DWIN Android LCD Screen Development Guide (RK3566)

1 Product Introduction

1.1 Product Feature

DWIN Android screen 32 series:

Sizes: 5~15.6 inches

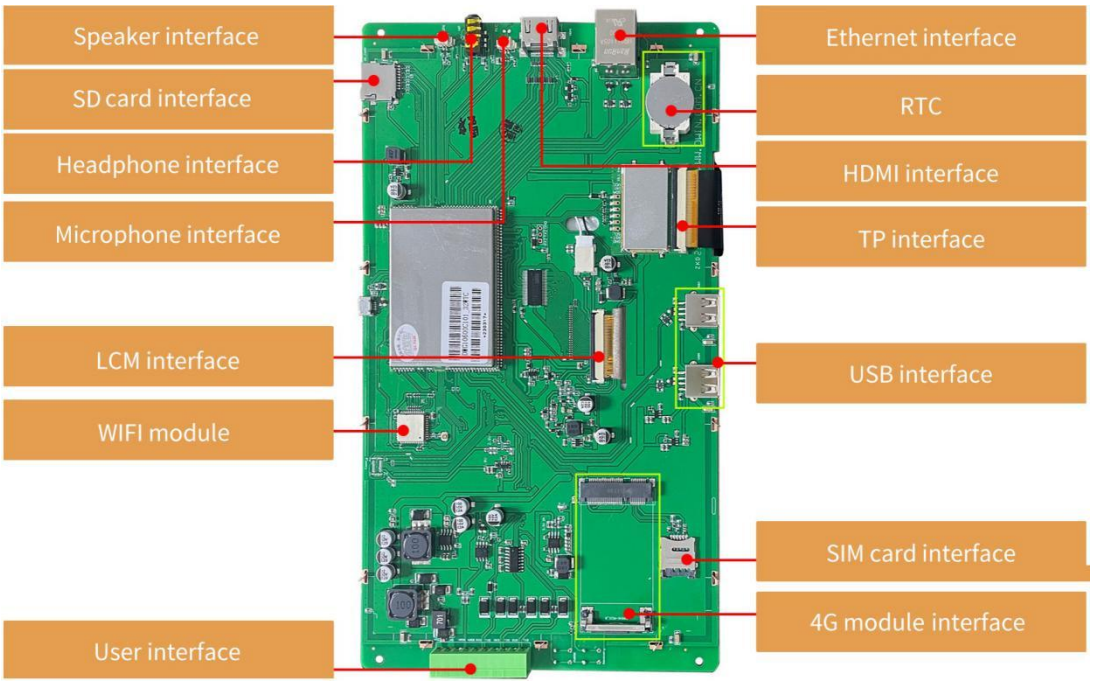
CPU: RK3566, Quad-core ARM Cortex-A55, 1.8GHz

RAM: 2GB LPDDR4, Flash: 8GB eMMC5.0 (Can be customized: 4GB LPDDR4+32GB eMMC, 8GB LPDDR4+256GB eMMC)

Android Version: Android 11

There is also an optional Google Play Trial Version (with the model suffix ZOS-2)

(DMG10600C101_32WTC front)



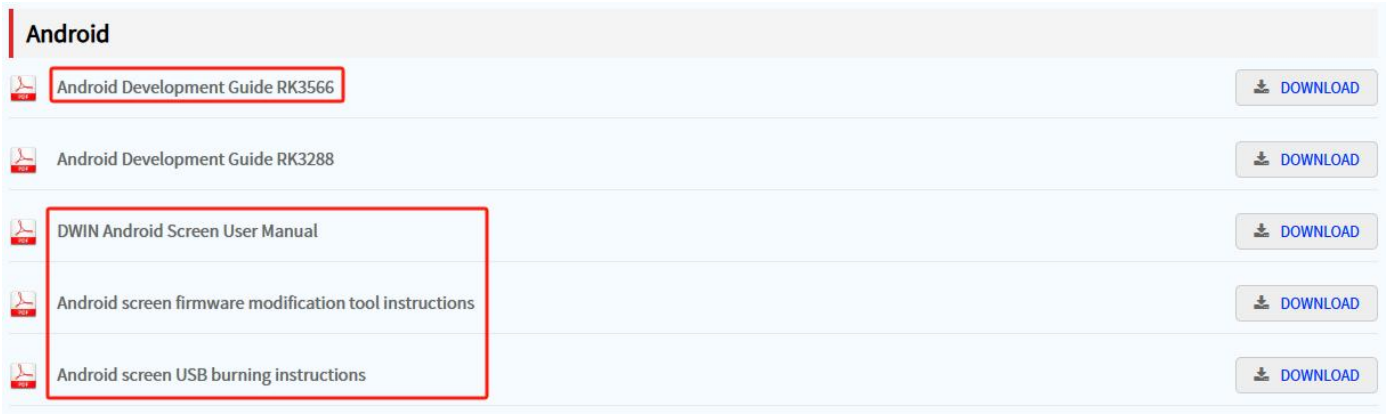
(DMG10600C101_32WTC back)

1.1.1 Development Method

Java development, or downloading existing Android programs (APKs)

1.1.2 Documentation

Documents: https: //www.dwin-global.com/development-guide/



Tool: https: //www.dwin-global.com/tool-page/

Official Site Tutorial Link:

https: //www.dwin-global.com/android/

Or YouTube Tutorial Link: https: //youtube.com/playlist?list=PLKfWyFPPaoDr3Vq98-orVxJqKA5MDaliN&si=w-o0baCtnraeGOEF

1.1.3 Shipping List (for reference)

- screen ×1 piece
- antenna × 1 piece

1.1.4 Optional Accessories

- **Speaker**

  DWIN material code B01851, cable length 180 mm, with socket 2PIN_1.25, 88±3dB, 8Ω, 0.8W

  DWIN material code B01279(louder volume), cable length 320 mm, with socket 2PIN_1.25, 8Ω, 2W.

- **SD card**

- **4G module**

  China and India: LUAT Air780EI

  Europe: QUECTEL EC200A-EU

  Australia: QUECTEL EC200A-AU

(Note: The above 4G modules can be directly used on our screen. We suggest that customers purchase the relevant modules by themselves. If your country is not included in the list, you will need to provide us with a 4G module that is available locally for us to debug before it can be used normally)

- **Camera**

  Support camera with USB interface

- **12V power adapter board**

  DWIN material code 08284: only for DMG80480T050_32WTC

1.2 Wiring

For definitions of peripheral interface and serial port, please refer to the relevant datasheet. The following image is a screenshot of the corresponding section.

Peripherals and Interfaces

Properties	Parameters	Description
COM	2-way RS232	UART5 & UART9
	1-way RS485	UART8
	1-way TTL/COMS	UART0

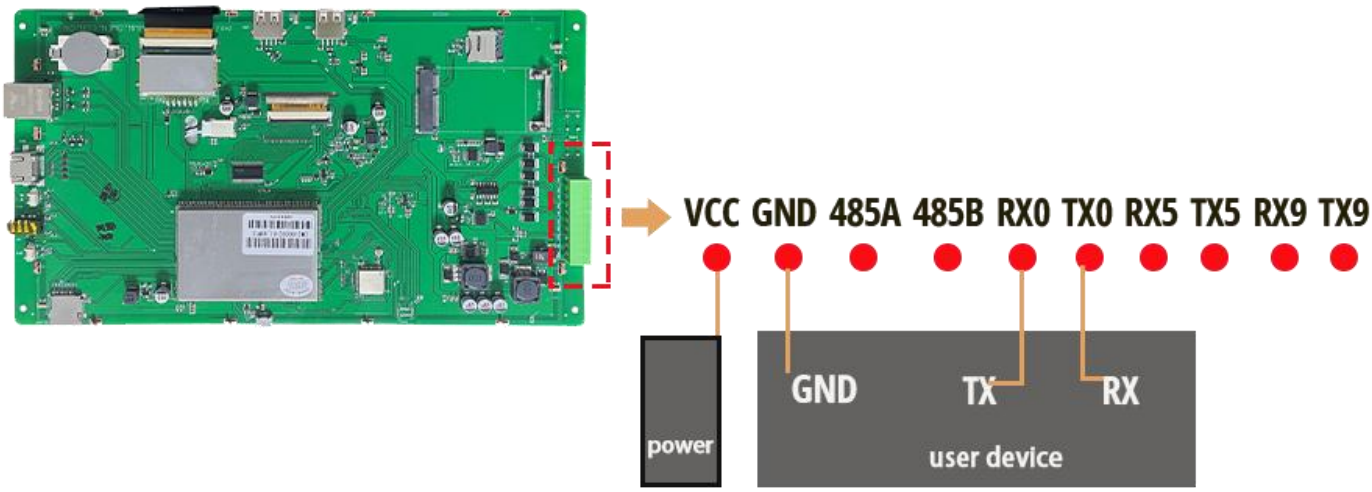
Definition	Pin#	IO	Description
VCC	1	P	Power Input
GND	2	P	GND
485A	3	A	RS485+
485B	4	B	RS485-
RX0	5	I	UART 0 Input
TX0	6	O	UART 0 Output
RX5	7	I	UART 5 Input
TX5	8	O	UART 5 Output
RX9	9	I	UART 9 Input
TX9	10	O	UART 9 Output

