\JRE\150\02\win_32-pentium\jre1.5.0_02\bin;
C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1
.0\;C:\Windows\System32\OpenSSH\;C:\Program Files\Perforce\;C:\Program Files\ARM\RDI\Deprecated\1.3.1\1\windows;C:\Program Files\ARM
\RVCT\Programs\3.1\569\win_32-pentium;C:\Program Files\ARM\RVD\Core\3.1\881\win_32-pentium\bin;C:\Program Files\ARM\IDEs\Eclipse\Dis
tribution\1.1\32\win_32-pentium\eclipse;C:\Program Files\ARM\Utilities\FLEXlm\10.8.5.0\1\win_32-pentium;C:\Program Files\TortoiseGit
\bin;C:\Program Files\Git\cmd;C:\Program Files\TortoiseSVN\bin;C:\Users\Q\AppData\Local\Microsoft\WindowsApps;c:\program files\esafe
net\cobra docguard client



-- QL_APP_FEATURE_FTP OFF
-- QL_APP_FEATURE_HTTP ON
-- QL_APP_FEATURE_MQTT ON
-- QL_APP_FEATURE_SSL ON
-- QL_APP_FEATURE_FILE ON
-- QL_APP_FEATURE_AUDIO ON
-- QL_APP_FEATURE_AUDIO_TTS ON
-- QL_APP_FEATURE_WIFISCAN ON
-- QL_APP_FEATURE_BT ON
-- QL_APP_FEATURE_GPS ON
-- QL_APP_FEATURE_LCD ON
-- QL_APP_FEATURE_CAMERA ON


-- The C compiler identification is GNU 7.2.1
-- The CXX compiler identification is GNU 7.2.1
-- The ASM compiler identification is GNU
-- Found assembler: [redacted]_SDK/prebuilts/win32/gcc-arm-none-eabi/bin/arm-none-eabi-gcc.exe
-- Configuring done
-- Generating done
-- Build files have been written to: [redacted]SDK/out/appimage_release

[41/41] Generating hex/appimage.pac


********************        PASS         ***********************
**************   build ended successfully   ********************
********************        PASS         ***********************


.\hex\appimage.elf
.\hex\appimage.img
.\hex\appimage.map
.\hex\appimage.pac
.\hex\appimg_flash_delete.pac
.\hex\fdl1.img
.\hex\fdl2.img
复制了 7 个文件
..\..\components\ql-config\download\pack\8915DM_cat1_open_core.elf
复制了 1 个文件
..\..\components\ql-config\download\pack\8915DM_cat1_open_core.map
复制了 1 个文件
..\..\components\ql-config\download\pack\8915DM_cat1_open_core.pac
复制了 1 个文件

2020/10/15
START TIME  15:21:19.40
END TIME:   15:21:24.03
```

Figure 3: APP Compilation Result

During the compilation process, a compilation output file will be generated in the *{ql-sdk-root}/out* directory.

After the compilation is successful, the target APP firmware version will be generated in the *{ql-sdk-root}/target* directory. The example is as follows:



