Android DFU SDK Integration Guide

V2.0

2022/10/10

Revision History

Date	Version	Comments	Author	Reviewer
2018/04/04	V1.0	Initial	nat_zhang	Jason
2022/08/02	V2.0	Reconstruction	nat_zhang	Annie

Contents

Table List

Figure List

Glossary

Terms	Definitions
OTA	Over the air
DFU	Device Firmware Update

1 Import SDK

1.1 Libraries

1. Copy jar libraries into the project's directory with the same name.

```
|-- libs/
    //Mandatory, responsible for managing bluetooth communication
    |-- rtk-core-1.2.17.jar
    //Mandatory, responsible for transmitting OTA data
    |-- rtk-dfu-3.5.2.jar
```

2. For ***SPP*** Channel, users should also add another library aditional.

```
|-- libs/
    //Optional for Headset/SPP
    |-- rtk-bbpro-core-1.7.1.jar
```

3. For ***USB*** Channel, users should also add another library aditional.

```
|-- libs/
    //Optional for USB
    |-- rtk-usb-1.0.7.jar
```

1.2 JNI SO

Copy SO libraries into project's directory, the structure of *jniLibs* folder should be like as follows.

libRtkAesJni.so is used to encrypt data in transit.

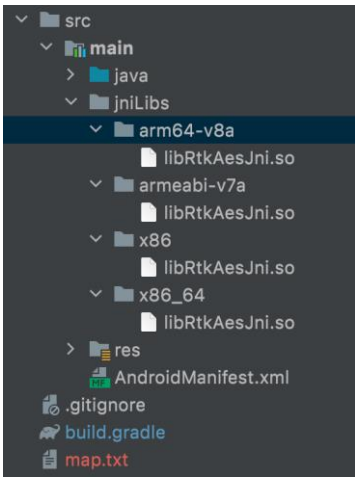

Figure 1-1 JNI SO

Generally, it only needs to support mobile phones, so users can configure it. As shown below:

```
android {
    defaultConfig {
        //...
        ndk {
            // Specifies the ABI configurations of your native
            // libraries Gradle should build and package with your APK.
            abiFilters 'armeabi-v7a', 'arm64-v8a'
        }
    }
}
```

1.3 Service

It still needs to decalre the service in the app manifest file, as shown in the following XML snippet:

```
<manifest>
    <application>
    <!-- declare DFU service -->
    <service
          android:name="com.realsil.sdk.dfu.DfuService"
          android:enabled="true"
          android:exported="false" />
    </application>
</manifest>
```

2 Permissions Required

Before using OTA functions, app must request the appropriate permissions. Add necessary permissions and service into **AndroidManifest.xml.**

2.1 Storage

Mandatory

Permissions of storage are used to access and read image bin file.

```
<manifest>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
</manifest>
```

2.2 Bluetooth

Optional

Used for Bluetooth Device only, transfer data over GATT or SPP channel.

```
<manifest>
<uses-feature
      android:name="android.hardware.bluetooth_le"
      android:required="true" />

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <!-- Needed only if your app looks for Bluetooth devices.You must add an attribute to this permission, or declare the ACCESS_FINE_LOCATION permission, depending on the results when you check location usage in your app. -->
    <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />

    <!-- Needed only if your app communicates with already-paired Bluetooth devices. -->
    <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />

</manifest>
```

2.3 Location

Optional

Used for Bluetooth Device only. App declare and request these permissions to scan and connect Bluetooth device.

```
<manifest>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
</manifest>
```

Since ACCESS_FINE_LOCATION is considered to be dangerous system permissions, in addition to adding them to the manifest, users must request this permission at runtime, as described in Requesting Permissions.