1.2.1 Hardware Connection

GND: Ground, connect to GND pin of the user device.

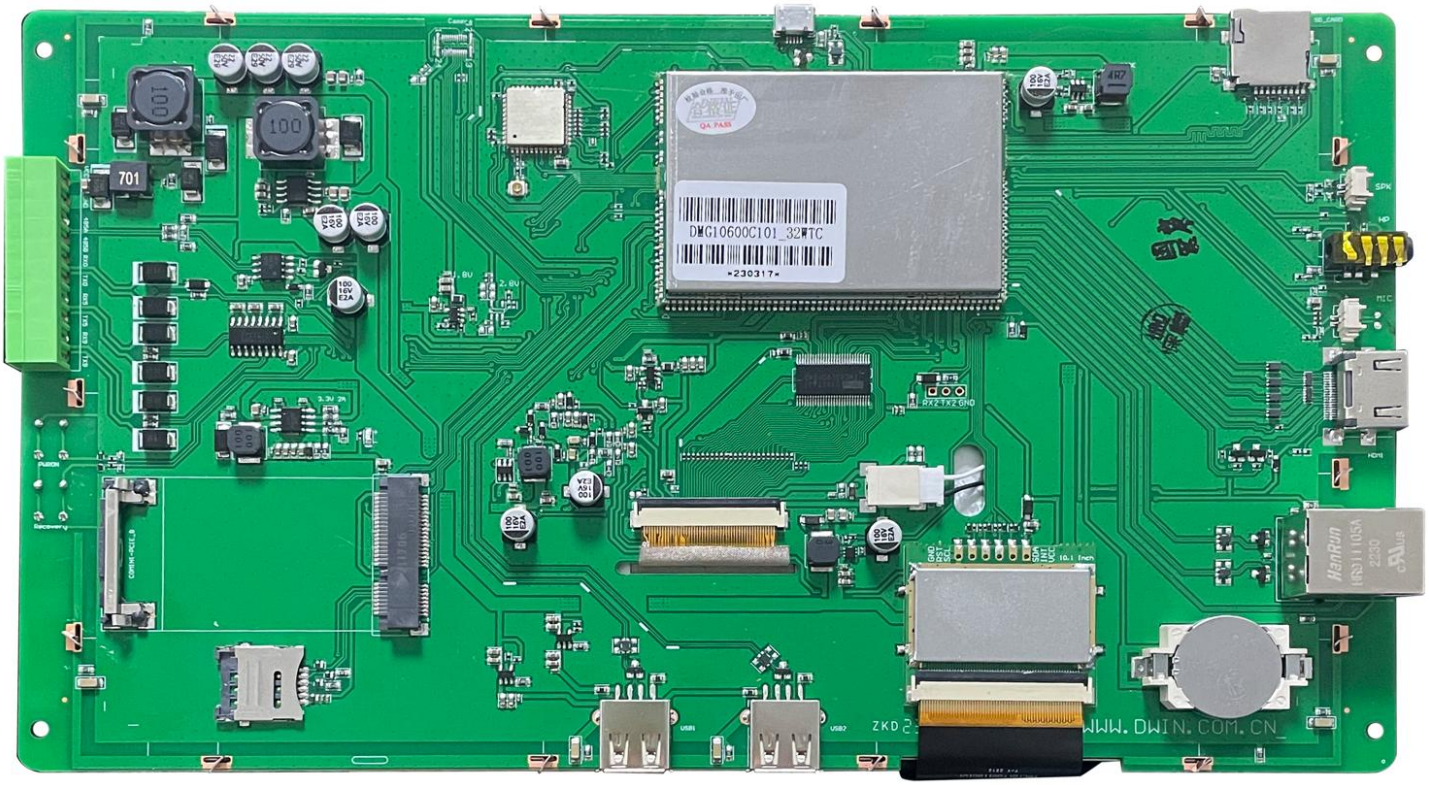
TXD: Transmit, connect to TX pin of the user device.

RXD: Receive, connect to RX pin of the user device.



1.2.2 Serial Parameter Setting

The baud rate for serial port 2 is set to 1500000, whereas the remaining serial ports are set to 115200.



1.2.3 Other Tools

We suggest using a 12V DC regulated power supply for testing and an SD card with a capacity between 1GB and 16GB for flashing the project.

1.3 Precautions

1. The standard firmware does not support the simultaneous use of keyboard input and barcode scanners when the android screen is connected to barcode scanner via USB. To use them simultaneously, please contact our sales for a customized firmware.

2. To set a static IP address, please connect the device directly to the router using an Ethernet cable. Connecting the device to a computer for setup might lead to configuration failures.

3. Ethernet-related pages follow the RK3566 official configuration, defaulting to English and will not change with the system language. To display Chinese or other languages, please contact our sales for a customized firmware.

4. An error occurred while running the DWINAndroidLibraryDemo：

```
D  Shutting down VM
E  FATAL EXCEPTION: main
   Process: com.dwin.dwinaardemo, PID: 2271
   java.lang.NoClassDefFoundError: Failed resolution of: Landroid/app/DWManager;
       at c.a.a.a.<init>(DWAndroidApi.java:27)
       at com.dwin.dwinaardemo.AndroidSDKActivity.onCreate(AndroidSDKActivity.java:35)
       at android.app.Activity.performCreate(Activity.java:8022)
       at android.app.Activity.performCreate(Activity.java:8006)
       at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1309)
       at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3404)
       at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3595)
```

Reason: The latest version of the firmware has not been flashed.

Solution: Flashing the new version of the firmware from the sales.

5. The size of the APP post-downloading onto the screen does not match that in the code.

Reason: The UI has not been designed in accordance with the screen resolution.

Solution: Design the pixel (px) and density-independent pixel (dp) in a 1:1 proportion. In case the screen density is 160, both UI designers and Android engineers can perform calculations based on the ratio of px : dp = 1:1. However, if the screen density differs from 160, Android engineers are required to convert the length proportionally in line with the actual design drawing.

6. Throw an error:

Unable to find method "void org.gradle.api.internal.DefaultDomainObjectSet.<init>(java.lang.Class)"

'void org.gradle.api.internal.DefaultDomainObjectSet.<init>(java.lang.Class)'

Gradle's dependency cache may be corrupt (this sometimes occurs after a network connection timeout.)

Reason: The versions of the development tool and Gradle are also inconsistent.

Solution: The user can create a new project with their own tool and then copy the code (such as the source code, lib libraries, and dependent libraries) over.

2 Firmware Modification

Please refer to Chapter 7 which introduces the new firmware, to upgrade from an older firmware version, please contact the sales or technical support team to acquire the new firmware.

Firmware Notes:

Old firmware tool: FWFactoryTool_RK3566_v2.3

New firmware tool: FWFactoryTool_RK3566_v2.4

2.1 Tools/Resource

(1) The extracted contents of the FWFactoryTool_RK3566_v2.3 zip file, located in the Android resource directory, are displayed in the fig. below.