```
8915DM_cat1_open_core.elf
8915DM_cat1_open_core.map
8915DM_cat1_open_core.pac
appimage.elf
appimage.img
appimage.map
appimage.pac
appimage_merge.pac
appimg_flash_delete.pac
fdl1.img
fdl2.img
```

Figure 4: APP Compilation Output File

If you only need to update the APP firmware, download the *.pac* file.

If you need to update core firmware and APP firmware, download the firmware file merged by core and APP firmware.

build_app clean is used to clean the compilation files in the *{ql-sdk-root}/out* directory, without deleting the target firmware files in *{ql-sdk-root}/target*. The previous firmware files in the *{ql-sdk-root}/target* directory will be cleared before a successful compilation.

5 APP Code Development

5.1. APP Demo Details

In the SDK directory structure, the APP code is located under*{ql-sdk-root}/components/ql-application*, the details are as follows:

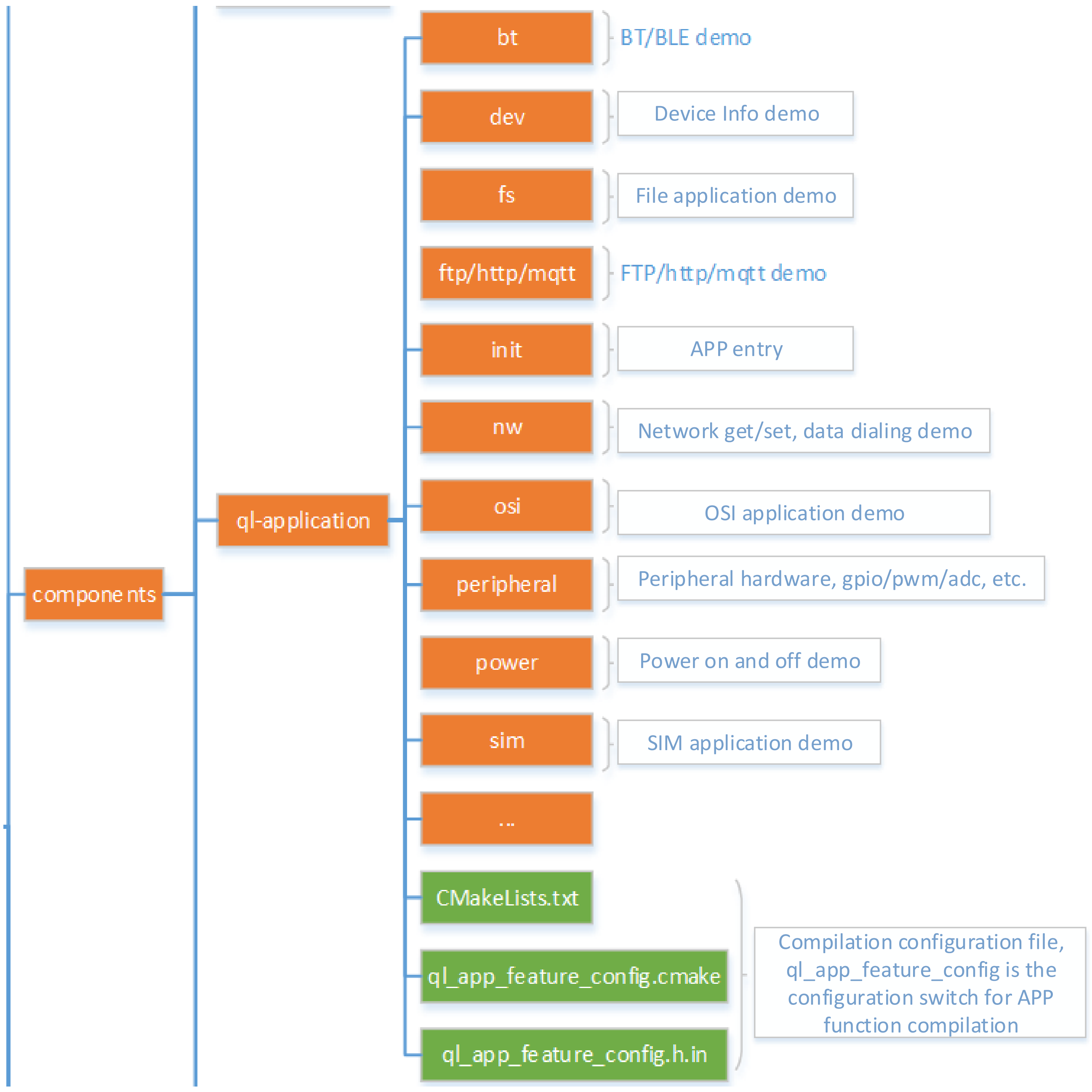
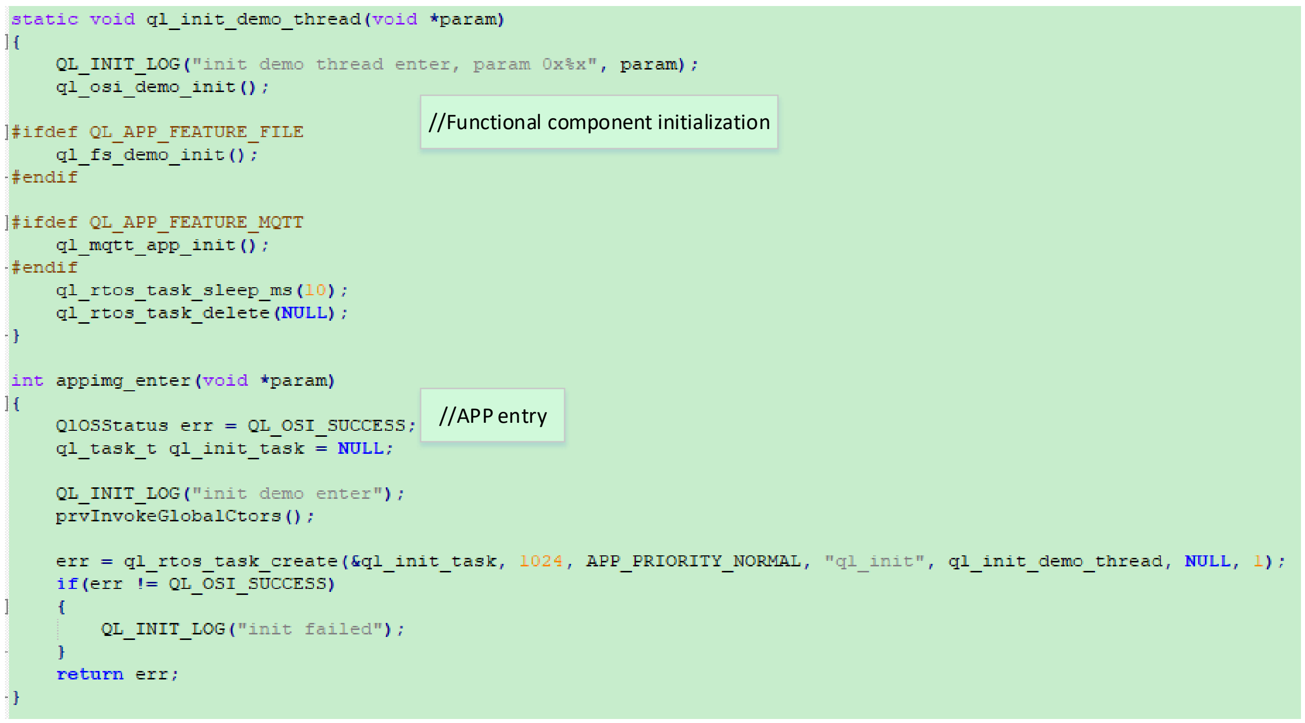

components
ql-application
bt
BT/BLE demo
dev
Device Info demo
fs
File application demo
ftp/http/mqtt
FTP/http/mqtt demo
init
APP entry
nw
Network get/set, data dialing demo
osi
OSI application demo
peripheral
Peripheral hardware, gpio/pwm/adc, etc.
power
Power on and off demo
sim
SIM application demo
...
CMakeLists.txt
ql_app_feature_config.cmake
ql_app_feature_config.h.in
Compilation configuration file, ql_app_feature_config is the configuration switch for APP function compilation

Figure 5: APP Code Structure

The *init* directory is the APP entry, that is, the *appimg_enter* function in *ql_init.c* in this directory. The demo code starts an initialization thread in this function, and the initialization function of each functional component needs to be called in the initialization thread.



```
static void ql_init_demo_thread(void *param)
{
    QL_INIT_LOG("init demo thread enter, param 0x%x", param);
    ql_osi_demo_init();

#ifdef QL_APP_FEATURE_FILE
    ql_fs_demo_init();
#endif

#ifdef QL_APP_FEATURE_MQTT
    ql_mqtt_app_init();
#endif
    ql_rtos_task_sleep_ms(10);
    ql_rtos_task_delete(NULL);
}

int appimg_enter(void *param)
{
    QlOSStatus err = QL_OSI_SUCCESS;
    ql_task_t ql_init_task = NULL;

    QL_INIT_LOG("init demo enter");
    prvInvokeGlobalCtors();