2.4 USB

Optional

Used for USB Device only.

```
<manifest>
    <uses-feature android:name="android.hardware.usb.host" />
    <uses-permission android:name="android.hardware.usb.accessory" />
</manifest>
```

2.5 Service

Declare the ***android.permission.FOREGROUND_SERVICE*** permission to make sure the DFU service can be run in the foreground.

```
<manifest>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
</manifest>
```

3 Initialize SDK

Initialize the dependence libraries.

```
// Mandatory, initialize rtk-core library
// this: context
// isDebug: true, switch on debug log; false, switch off debug log
val configure = RtkConfigure.Builder()
        .debugEnabled(isDebug)
        .printLog(true)
        .logTag("OTA")
        .globalLogLevel(ZLogger.INFO)
        //.globalLogLevel(DebuggerSettings.getInstance()!!.debugLevel)
        .build()
RtkCore.initialize(this, configure)

// Optional for demo
RtkSupport.initialize(this, true)

// Mandatory, initialize rtk-dfu library
// this: context
// isDebug: true, switch on debug log; false, switch off debug log
RtkDfu.initialize(this, isDebug)
```

4 DFU Procedure

4.1 Create DFU Adapter

Create a reference to the instance of the *DfuAdapter* as follows.

4.1.1 HID

It is used for HID device (like RCU).

```
1. val mDfuAdapter = HidDfuAdapter.getInstance(this)
```

4.1.2 GATT

It is used for BLE device.

```
1. val mDfuAdapter = GattDfuAdapter.getInstance(this)
```

4.1.3 Headset

It is used for headset scenario.

```
1. val mDfuAdapter = HeadsetDfuAdapter.getInstance(this)
```

4.1.4 SPP

It is used for Bluetooth Legacy Device.

```
1. val mDfuAdapter = SppDfuAdapter.getInstance(this)
```

4.1.5 USB

It is used for USB Dongle device.

```
1. val mDfuAdapter = UsbGattDfuAdapter.getInstance(this)
```

4.2 Initialize DfuAdapter

Initialize ***DfuAdapter*** and set a callback which implements ***DfuAdapter.DfuHelperCallback*** class. This is an asynchronous operation, when ***DfuAdapter*** initialize success, *onStateChanged(int state)* callback will be invoked. Note: It should wait until the state change to STATE_INIT_OK.

```
1. mDfuAdapter.initialize(mDfuHelperCallback)
2.
3. private final DfuAdapter.DfuHelperCallback mDfuHelperCallback = new DfuAdapter.DfuHelper
Callback() {
4.        @Override
5.        public void onStateChanged(int state) {
6.           super.onStateChanged(state);
7.
8.           if (state == DfuAdapter.STATE_INIT_OK) {
9.             // indidates DFU SDK is ready
10.              // TODO …
11.           }
12.       }
13. }
```

4.3 Connect Device

Before starting OTA, it may need to establish a connection to a remote device to get the peer device in fo. A ***DfuHelperCallback#onStateChanged(int)*** callback will be invoked when the connection state chang es as a result of this function. When the state change to ***DfuAdapter.STATE_PREPARED***, you can use ***mDfuAdapter.otaDeviceInfo*** to get the device info.

4.3.1 Set Connect Parameters

```
1. public void connectRemoteDevice(BluetoothDevice bluetoothDevice) {
2.
3.        val connectParamsBuilder = ConnectParams.Builder()
4.              .address(bluetoothDevice.getAddress())
5.              .reconnectTimes(3)
6.              .batteryValueFormat(ConnectParams.BATTERY_VALUE_F1)
7.
8.        val ret = dfuAdapter.connectDevice(connectParamsBuilder.build())
9.        if (!ret) {
10.         }
11.    }
```

4.3.2 Establish Connection

```
mDfuAdapter().connectDevice(connectParamsBuilder.build());

    private val mDfuHelperCallback = object : DfuAdapter.DfuHelperCallback() {
        override fun onStateChanged(state: Int) {
            super.onStateChanged(state)
            runOnUiThread {
                if (state == DfuAdapter.STATE_PREPARED) {

                }
                 else if (state == DfuAdapter.STATE_DISCONNECTED || state == DfuAdapter.STATE_CONNECT_FAILED) {

                }
            }
        }

        override fun onError(type: Int, code: Int) {

            runOnUiThread {
                cancelProgressBar()
                val message = DfuHelperImpl.parseError(applicationContext, type, code)
            }
        }

    }
```