名称	修改日期	类型	大小
Bin	2023/5/6 11:46	文件夹	
FileContexts	2023/5/6 11:46	文件夹	
imageformats	2023/5/6 14:01	文件夹	
Language	2023/12/18 13:30	文件夹	
Licenses	2023/5/6 14:41	文件夹	
platforms	2023/5/6 14:01	文件夹	
Rockchip	2023/5/6 11:51	文件夹	
config.ini	2023/5/6 15:07	配置设置	1 KB
FWFactoryTool.exe	2023/12/18 13:29	应用程序	439 KB
icudt51.dll	2013/4/23 17:50	应用程序扩展	21,794 KB
icuin51.dll	2013/4/23 17:49	应用程序扩展	1,759 KB
icuuc51.dll	2013/4/23 17:49	应用程序扩展	1,305 KB
libEGL.dll	2013/7/2 20:59	应用程序扩展	42 KB
libGLESv2.dll	2013/7/2 20:59	应用程序扩展	682 KB
msvcp110.dll	2012/11/6 2:20	应用程序扩展	523 KB
msvcr110.dll	2022/6/9 9:38	应用程序扩展	855 KB
Qt5Core.dll	2022/6/8 13:56	应用程序扩展	3,774 KB
Qt5Gui.dll	2013/7/2 21:05	应用程序扩展	2,981 KB
Qt5Widgets.dll	2013/7/2 21:08	应用程序扩展	4,197 KB

Log folder: Stores the log files created while the tool is running.

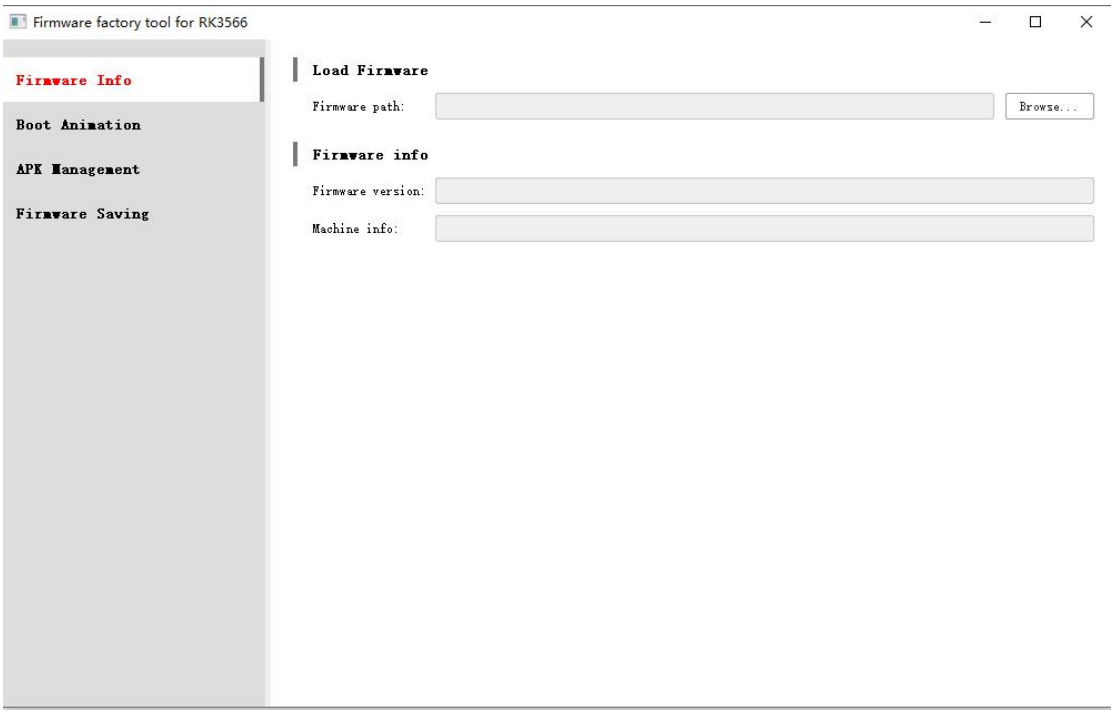
Temp folder: This folder contains files extracted from the unpacked firmware update.img. The subsequent operations of the tool involve editing the files in the Temp folder and then packaging them to generate a modified update.img file.

FWFactoryTool.exe: This is the application for firmware modification.

(2) update.img: Original firmware. Please request from the R&D team according to the model and manufacturing date to obtain this firmware. All the following operations are performed based on this firmware.

2.2 Overview of the Tool Interface

2.2.1 Start-up Interface



The image above shows the tool's startup interface. To select the update.img file, click the 'Browse' button located in the upper right corner. This firmware serves as the base image for device flashing and user modification.

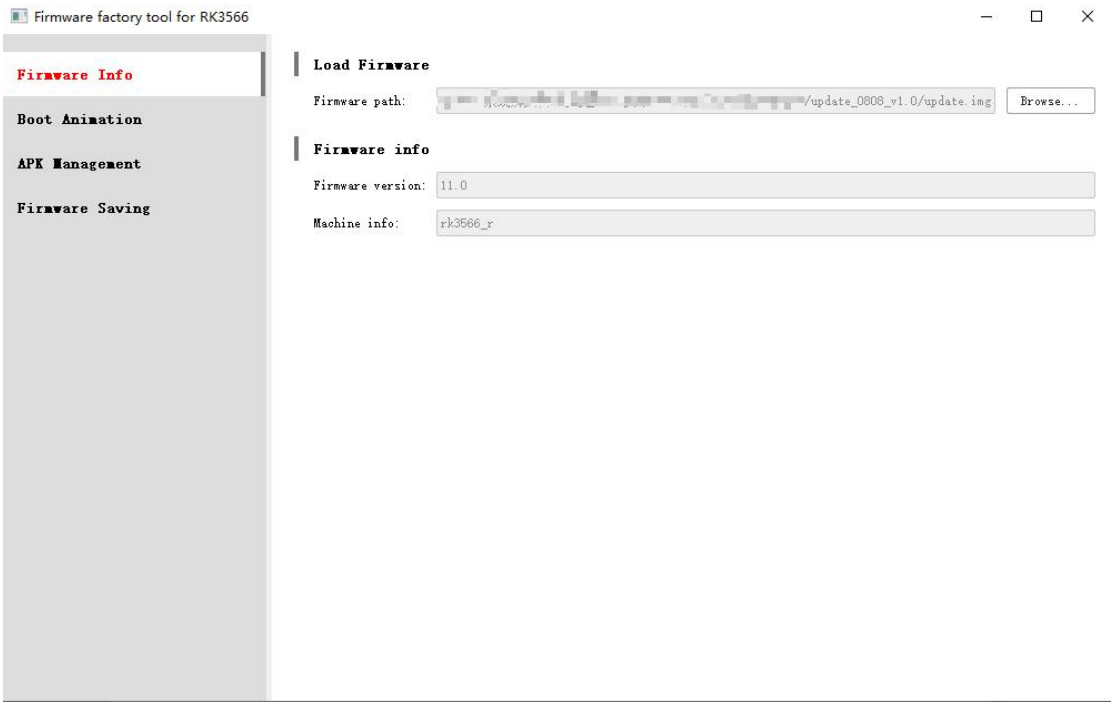
While loading, the tool begins to unpack the firmware, extracting the corresponding images to the Temp folder in the tool's root directory.

Depending on the size of the firmware and the configuration of the computer, the unpacking time may vary. Generally, it takes about 5 minutes to complete the unpacking process.

It's important to note that since the tool invokes various third-party programs, some of which might not fully support directories with special characters, this could affect the unpacking process of the images. Therefore, it is recommended to place the entire firmware tool in a relatively simple path that does not include Chinese

characters (for example, D: \FWFactoryTool\) and then run the firmware modification tool program.

2.2.2 Firmware Loaded Successfully Interface



Above is the interface after successfully extracting the update.img firmware.

While loading, the tool begins to unpack the firmware, extracting the corresponding image to the Temp folder in the tool's root directory. The unpacking time varies depending on the size of the firmware and the configuration of the computer; generally, it takes about 5 minutes to complete.