    err = ql_rtos_task_create(&ql_init_task, 1024, APP_PRIORITY_NORMAL, "ql_init", ql_init_demo_thread, NULL, 1);
    if(err != QL_OSI_SUCCESS)
    {
        QL_INIT_LOG("init failed");
    }
    return err;
}
```

Figure 6: APP Entry

Each functional component is classified according to the folder. You can add them based on the folder name as needed.

5.2. APP Functional Components Adding

You can add the function directory under the *{ql-sdk-root}/components/ql-application* directory, which contains the required code source files, header files and compilation configuration file *CMakeLists.txt*. Take audio_demo as an example, its directory structure is as follows:

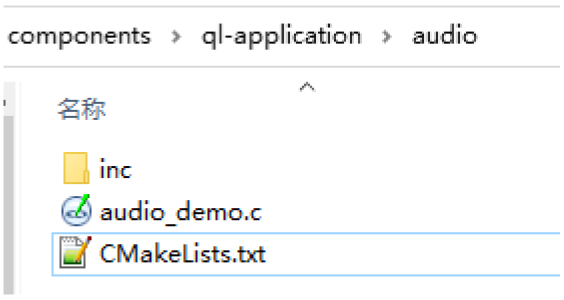
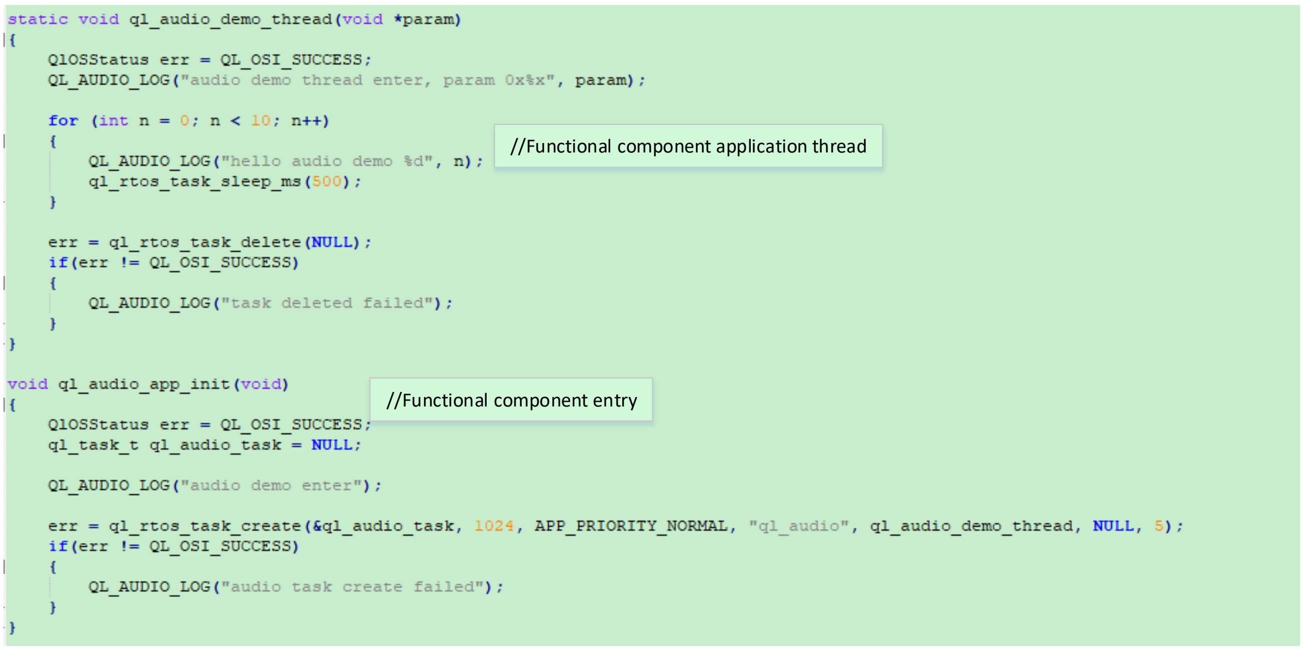


Figure 7: Audio Demo Directory

5.2.1. Functional Component Entry

In the entry function of the functional component, you can use *ql_rtos_task_create()* to create the application thread of the functional component. An example is as follows:



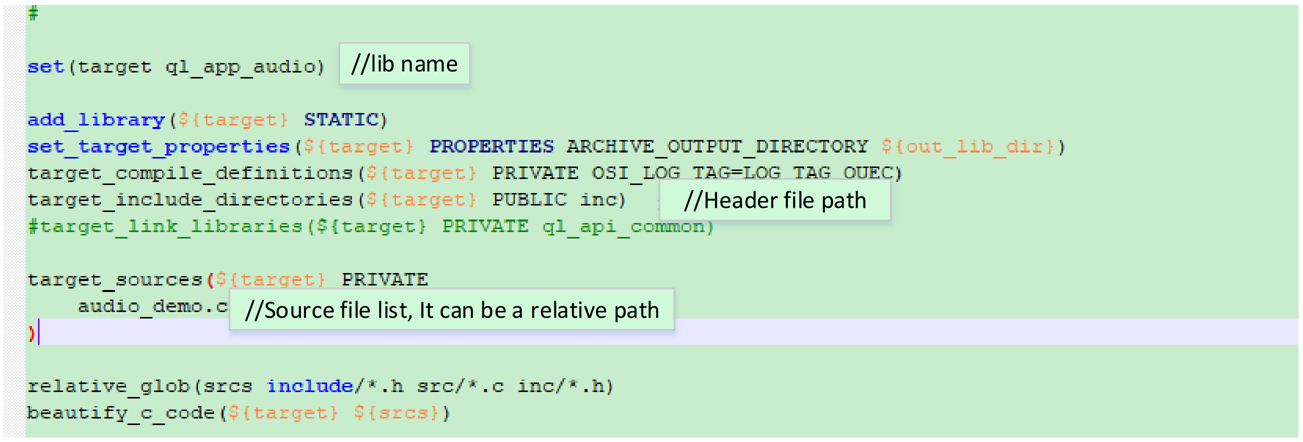
```
static void ql_audio_demo_thread(void *param)
{
    QlOSStatus err = QL_OSI_SUCCESS;
    QL_AUDIO_LOG("audio demo thread enter, param 0x%x", param);

    for (int n = 0; n < 10; n++)
    {
        QL_AUDIO_LOG("hello audio demo %d", n);
        ql_rtos_task_sleep_ms(500);
    }

    err = ql_rtos_task_delete(NULL);
    if(err != QL_OSI_SUCCESS)
    {
        QL_AUDIO_LOG("task deleted failed");
    }
}

void ql_audio_app_init(void)
{
    QlOSStatus err = QL_OSI_SUCCESS;
    ql_task_t ql_audio_task = NULL;

    QL_AUDIO_LOG("audio demo enter");