4.3.3 Device Info

When the connection established, you can retrieve the device info use below APIs.

```
var mOtaDeviceInfo = mDfuAdapter.otaDeviceInfo
```

- **OTA HEADER**

```
mOtaDeviceInfo.getOtaHeaderImageVersion()
```

- **APP UI Parameters**

```
mOtaDeviceInfo.getAppUiParameterVersion()
```

- **APP**

```
mOtaDeviceInfo.getAppVersion()
```

- **PATCH**

```
1. mOtaDeviceInfo.getAppPatchVersion()
```

4.4 Setup DFU Configurations

This section describes the usage of DFU configuration options in the DFU procedure. Please make sure that set the right DFU configuration first before you start OTA procedure.

```
1. val mDfuConfig = DfuConfig()
```

4.4.1 Mandatory Configurations

4.4.1.1 Channel Type

Mandatory

The default value is *DfuConfig.CHANNEL_TYPE_GATT*, users can set it to the value as shown below:

```
1. mDfuConfig.setChannelType(DfuConfig.CHANNEL_TYPE_GATT)
```

Table 4-1 Channel Type

Field	Value	Description
DfuConfig.CHANNEL_TYPE_GATT	0x00	used for BLE device, like RCU and etc.
DfuConfig.CHANNEL_TYPE_SPP	0x01	used for SPP device, like headset and etc.
DfuConfig.CHANNEL_TYPE_USB	0x02	used for dongle device

4.4.1.2 Device Address

Mandatory

Set the device address or device name to establish connection during OTA.

If transfer OTA data over BLE/SPP, it should set BD_ADDR of the device.

If transfer OTA data over USB (Dongle device), it should set the name of the USB device.

```
1. //Bluetooth Device
2. mDfuConfig.setAddress(bluetoothDevice.getAddress());
3. //USB Device
4. mDfuConfig.setAddress(usbDevice.getDeviceName());
```

4.4.1.3 Image File Path

Mandatory

Used to set the bin file path, for sdcard, it should set the full path, format like: */storage/emulated/0/xxx.bin*.

Note: it should double confirm that the App can access this path. For assets file, it should set the value like: *images/xxx.bin*, images is the assets path and set the file location also.

```
1. mDfuConfig.setFilePath(mFilePath)
```

Table 4-2 File Location

Field	Value	Description
DfuConfig.FILE_LOCATION_SDCARD	0x00	used for SD card file
DfuConfig.FILE_LOCATION_ASSETS	0x01	used for asset file

4.4.1.4 Work Mode

Mandatory

Each device needs to specify a work mode, which may be different for different devices.

If you know the exactly work mode, you can set it directly as shown below:

```
1. mDfuConfig.setOtaWorkMode(workMode)
```

If you don't know which work mode the device supports, you can query the priority work mode through the following API.

```
1.  val modeInfo = mDfuAdapter.getPriorityWorkMode(DfuConstants.OTA_MODE_SILENT_FUN
CTION);
2.  getDfuConfig().setOtaWorkMode(modeInfo.getWorkmode());
3.  mDfuConfig.setOtaWorkMode(modeInfo.getWorkmode)
```

Note: you should call connectDevice method first to establish a connection, otherwise, it will return the default value.

A device may support multiple work modes. You can choose the appropriate work mode according to your needs.

```
1. val otaModeInfos = getDfuAdapter().getSupportedModes();
```

4.4.2 Recommend Configurations

4.4.2.1 Protocol Type

Optional

The default value is 0, it is used to identify different versions of the device. if you didn't pass the DeviceInfo to startOtaProcedure API, you should add this below code.

```
if (mOtaDeviceInfo != null) {
            getDfuConfig().setProtocolType(mOtaDeviceInfo.getProtocolType());
        } else {
            getDfuConfig().setProtocolType(0);
        }
```