It is worth noting that due to the use of multiple third-party programs by the tool, some of these programs may not fully support directories with special characters, which can affect the unpacking process of the image. Therefore, it is recommended to place the entire firmware tool in a relatively simple path without Chinese characters (e.g., D: \FWFactoryTool) before running the firmware modification tool program.

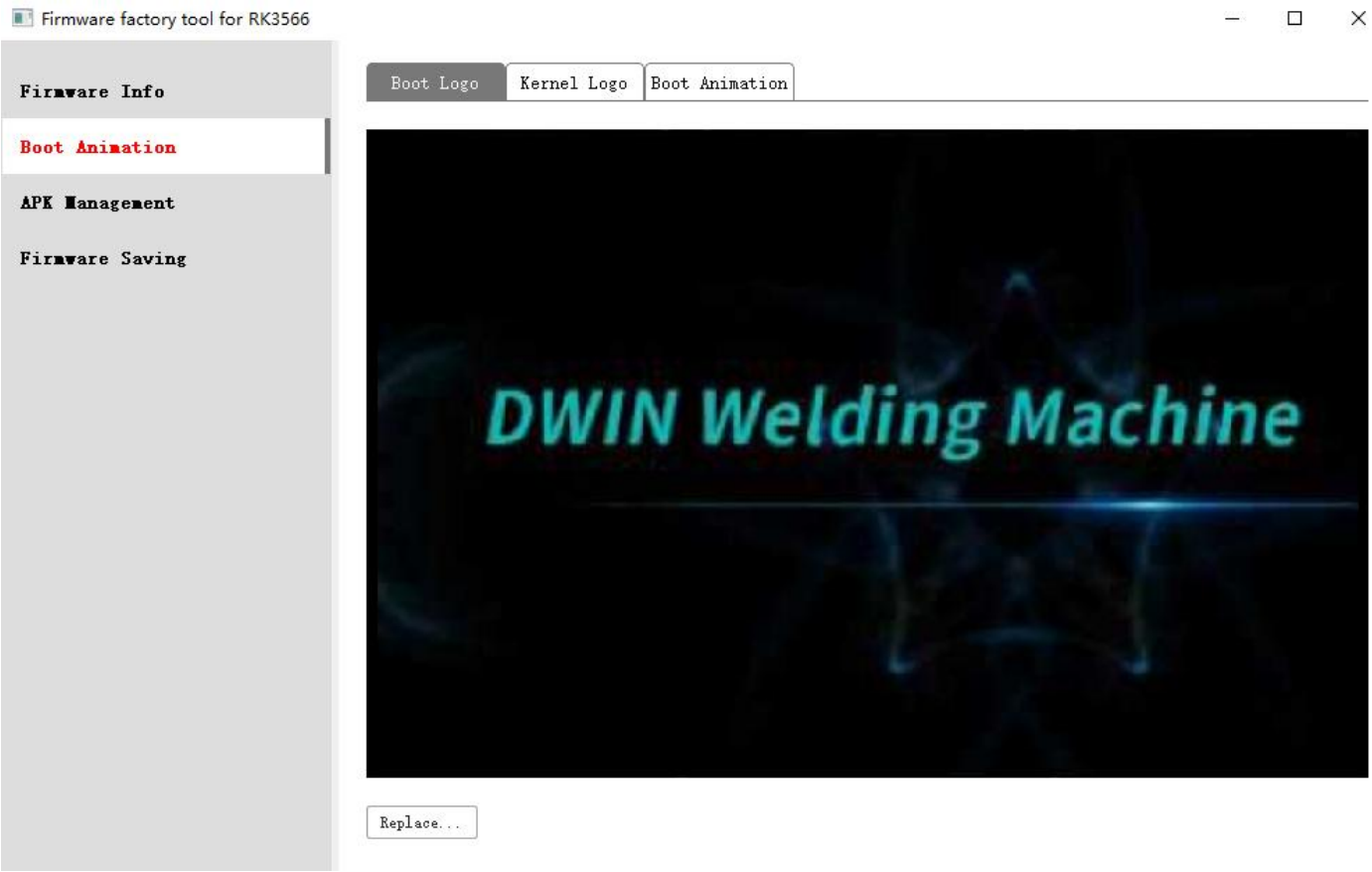
2.3 Procedure for Replacing Boot Logo

Prepare an 8-bit BMP image for replacement in advance. It's recommended to convert it to the required format before use. User can do this by visiting the following link:

https: //online-converting.com/image/convert2bmp/

Click on [Boot Animation] and select either the boot logo or kernel logo. Click [Replace...], then choose an image that meets the requirements (adjust the image as needed when switching from portrait to landscape

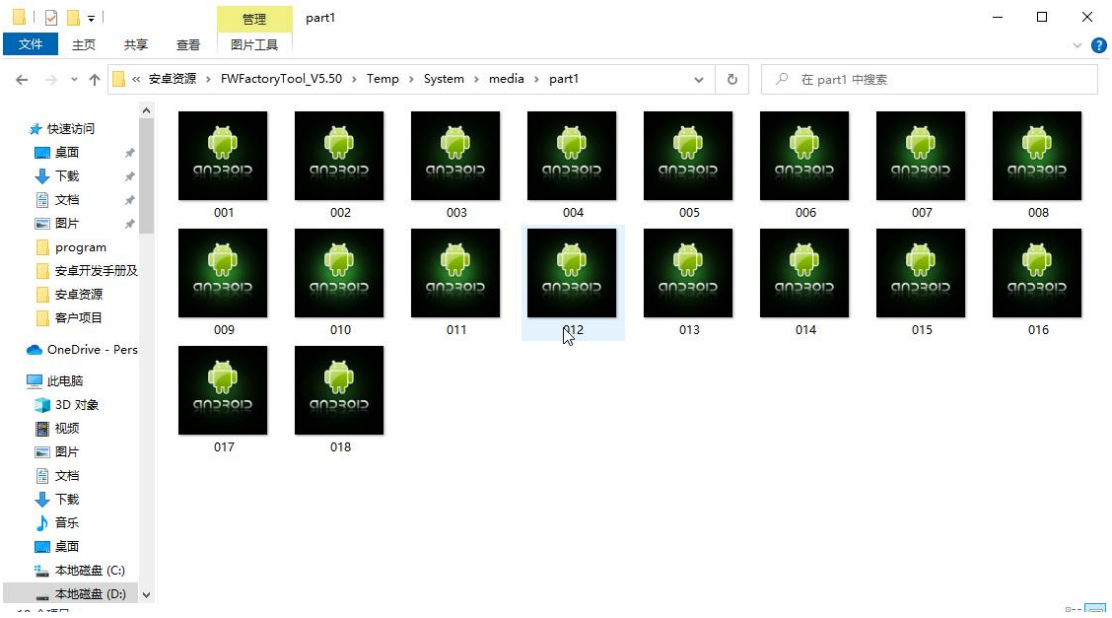
mode). Finally, click Save to complete the replacement process.



2.4 Boot Animation Replacement Process

2.4.1 Creating Boot Animation:

To start, determine the approximate size and quantity of the images based on the device specification and project requirement and create a photo set like below.



Then, place these images into the "part1" folder and create a "desc.txt" file according to user's needs.

```
1080 1920 7
p 0 0 part1
```

In the example image, "1080" and "1920" represent the animation image resolution, and "7" denotes the frames per second for playback.

p 0 (repeat play) 0 (null command) part1 // indicates that the images in the 'part1' folder will be played on a loop. After these two lines of commands, press Enter to create a new line, don't input anything else, and save the file as "desc".

Compress the "part1" folder and "desc.txt" file into a "bootanimation.zip" file to complete the animation creation.

There is no need to put the desc and picture folders into the upper-level folder for compression. The correct example is as follows:

select 0-store when compressing.