    err = ql_rtos_task_create(&ql_audio_task, 1024, APP_PRIORITY_NORMAL, "ql_audio", ql_audio_demo_thread, NULL, 5);
    if(err != QL_OSI_SUCCESS)
    {
        QL_AUDIO_LOG("audio task create failed");
    }
}
```

//Functional component application thread
//Functional component entry

Figure 8: APP Function Component Entry

5.2.2. Compilation Configuration File Modification

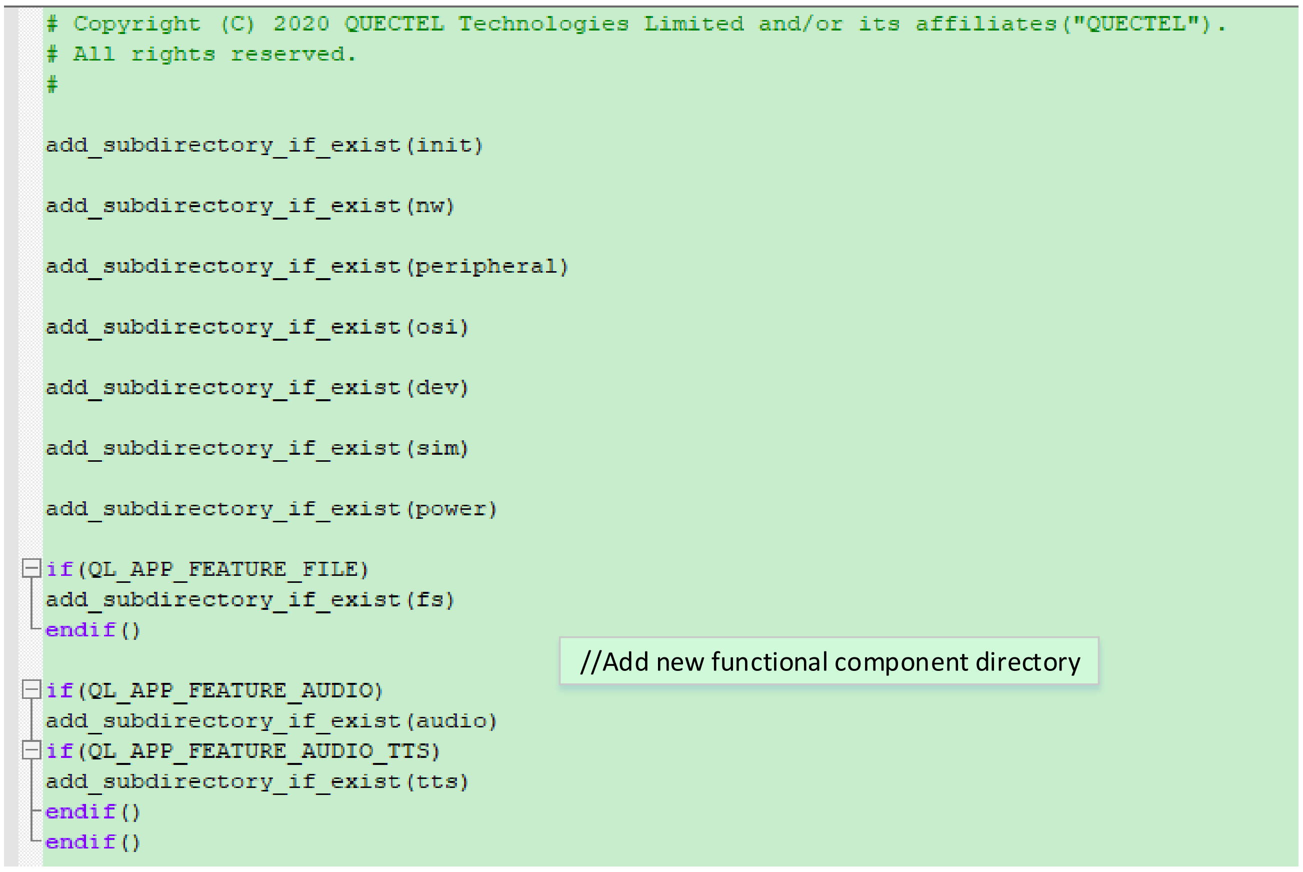
In the configuration file *CMakeLists.txt*, there are three points below needed to be modified:



```
#

set(target ql_app_audio)

add_library(${target} STATIC)
set_target_properties(${target} PROPERTIES ARCHIVE_OUTPUT_DIRECTORY ${out_lib_dir})
target_compile_definitions(${target} PRIVATE OSI_LOG_TAG=LOG_TAG_QUEC)
target_include_directories(${target} PUBLIC inc)
#target_link_libraries(${target} PRIVATE ql_api_common)

target_sources(${target} PRIVATE
    audio_demo.c
)

relative_glob(srcs include/*.h src/*.c inc/*.h)
beautify_c_code(${target} ${srcs})
```

Figure 9: APP Function Component Compilation Configuration

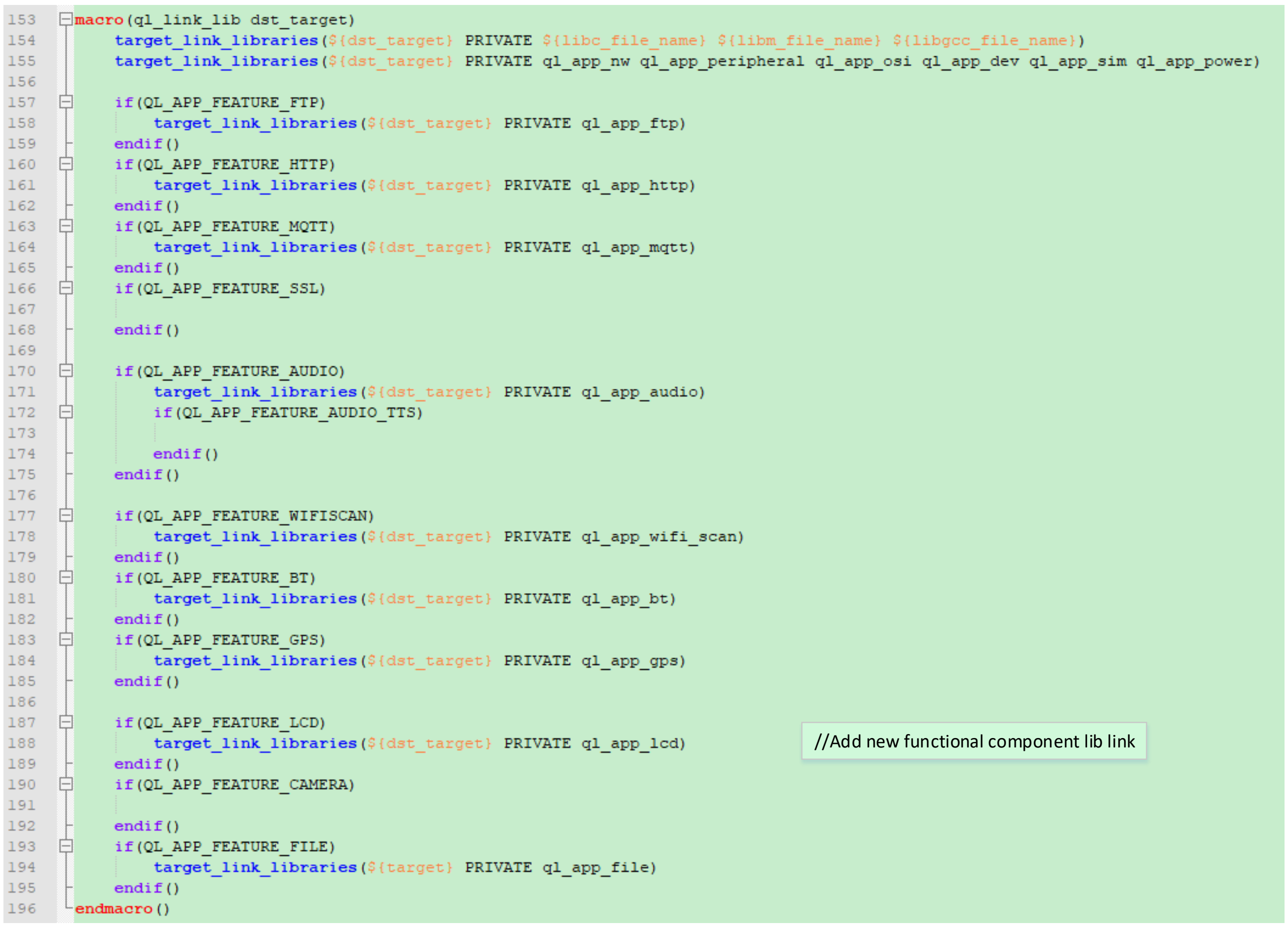
After that, you need to include this directory in the configuration file *CMakeLists.txt* in the *{ql-sdk-root}/components/ql-application* directory to enable it to be compiled, as follows:



```
# Copyright (C) 2020 QUECTEL Technologies Limited and/or its affiliates("QUECTEL").
# All rights reserved.
#

add_subdirectory_if_exist(init)

add_subdirectory_if_exist(nw)

add_subdirectory_if_exist(peripheral)

add_subdirectory_if_exist(osi)

add_subdirectory_if_exist(dev)

add_subdirectory_if_exist(sim)

add_subdirectory_if_exist(power)

if(QL_APP_FEATURE_FILE)
add_subdirectory_if_exist(fs)
endif()

if(QL_APP_FEATURE_AUDIO)
add_subdirectory_if_exist(audio)
if(QL_APP_FEATURE_AUDIO_TTS)
add_subdirectory_if_exist(tts)
endif()
endif()
```

Figure 10: New Functional Component Compilation

At this point, the new functional components can be compiled. Next, you need to configure the link to use the component: in the configuration file (*CMakeLists.txt*) in the root directory of *{ql-sdk-root}*, add the lib link of the new functional component, as follows:



```
macro(ql_link_lib dst_target)
    target_link_libraries(${dst_target} PRIVATE ${libc_file_name} ${libm_file_name} ${libgcc_file_name})
    target_link_libraries(${dst_target} PRIVATE ql_app_nw ql_app_peripheral ql_app_osi ql_app_dev ql_app_sim ql_app_power)

    if(QL_APP_FEATURE_FTP)
        target_link_libraries(${dst_target} PRIVATE ql_app_ftp)
    endif()
    if(QL_APP_FEATURE_HTTP)
        target_link_libraries(${dst_target} PRIVATE ql_app_http)
    endif()
    if(QL_APP_FEATURE_MQTT)
        target_link_libraries(${dst_target} PRIVATE ql_app_mqtt)
    endif()
    if(QL_APP_FEATURE_SSL)

    endif()

    if(QL_APP_FEATURE_AUDIO)
        target_link_libraries(${dst_target} PRIVATE ql_app_audio)
        if(QL_APP_FEATURE_AUDIO_TTS)

        endif()
    endif()

    if(QL_APP_FEATURE_WIFISCAN)
        target_link_libraries(${dst_target} PRIVATE ql_app_wifi_scan)
    endif()
    if(QL_APP_FEATURE_BT)
        target_link_libraries(${dst_target} PRIVATE ql_app_bt)
    endif()
    if(QL_APP_FEATURE_GPS)
        target_link_libraries(${dst_target} PRIVATE ql_app_gps)
    endif()

    if(QL_APP_FEATURE_LCD)
        target_link_libraries(${dst_target} PRIVATE ql_app_lcd)
    endif()
    if(QL_APP_FEATURE_CAMERA)