4.4.3 Optional Configurations

4.4.3.1 Automatic Active Image

Optional

The default value is true.When you set the value to false, you will receive a status *PROGRESS_PENDING_ACTIVE_IMAGE* after the image file is sent. You can choose whether to activate firmware here.

```
mDfuConfig.setAutomaticActiveEnable(false)

    private val mDfuHelperCallback = object : DfuAdapter.DfuHelperCallback() {
        override fun onProcessStateChanged(state: Int, throughput: Throughput?) {
            super.onProcessStateChanged(state, throughput)
            if (state == DfuConstants.PROGRESS_PENDING_ACTIVE_IMAGE) {
                //set true to active image and reset
                //getDfuAdapter().activeImage(true)
            }

        }

    }
```

4.4.3.2 Battery Level Check

Optional

It is disabled by default. You can call *setBatteryCheckEnabled* to enable this. As shown below:

```
1. //enable battery level check feature
2. mDfuConfig.setBatteryCheckEnabled(true);
3. //set the threshold value
4. mDfuConfig.setLowBatteryThreshold(30);
```

Note: If you enable this feature, the battery level must be higher that the threshold value to upgrade.You can also disable this feature, and check in your App before OTA.

4.4.3.2.1 Battery Value Parse Format

The battery value size is 1 byte, indicate battery percent. if you customized this value to voltage value, and size change to 2 bytes, you should set the ***ConnectinParams*** when you call the ***connectDevice*** API.

```
1. val connectParamsBuilder = ConnectParams.Builder()
2.         //optional, the default value is ConnectParams.BATTERY_VALUE_F1
3.         .batteryValueFormat(ConnectParams.BATTERY_VALUE_F1)
```

Table 4-3 Battery Value Parse Format

Field	Value	Description
ConnectParams.BATTERY_VALUE_F1	0x00	default, support only 1 byte
ConnectParams.BATTERY_VALUE_F2	0x01	support 1 or 2 bytes

4.4.4 Image Version Check

Optional

It is enabled by default. You can call *setVersionCheckEnabled* to disable it. As shown below:

```
1. mDfuConfig.setVersionCheckEnabled(false);
```

Note: If you enable this feature, the low version and the same version will be automatically filtered out, and only the higher version files will be upgrade. If no files that can be upgraded are detected, an error will be prompted.

4.4.4.1 Check Policy

There are two policies: **strict & loose.**

The default policy is strict mode, for some special situations, you can change the check policy. As shown below:

```
1. mDfuConfig.setVersionCheckMode(1);
```

Table 4-4 Version Check Policy

Value	Description
0x00	default, strict mode
0x01	loose mode

4.4.5 Customized Configurations

4.4.5.1 Secret Key

The secret key (32 bytes) is used to encrypt the packet during OTA, you can call *setSecretKey* to configure this. As shown below:

```
1. private val SECRET_KEY = byteArrayOf(
2.         0x4E.toByte(),
3.         0x46.toByte(),
4.         0xF8.toByte(),
5.         0xC5.toByte(),
6.         0x09.toByte(),
7.         0x2B.toByte(),
8.         0x29.toByte(),
9.         0xE2.toByte(),
10.        0x9A.toByte(),
11.        0x97.toByte(),
12.        0x1A.toByte(),
13.        0x0C.toByte(),
14.        0xD1.toByte(),
15.        0xF6.toByte(),
16.        0x10.toByte(),
17.        0xFB.toByte(),
18.        0x1F.toByte(),
19.        0x67.toByte(),
20.        0x63.toByte(),
21.        0xDF.toByte(),
22.        0x80.toByte(),
23.        0x7A.toByte(),
24.        0x7E.toByte(),
25.        0x70.toByte(),
26.        0x96.toByte(),
27.        0x0D.toByte(),
28.        0x4C.toByte(),
29.        0xD3.toByte(),
30.        0x11.toByte(),
```

```
        0x8E.toByte(),
        0x60.toByte(),
        0x1A.toByte()
    )
getDfuConfig().setSecretKey(SECRET_KEY);
```

Note: This key must be consistent with the SOC.

4.4.5.2 Customized OTA Service UUID

Optional

If you want to use your customized UUID with device, you should add this below code.

```
val OTA_SERVICE = UUID.fromString("0000xxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
getDfuConfig().setOtaServiceUuid(OTA_SERVICE);
```

Note: Please make sure that using the same UUID as your device.