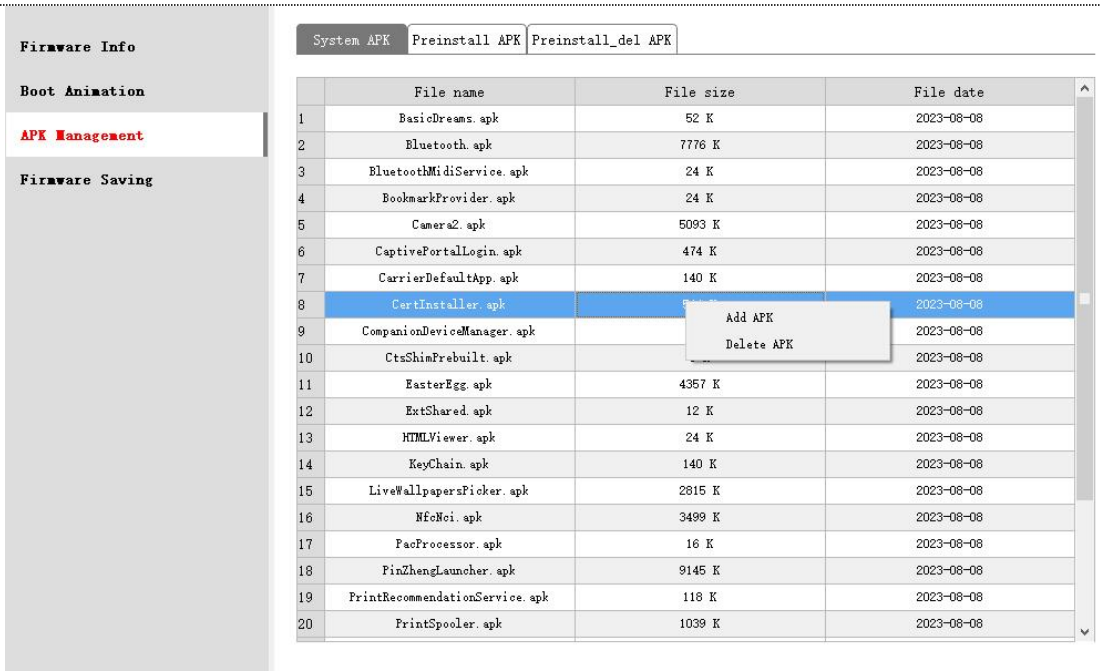
When the startup animation is too large, user can choose to handle the picture collection size.

2.4.2 Boot Animation Replacement:

Click on [Boot Animation], select the boot animation, and then click [Replace...]. Choose the previously created compressed package to replace the animation.

2.5 Pre-installed APK Installation Process

Choose [APK Management], then select either the System APK, pre-installed APK, or preinstall_del APK. Next, select a specific item and right-click to perform the add or delete operation.



2.6 Error Analysis and Resolution

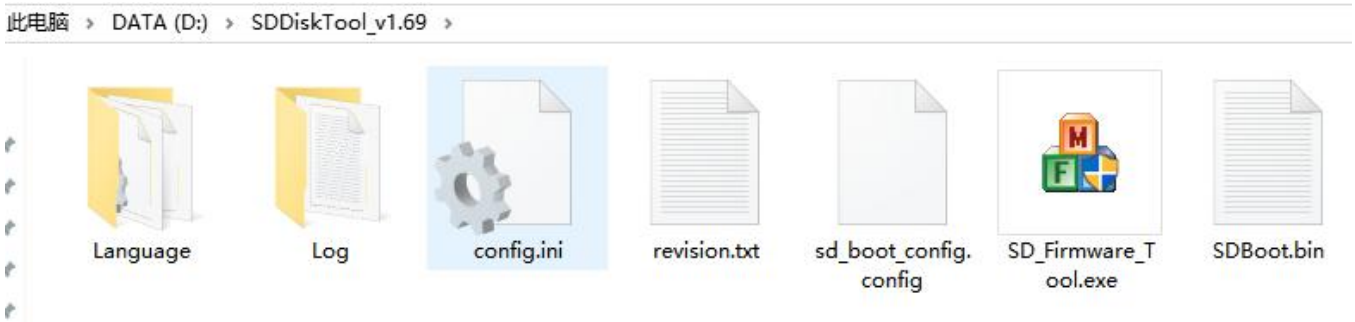
- Unpacking the firmware with this tool will fail if the update.img is currently in use by another application.
- If an error occurs during the flashing process of the generated update.img package file, try updating the flashing software to the latest version and flashing again.

3 Firmware Update and Burning

3.1 SD Card Flashing (SDDiskTool)

3.1.1 Tools/Resource

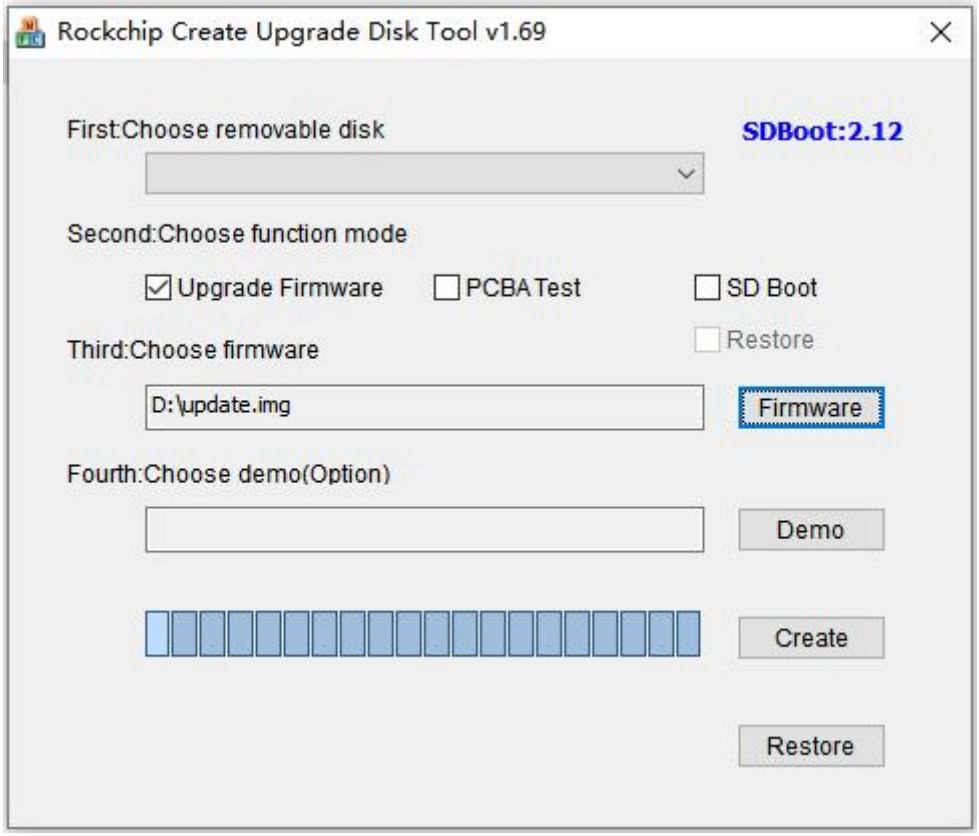
(1) SDDiskTool - SD Card flashing software



(2) Firmware for upgrading (same as used for USB flashing)

3.1.2 SD Card Burning Procedure

Open the folder containing the flashing tool and double-click on "SD_Firmware_Tool.exe."



Insert the SD card into a card reader and connect it to the computer. Follow these steps:

At the first step, choose user SD card's drive letter.