    endif()
    if(QL_APP_FEATURE_FILE)
        target_link_libraries(${target} PRIVATE ql_app_file)
    endif()
endmacro()
```

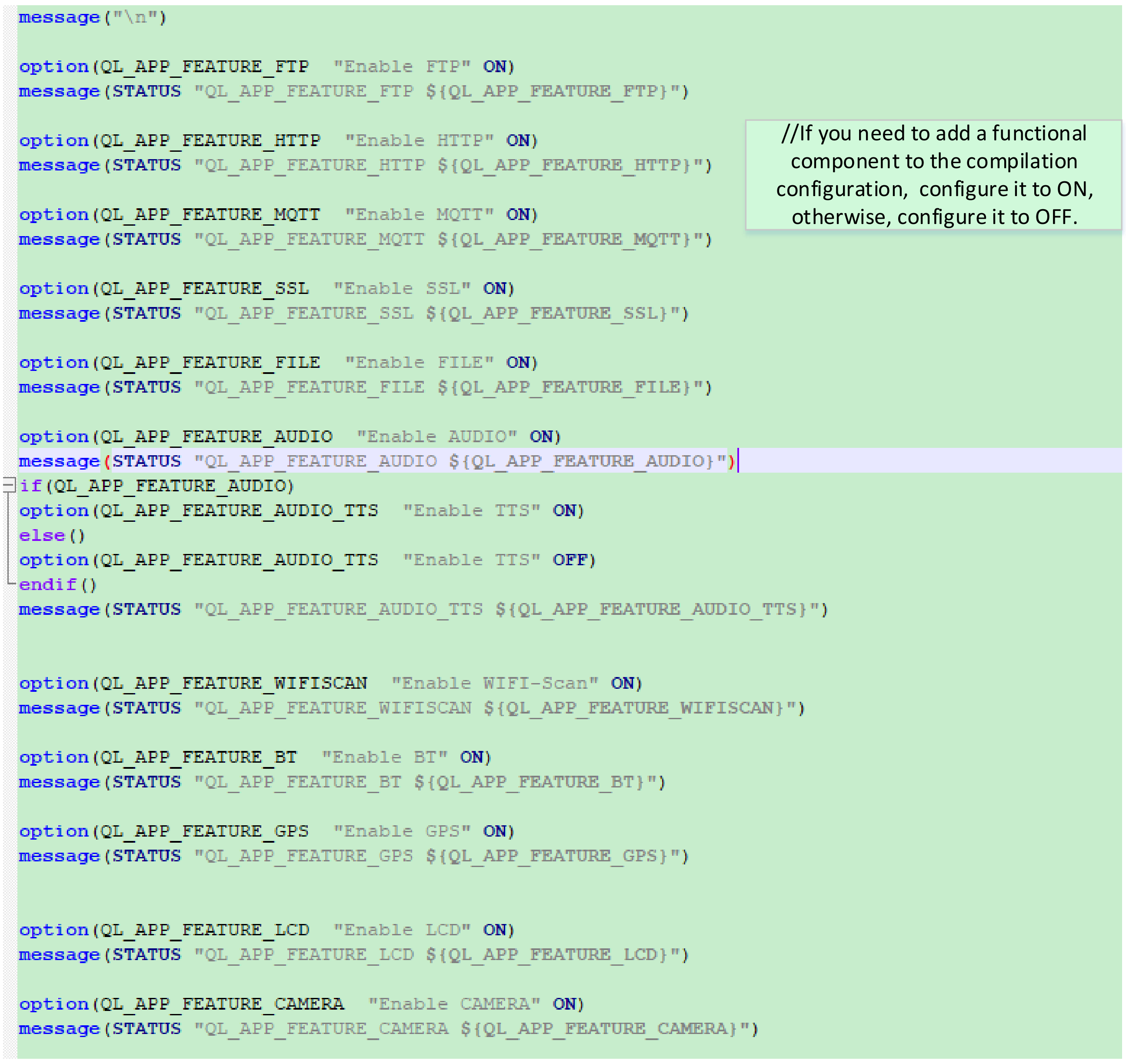
Figure 11: Link New Functional Components

At this time, the new component has been added. Then, refer to ***Chapter 4*** to compile the firmware.

5.3. Function Component Compilation On/Off

In order to facilitate the APP side to tailor the required functions, the *ql_app_feature_config.cmake* file in the *{ql-sdk-root}/components/ql-application* directory is used to configure which components are compiled by the compilation script; the *ql_app_feature_config.h.in* file is used to generate the component to enable the macro control in the code.

The details about *ql_app_feature_config.cmake* file are as follows:

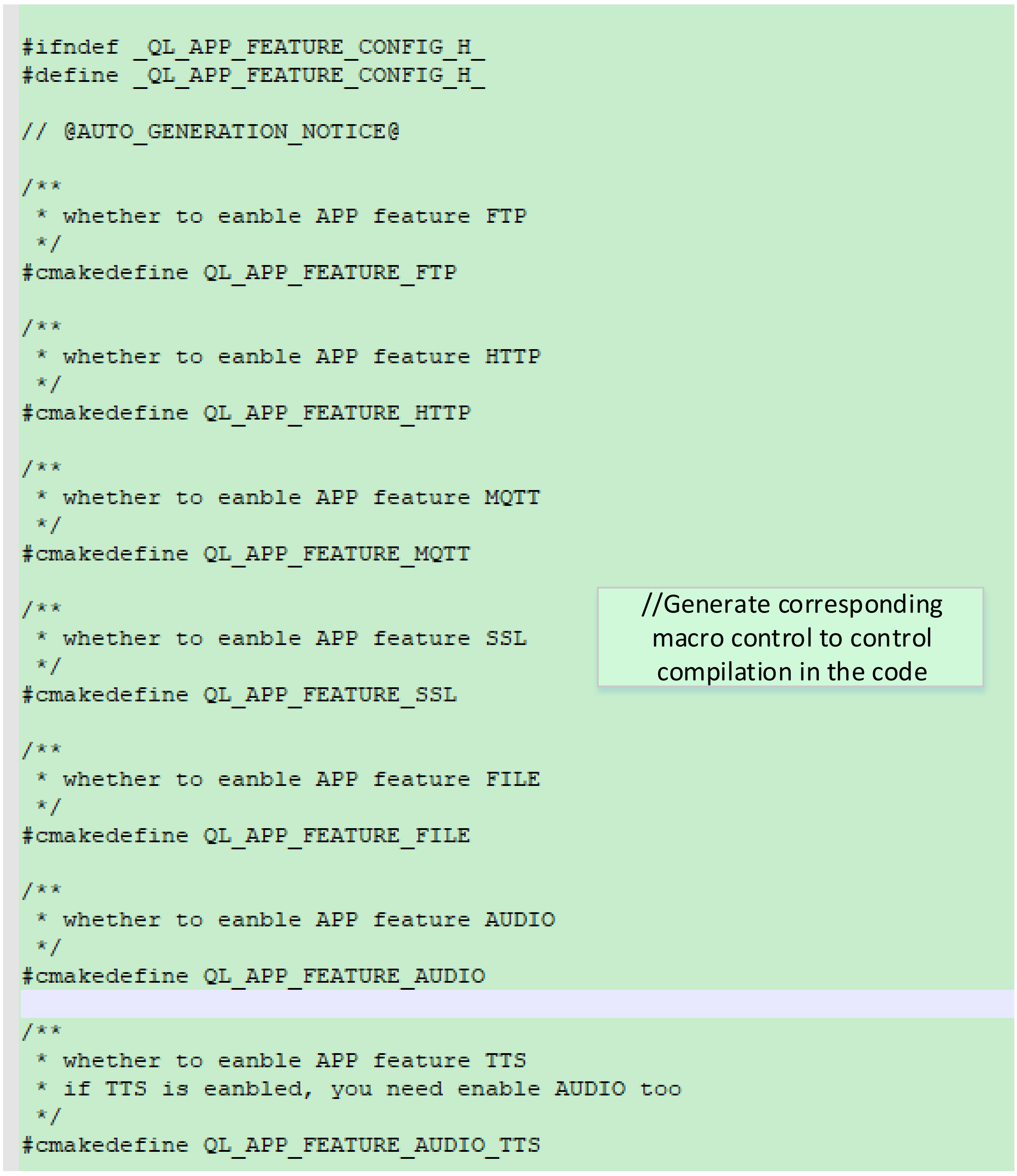


```
message("\n")

option(QL_APP_FEATURE_FTP  "Enable FTP" ON)
message(STATUS "QL_APP_FEATURE_FTP ${QL_APP_FEATURE_FTP}")

option(QL_APP_FEATURE_HTTP  "Enable HTTP" ON)
message(STATUS "QL_APP_FEATURE_HTTP ${QL_APP_FEATURE_HTTP}")

option(QL_APP_FEATURE_MQTT  "Enable MQTT" ON)
message(STATUS "QL_APP_FEATURE_MQTT ${QL_APP_FEATURE_MQTT}")

option(QL_APP_FEATURE_SSL  "Enable SSL" ON)
message(STATUS "QL_APP_FEATURE_SSL ${QL_APP_FEATURE_SSL}")

option(QL_APP_FEATURE_FILE  "Enable FILE" ON)
message(STATUS "QL_APP_FEATURE_FILE ${QL_APP_FEATURE_FILE}")

option(QL_APP_FEATURE_AUDIO  "Enable AUDIO" ON)
message(STATUS "QL_APP_FEATURE_AUDIO ${QL_APP_FEATURE_AUDIO}")
if(QL_APP_FEATURE_AUDIO)
option(QL_APP_FEATURE_AUDIO_TTS  "Enable TTS" ON)
else()
option(QL_APP_FEATURE_AUDIO_TTS  "Enable TTS" OFF)
endif()
message(STATUS "QL_APP_FEATURE_AUDIO_TTS ${QL_APP_FEATURE_AUDIO_TTS}")