4.4.6 Developer Configurations

4.4.6.1 Breakpoint Resume

Optional

It is disabled by default. You can call *setBreakpointResumeEnabled* to enable this. As shown below:

```
getDfuConfig().setBreakpointResumeEnabled (true);
```

Note: Only when the device supports this feature, the API will be effective.

4.4.6.2 Flow Control

Optional

Flow control is an Optional feature, which is disabled by default. You can call *setFlowControlEnabled* to enable it. As shown below:

```
getDfuConfig().setFlowControlEnabled (true);
```

When you enable the Flow Control feature, you should also call *setFlowControlInterval* to set the interval value. As shown below:

```
getDfuConfig().setFlowControlInterval (1);
```

Note: The unit of flow control interval is 50 *ms*, for example, if you set the interval to 2, the real interval between packet will be 2*50=100 (*ms*)

4.4.6.3 Connection Parameters

Optional

You can change it before OTA. As shown below:

```
dfuConfig.connectionParameters = ConnectionParameters.Builder()
        .maxInterval(0x0006)
        .minInterval(0x0011)
        .latency(0)
        .timeout(500)
        .build()
```

Table 4-5 Connection Parameters

Field	Value	Description
minInterval	0x0006	Connection interval used on this connection, 1.25ms unit. Valid range is from 6 (7.5ms) to 3200 (4000ms)
maxInterval	0x0011	Connection interval used on this connection, 1.25ms unit. Valid range is from 6 (7.5ms) to 3200 (4000ms)
latency	0x0000	Slave latency for the connection in number of connection events. Valid range is from 0 to 499
timeout	500	Supervision timeout for this connection, in 10ms unit. Valid range is from 10 (0.1s) to 3200 (32s)

4.4.6.4 Retrans Connect Times

Optional

It is used to set the retrans connect times when connection establish failed, the default value is 3, You can call *setRetransConnectTimes* to change it. As shown below:

```
dfuConfig.retransConnectTimes = 1
```

4.4.6.5 Image Section Size Check

Optional

Image section size check is an optional feature, which is enable by default. Image file can't be loaded if the image size exceed the section size of the SOC. You can call *setSectionSizeCheckEnabled* to disable this. As shown below:

```
1. mDfuConfig. setSectionSizeCheckEnabled(false);
```

Note: If you disable this feature, it will ignore the section size limit.

4.4.6.6 Chip type Check

Optional

It is enabled by default.When each bin file is generated or packaged, there will be a chip type, which is used to distinguish different IC upgrades. Before the upgrade, the chip type of the file and the information of the device will be compared, and the upgrade can only be done when they are the same, otherwise an error will be prompted.You can call *setIcCheckEnabled* to disable this. As shown below:

```
1. mDfuConfig. setIcCheckEnabled (false);
```

Note: If you disable this feature, it may upload a unmatched file.

4.4.6.7 Buffer Check MTU Size

Optional

It is enabled by default. You can call *setBufferCheckMtuUpdateEnabled* to disable it. As shown below:

```
1. mDfuConfig. setBufferCheckMtuUpdateEnabled (false);
```

4.4.6.8 Notification Timeout

Optional

Notification timeout indicates that the max timeout of the response of the command sent. This is an optional feature, you can configure the timeout to compatible with different devices. As shown below:

```
1. // unit is millis, range from 0~60*1000
2. getDfuConfig(). setNotificationTimeout (10*1000);
```

4.4.6.9 Active and Reset delay time

Optional

The default value is 0. set this to control the delay time before send ACTIVE_AND_RESET command.

```
1. getDfuConfig(). setActiveImageDelayTime (10*1000);
```

Note: Used for BLE only.

4.4.6.10 Active and Reset ACK

Optional

The default value is false. In normal case, If not received response will be considered successful. When you set this value to true, phone app must wait for response, if not received will report an error 0x0105.

```
1. mDfuConfig. setWaitActiveCmdAckEnabled (true);
```

4.4.7 Others

4.4.7.1 File Suffix

```
1. mDfuConfig. setFileSuffix ("bin");
```

4.5 Start OTA Procedure

Call *mDfuAdapter.startOtaProcedure(mDfuConfig)* to start OTA procedure.