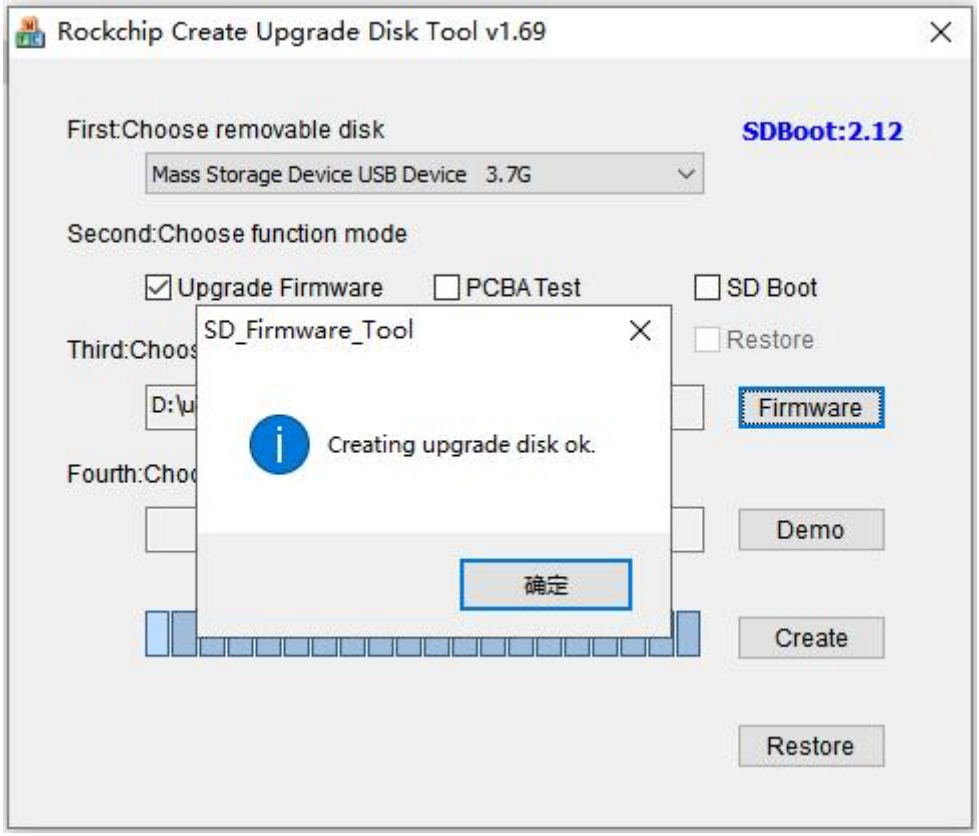
At the second step, select [Upgrade Firmware].

Click [Firmware] on the right side to choose the firmware. Locate the .img file of the firmware user want to burning.

Finally, click on " Create."

Note: Creating the burning card will format the SD card and erase all the existing files. Please make a backup before use.

Upon successful creation, will receive a prompt: " creating upgrade disk ok."



Note: If the process fails, it's recommended to try the above steps again with a different SD card.

3.1.3 Burning Process

With the device powered off, insert the SD card into the device.

Power on the device to initiate the flashing process.

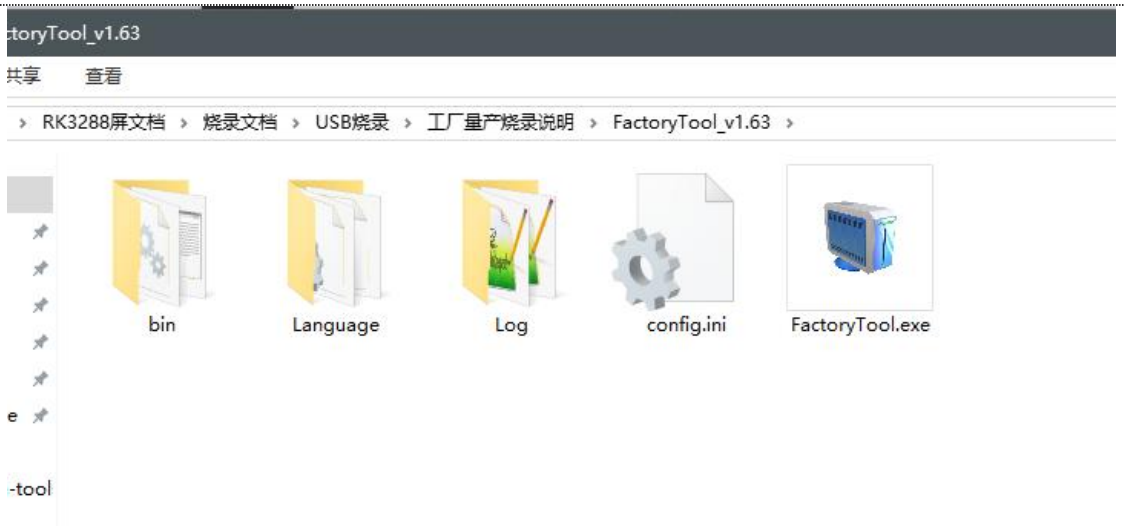
After the flashing process is complete, will receive a prompt to remove the SD card. Simply eject the SD card at this point.

3.2 USB Mass Production Flashing (FactoryTool)

If this is the first time to use the computer for flashing, user might need to install drivers. Please refer to the "USB Driver Installation Instructions" in the USB driver directory.

3.2.1 Tools/Resource

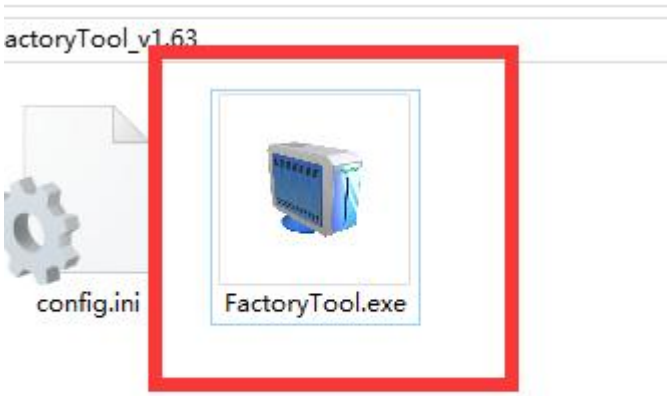
FactoryTool - USB mass production flashing software



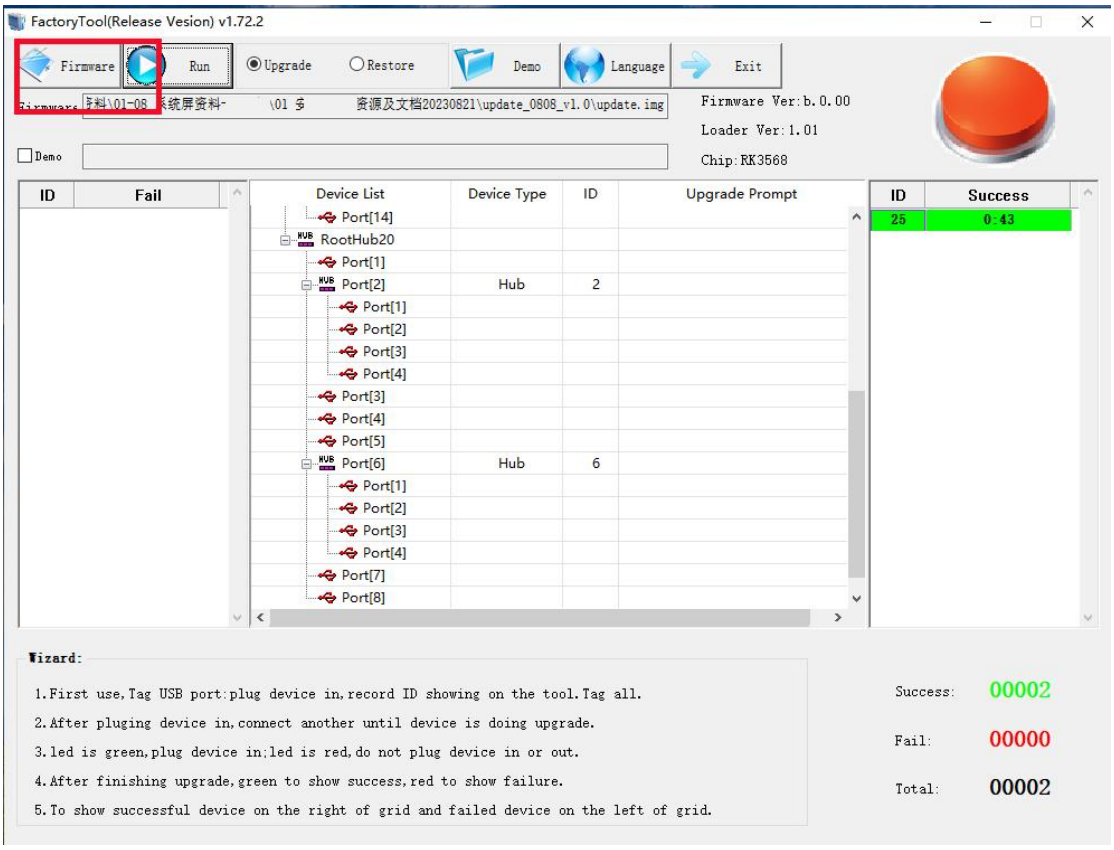
Firmware for upgrading (same as used for SD card flashing).