option(QL_APP_FEATURE_WIFISCAN  "Enable WIFI-Scan" ON)
message(STATUS "QL_APP_FEATURE_WIFISCAN ${QL_APP_FEATURE_WIFISCAN}")

option(QL_APP_FEATURE_BT  "Enable BT" ON)
message(STATUS "QL_APP_FEATURE_BT ${QL_APP_FEATURE_BT}")

option(QL_APP_FEATURE_GPS  "Enable GPS" ON)
message(STATUS "QL_APP_FEATURE_GPS ${QL_APP_FEATURE_GPS}")


option(QL_APP_FEATURE_LCD  "Enable LCD" ON)
message(STATUS "QL_APP_FEATURE_LCD ${QL_APP_FEATURE_LCD}")

option(QL_APP_FEATURE_CAMERA  "Enable CAMERA" ON)
message(STATUS "QL_APP_FEATURE_CAMERA ${QL_APP_FEATURE_CAMERA}")
```

//If you need to add a functional component to the compilation configuration, configure it to ON, otherwise, configure it to OFF.

Figure 12: Component Compilation Enabling Configuration

The details about *ql_app_feature_config.h.in* file are as follows:



```
#ifndef _QL_APP_FEATURE_CONFIG_H_
#define _QL_APP_FEATURE_CONFIG_H_

// @AUTO_GENERATION_NOTICE@

/**
 * whether to eanble APP feature FTP
 */
#cmakedefine QL_APP_FEATURE_FTP

/**
 * whether to eanble APP feature HTTP
 */
#cmakedefine QL_APP_FEATURE_HTTP

/**
 * whether to eanble APP feature MQTT
 */
#cmakedefine QL_APP_FEATURE_MQTT

/**
 * whether to eanble APP feature SSL
 */
#cmakedefine QL_APP_FEATURE_SSL

/**
 * whether to eanble APP feature FILE
 */
#cmakedefine QL_APP_FEATURE_FILE

/**
 * whether to eanble APP feature AUDIO
 */
#cmakedefine QL_APP_FEATURE_AUDIO

/**
 * whether to eanble APP feature TTS
 * if TTS is eanbled, you need enable AUDIO too
 */
#cmakedefine QL_APP_FEATURE_AUDIO_TTS
```

//Generate corresponding macro control to control compilation in the code

Figure 13: Macro Control Enabling Configuration File

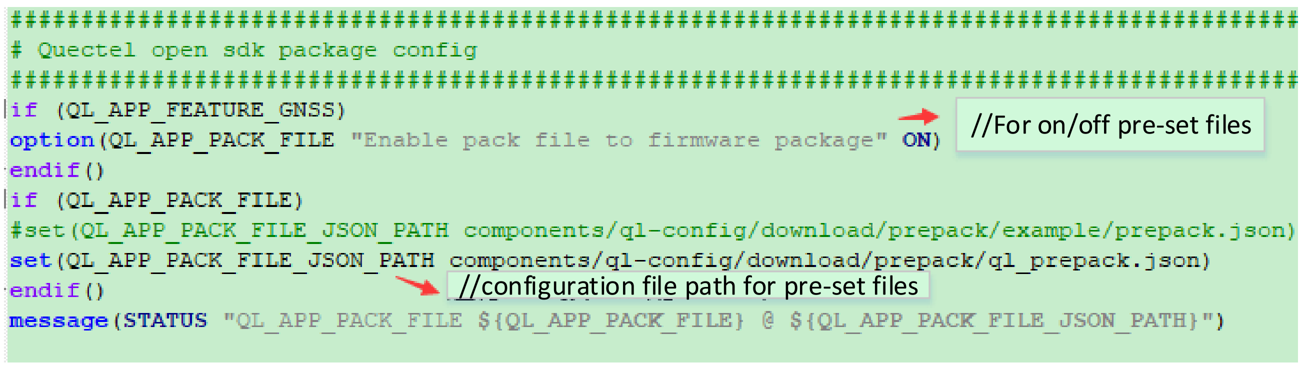
For adding your own component switch, please add it in these two files by referring to the above examples. In addition, you need to add option switch control in the top-level compilation configuration file *{ql-sdk-root}/CMakeLists.txt*, refer to ***Figure 11*** for details.

6 File Pre-Set

File pre-set refers to packaging the specified initial files in the firmware package, so that these files can be directly written into the file system of the module during firmware downloading. Therefore, the pre-set file will occupy the memory of the file system. At present, the maximum size of a single pre-set file is limited to 512 KB. If it exceeds 512 KB, this file will not be downloaded during firmware downloading.

Also, in the *ql_app_feature_config.cmake* file, you can configure whether to package the pre-set file through QL_APP_PACK_FILE. The pre-set files on/off is affected by the GNSS function in SDK. When the GNSS function is enabled, the GNSS chip firmware (currently with the size about 238 KB) will be packaged into the APP firmware as a pre-set file by default.

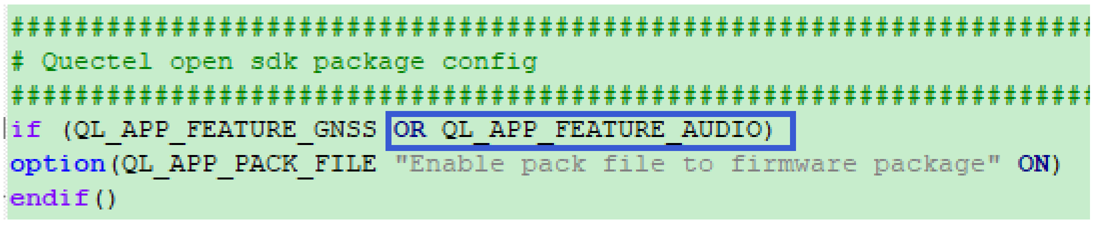
The specific configuration is as follows:



```
#####################################################################################
# Quectel open sdk package config
#####################################################################################
if (QL_APP_FEATURE_GNSS)
option(QL_APP_PACK_FILE "Enable pack file to firmware package" ON)   //For on/off pre-set files
endif()
if (QL_APP_PACK_FILE)
#set(QL_APP_PACK_FILE_JSON_PATH components/ql-config/download/prepack/example/prepack.json)
set(QL_APP_PACK_FILE_JSON_PATH components/ql-config/download/prepack/ql_prepack.json)
endif()   //configuration file path for pre-set files
message(STATUS "QL_APP_PACK_FILE ${QL_APP_PACK_FILE} @ ${QL_APP_PACK_FILE_JSON_PATH}")
```

Figure 14: Pre-Set File Package Compilation Configuration

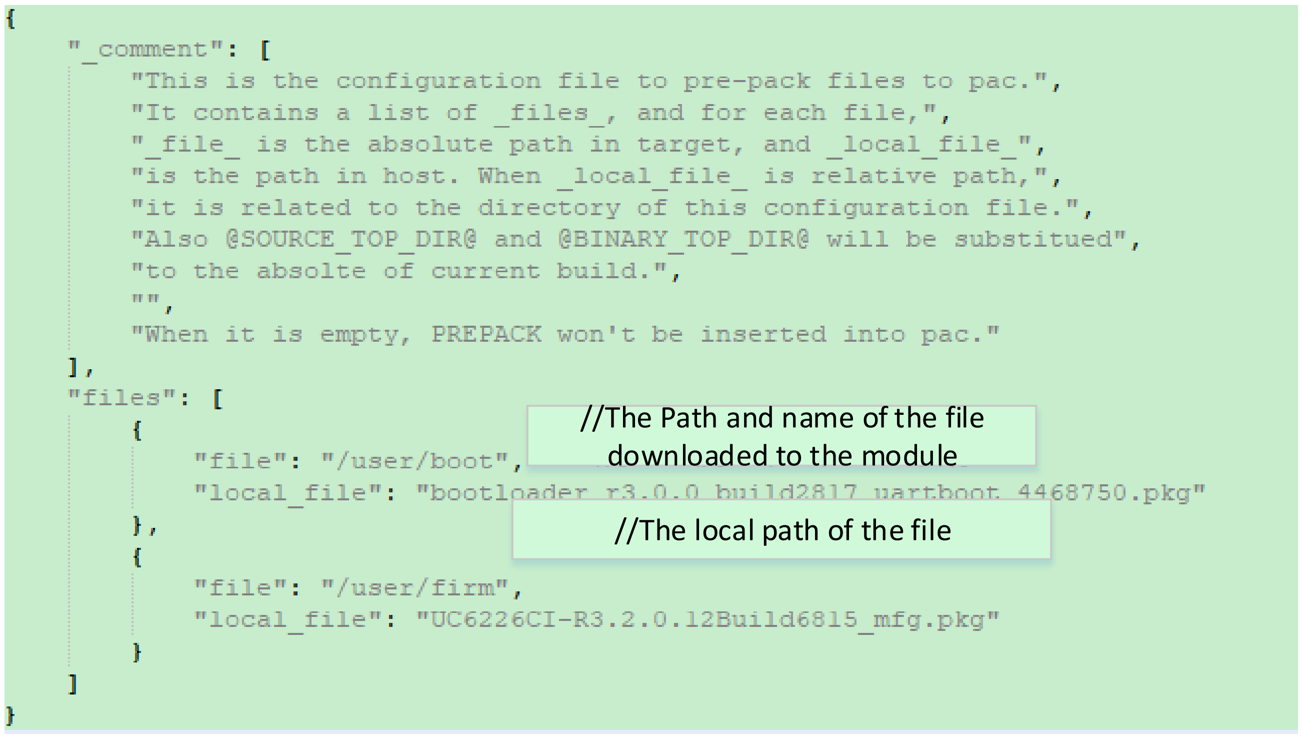
In addition to the GNSS function, if you need to preset files for other functions, for example, to pre-set certain audio files for Audio function, you can add a switch to turn on or off the audio function, as follows:



```
#####################################################################
# Quectel open sdk package config
#####################################################################
if (QL_APP_FEATURE_GNSS OR QL_APP_FEATURE_AUDIO)
option(QL_APP_PACK_FILE "Enable pack file to firmware package" ON)
endif()
```

Figure 15: Add Function Switches for Pre-set Files

QL_APP_PACK_FILE_JSON_PATH specifies the path of the specific configuration of the pre-set file. The script will package the pre-set file according to the specified *json* file. The *json* file in SDK specifies the path (represented by a relative path) of the GNSS chip firmware, and the path same as the path to store the *json* file. You can refer to this format in the *json* file to add and modify in the "files" node in the figure below. Be careful not to preset too many files to avoid taking up too much file system space.



```
{
    "_comment": [
        "This is the configuration file to pre-pack files to pac.",
        "It contains a list of _files_, and for each file,",
        "_file_ is the absolute path in target, and _local_file_",
        "is the path in host. When _local_file_ is relative path,",
        "it is related to the directory of this configuration file.",
        "Also @SOURCE_TOP_DIR@ and @BINARY_TOP_DIR@ will be substitued",
        "to the absolte of current build.",
        "",
        "When it is empty, PREPACK won't be inserted into pac."
    ],
    "files": [
        {
            "file": "/user/boot",
            "local_file": "bootloader_r3.0.0_build2817_uartboot_4468750.pkg"
        },
        {
            "file": "/user/firm",
            "local_file": "UC6226CI-R3.2.0.12Build6815_mfg.pkg"
        }
    ]
}
```

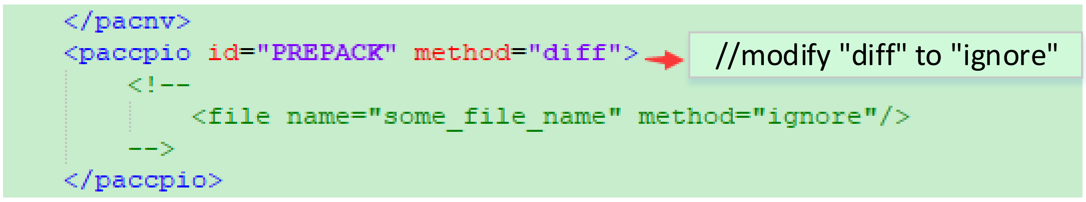
//The Path and name of the file downloaded to the module
//The local path of the file

Figure 16: Pre-Set File Package Configuration Files

The *user* directory in the module generally stores the files downloaded, so that the files can be queried through EC200U-CN FILE related APIs. Please note that the total file path length and file name to be written should not exceed 192 bytes.

The pre-set file can also be downloaded or upgraded through FOTA. The following conditions are supported:

1. Upgrade the pre-set file in the old firmware version to that of in the new firmware version.
2. If the old firmware version does not have a pre-set file, you can add a pre-set file through FOTA upgrade;
3. If the old firmware version has pre-set files and the new firmware version does not have a pre-set file, the pre-set files will be deleted after FOTA upgrade;
4. If you do not need to upgrade the pre-set file during FOTA, please modify the *xml* configuration file that is used to make FOTA package: modify **"diff"** in <paccpio id="PREPACK" method="diff"></paccpio> to **"ignore"**. You can also refer to *Quectel_EC200U-CN_QuecOpen_FOTA_API_Reference Manual* for details.



```
</pacnv>
<paccpio id="PREPACK" method="diff">
    <!--
        <file name="some_file_name" method="ignore"/>
    -->
</paccpio>
```

//modify "diff" to "ignore"

7 Appendix

Table 2: Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
APP	Application
GCC	GNU Compiler Collection
OS	Operating System
SDK	Software Development Kit
USB	Universal Serial Bus