If any error happens, *onError* callback will be invoked.

You should input two parameters DeviceInfo and DfuConfig.

```
1. boolean ret = mDfuAdapter.startOtaProcedure(mOtaDeviceInfo, mDfuConfig);
2.         if (!ret) {
3.             // TO DO ...
4.         }
5.
6.         private final DfuAdapter.DfuHelperCallback mDfuHelperCallback = new DfuAdapter.DfuH
elperCallback() {
7.             @Override
8.             public void onError(int type, int code) {
9.              // Something error happens
10.                // TO DO ...
11.
12.             }
13.
14.             @Override
15.             public void onProcessStateChanged(int state, Throughput throughput) {
16.                 super.onProcessStateChanged(state, throughput);
```

```
17.
18.             if (state == DfuConstants.PROGRESS_IMAGE_ACTIVE_SUCCESS) {
19.                 //ota procedure complete and success
20.             } else if (state == DfuConstants.PROGRESS_START_DFU_PROCESS) {
21.                 //ota processing
22.             } else if (state == DfuConstants.PROGRESS_STARTED) {
23.                 //start to ota
24.             }
25.
26.         }
27.
28.         @Override
29.         public void onProgressChanged(DfuProgressInfo dfuProgressInfo) {
30.             super.onProgressChanged(dfuProgressInfo);
31.             //refresh UI with the dfuProcessInfo
32.             //dfuProgressInfo.getTotalProgress()
33.         }
34.     };
```

4.5.1 Progress State

Table 4-6 Progress State

Field	Description
DfuConstants.PROGRESS_STARTED	start to OTA
DfuConstants.PROGRESS_START_DFU_PROCESS	OTA processing
DfuConstants.PROGRESS_IMAGE_ACTIVE_SUCCESS	OTA procedure complete and success

4.6 Abort OTA Procedure

Optional

If you want to cancel the OTA procedure, you can call this API:

```
1. Val ret = mDfuAdapter.abort();
2. if (!ret) {
3.    //TODO(FAIL)
4. } else {
5.    // TODO(success), wait the onError callback
6. }
7.
8. private final DfuAdapter.DfuHelperCallback mDfuHelperCallback = new DfuAdapter.DfuHelper
Callback() {
```

```
9.          @Override
10.          public void onError(int type, int code) {
11.          // Something error happens
12.          // TO DO ...
13.
14.          }
15.
16.      };
```

4.7 Release

Generally, this API is called when your application exits or closes the service or activity.

```
1. if (mDfuAdapter != null) {
2.    mDfuAdapter.abort()
3.    mDfuAdapter.close()
4. }
```

5 Bin Image Information

5.1 Load Bin Info

Optional

Load the image file info. If you want to show image info in your own UI, you can refer to bellow codes:

Sdcard file

```
1. LoadParams.Builder builder = new LoadParams.Builder()
2.       //mandatory, your context
3.       .with(this)
4.       //mandatory, format like :"/storage/emulated/0/xxx.bin"
5.       .setFilePath(mFilePath)
6.       .setFileSuffix("bin") //optional
7.       .setOtaDeviceInfo(mOtaDeviceInfo)//recommend
8.       .setSectionSizeCheckEnabled(false) //optional
9.       .setIcCheckEnabled(true) //optional
10.        .versionCheckEnabled(true); //optional
11. mBinInfo = FirmwareLoaderX.loadImageBinInfo(builder.build());
```

Assets file

```
1. LoadParams.Builder builder = new LoadParams.Builder()
2.       //mandatory, your context
3.       .with(this)
4.       .fileLocation(DfuConfig.FILE_LOCATION_ASSETS)
5.       //mandatory, format like :"images/xxx.bin",image is your asserts path
6.       .setFilePath(mFilePath)
7.       .setFileSuffix("bin") //optional
8.       .setOtaDeviceInfo(mOtaDeviceInfo)//recommend
9.       .setSectionSizeCheckEnabled(false) //optional
10.        .setIcCheckEnabled(true) //optional
11.        .versionCheckEnabled(true); //optional
12. mBinInfo = FirmwareLoaderX.loadImageBinInfo(builder.build());
```