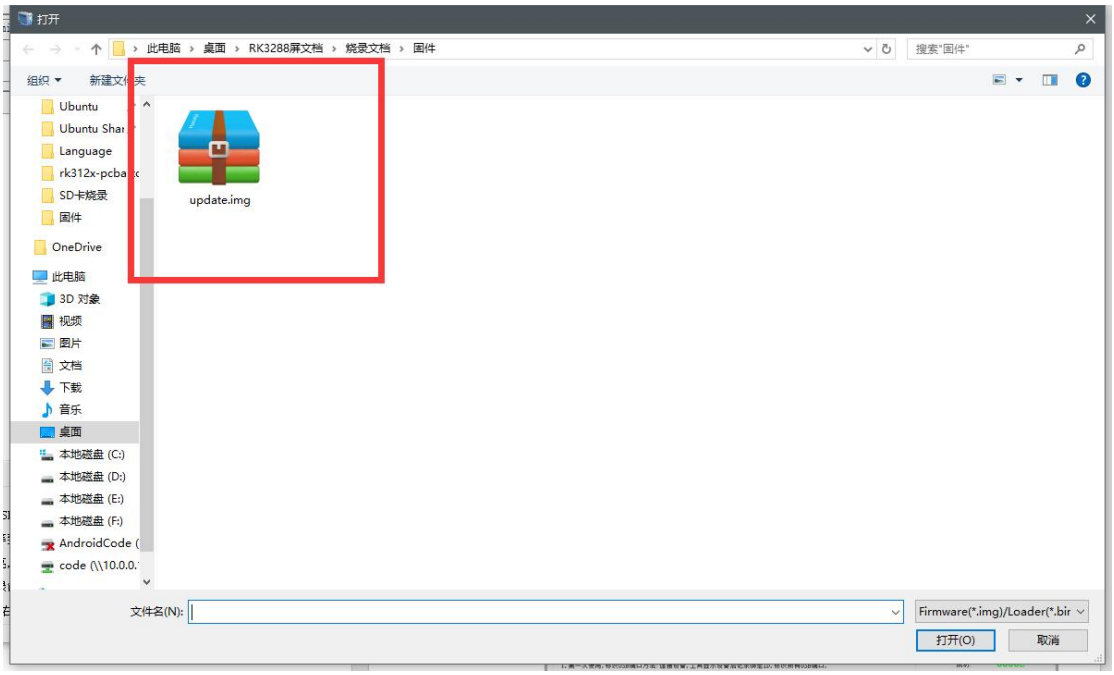
3.2.2 USB Mass Production Burning Procedure

Open the FactoryTool.exe application for USB mass production burning.

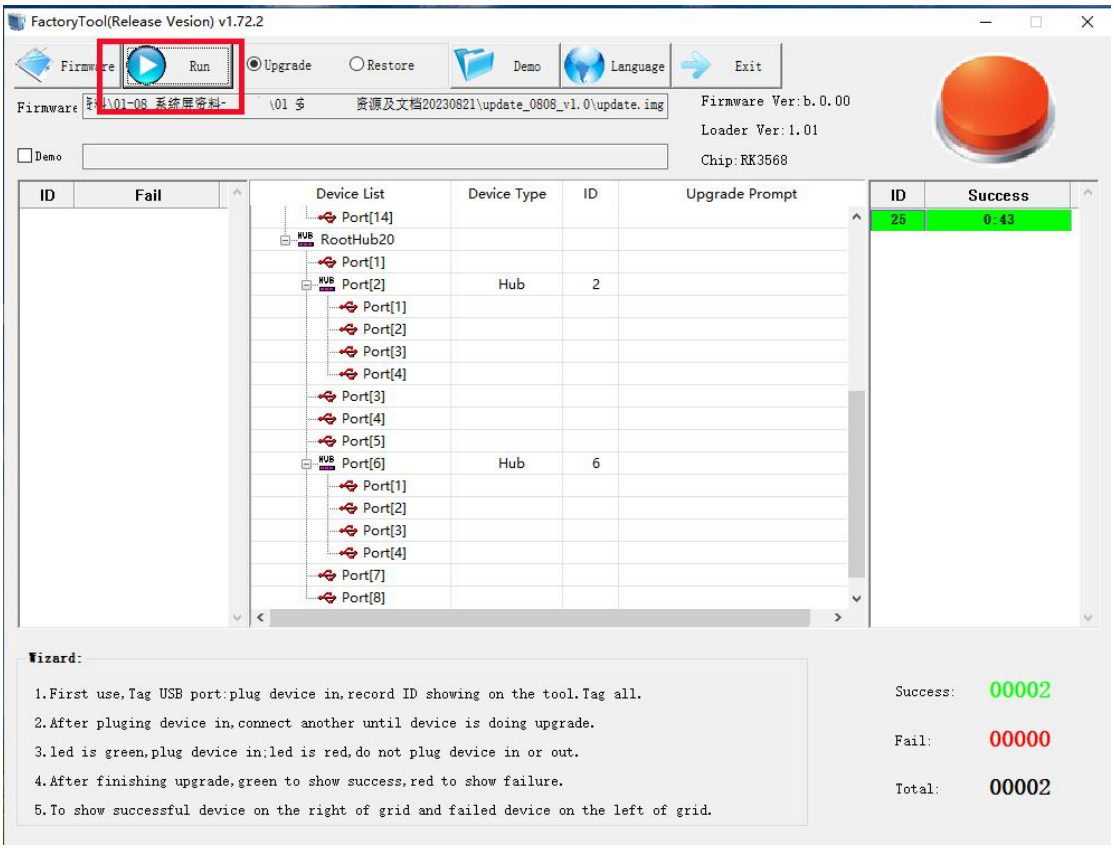


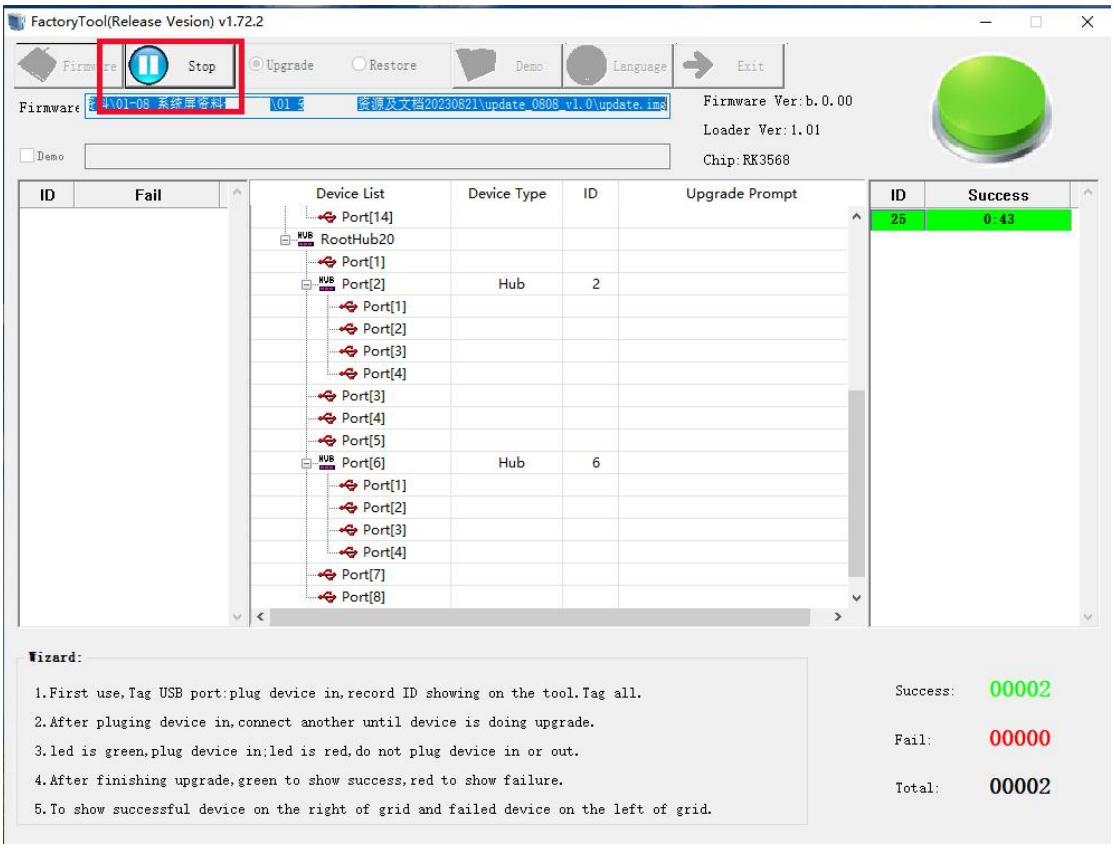
Once the software is opened, select the firmware that needs to be burned.