5.1.1 Load Parameters

Table 5-1 Load Parameters

Field	Requirement	Description
fileLocation	M	location of the bin file
filePath	M	path of the bin file
fileSuffix	O	the suffix of the bin file. default value is ".bin"
otaDeviceInfo	O	the device info of the device
icCheckEnabled	O	indicate that if theck the IC type or not
versionCheckEnabled	O	indicate that if check the image version or not
sectionSizeCheckEnabled	O	indicate that if check the image section size or not

5.2 Image Version

- **OTA HEADER**

```
1. ZLogger.v(
2. String.format("otaHeader version:0x%04X", mOtaDeviceInfo.getOtaHeaderImageVersion()))
```

- **APP UI Parameter**

```
1. ZLogger.v(
2. String.format("appUiParameter version:0x%04X", mOtaDeviceInfo.getAppUiParameterVersion(
)))
```

- **APP**

```
1. // bin
2. mBinInfo.getAppImageVersion(bankNumber /*default value is 0*/)
3. // device
4. mOtaDeviceInfo.getAppVersion()
```

- **PATCH**

```
1. // bin
2. mBinInfo.getPatchImageVersion(bankNumber /*default value is 0*/)
3. // device
4. mOtaDeviceInfo.getPatchVersion()
```

6 Error Code

6.1 Connect Exception

Error when establish connection.

Table 6-1 Connect Exception

Constant Value	Description
0x00	establish GATT connection failed
0x01	discover OTA service failed
0x02	OTA service not supported
0x03	OTA service miss
0x04	get characteristic failed
0x05	read characteristic failed. App receive an error response from SOC
0x06	establish SPP connection failed

6.2 DFU Exception

Error when upload image.

Table 6-2 DFU Exception

Constant Value	Description
256	Can’t connect the device which enter OTA mode
257	read file exception
258	Launch the GATT search service failure
259	Invokes the Java synchronization lock exceptions
260	Send command but no callback
261	Send command but no callback
262	Service mismatch
263	Characteristic mismatch
264	establish le connection failed
265	Can’t scan the device
266	enable CCCD failed
267	write data error
268	Resend time exceed max
269	The battery is low
270	Read bank info error

Constant Value	Description
271	Read app info error
272	Read patch info error
273	Read image version failed
274	User didn't active image update
275	Buffer Check retry times exceed to MAX value
276	Request MTU failed
277	read remote mac address failed
279	send command failed
280	enter OTA mode failed
281	OTA Over SPP not supported
282	RWS OTA NOT READY
283	role swap failed
284	enable buffer check failed
285	pub keys conflict
286	hand shake failed
514	Invalid parameters
517	CRC Check failed
520	Flash erase error
766	opcode response not supported
767	Didn't receive remote notification
1027	ERROR_WRITE_NOT_PERMIT
1028	GATT_INVALID_PDU
1031	GATT_INVALID_OFFSET
1037	GATT_INVALID_ATTR_LEN
1153	GATT_INTERNAL_ERROR
1157	GATT_ERROR
2048	Connection disconnected
2056	GATT_CONN_TIMEOUT
2067	GATT_CONN_TERMINATE_PEER_USER
2088	GATT_CONN_TIMEOUT
2110	GATT_CONN_FAIL_ESTABLISH
2177	GATT_INTERNAL_ERROR
2181	establish connection failed
2305	Can't register *Client_if*
4097	load image file failed
4112	Invalid Bluetooth Address

Constant Value	Description
4113	Secret key invalid
4114	configuration invalid
4128	aborted

6.3 Load File Exception

Error when load image file.

Table 6-3 Load File Exception

Constant Value	Description
0x1001 (4097)	load image file failed
0x1002 (4098)	Image file path invalid
0x1003 (4099)	file extension invalid
0x1004 (4100)	Image file not exist
0x1005 (4101)	Image invalid