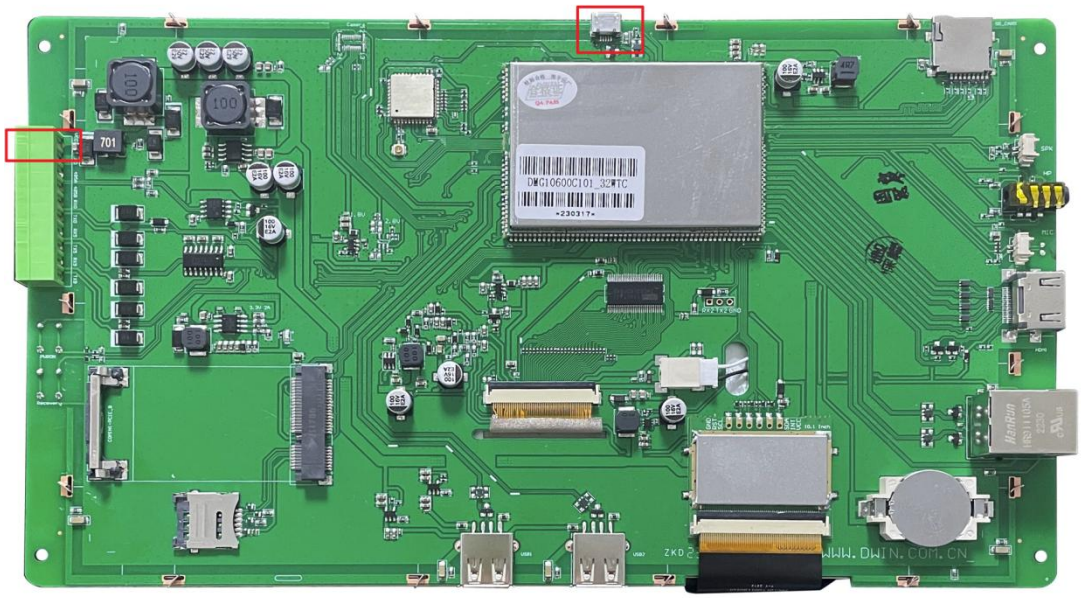
After complete selection, click on "Run" to proceed.



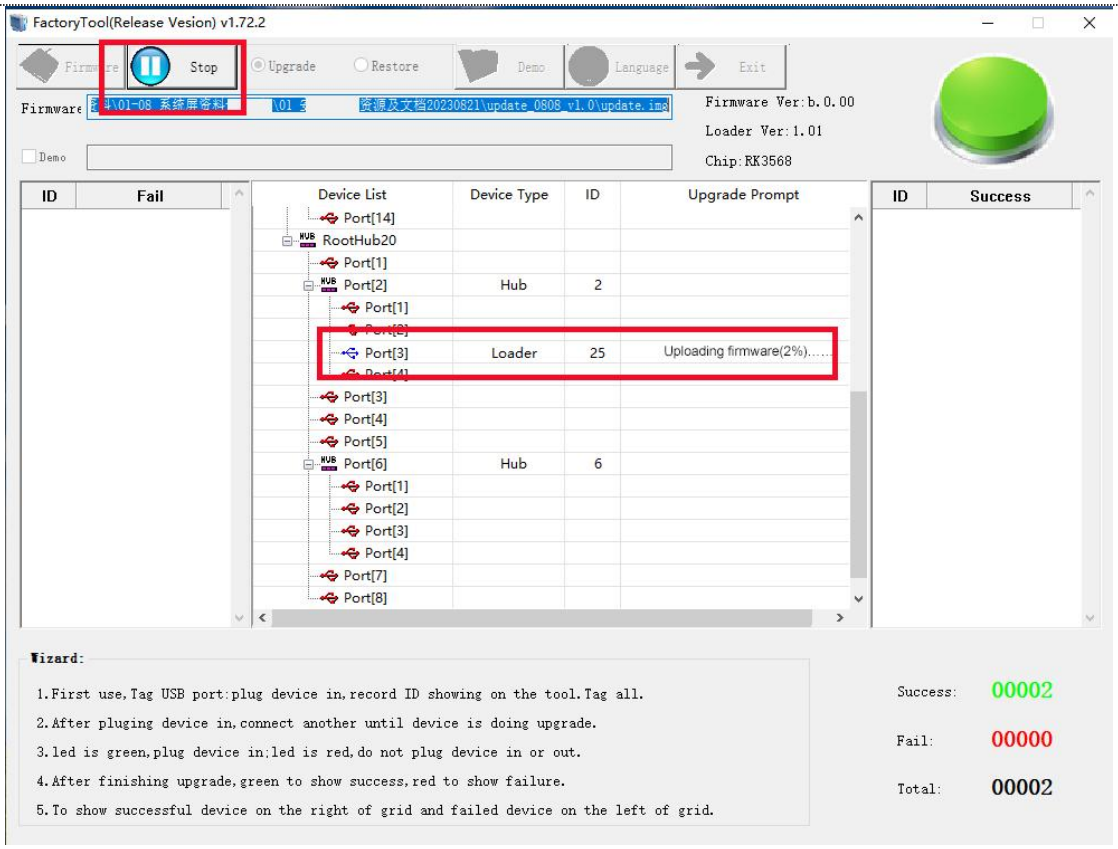


At this point, burning has been initiated.

In a power-off state, press and hold the MASROM/RECOVERY key on the Android screen, connect the device to the PC using a USB cable and then access the power supply.



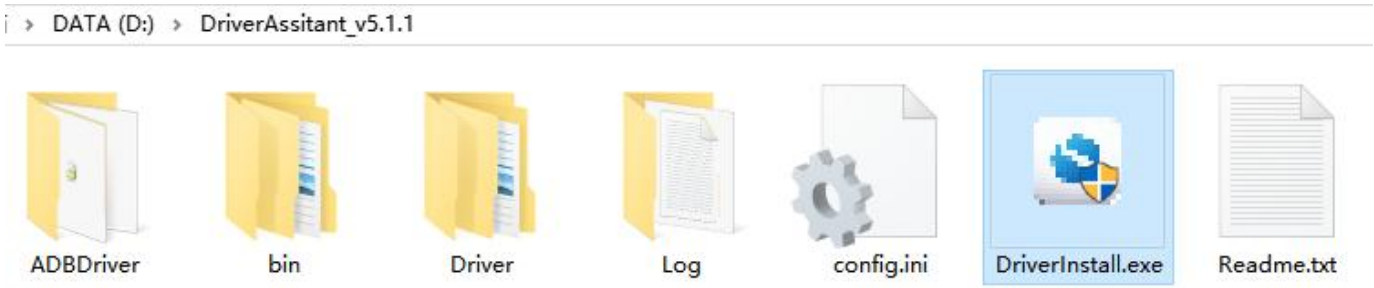
When connect the device to the computer as outlined in step 3, the burning process will start automatically. The software will display a list of devices.



Each port corresponds to a USB port on the computer. Once the burning process is completed, the device will power on automatically. User can check the bottom right corner for a statistical summary of successful or failed burning.

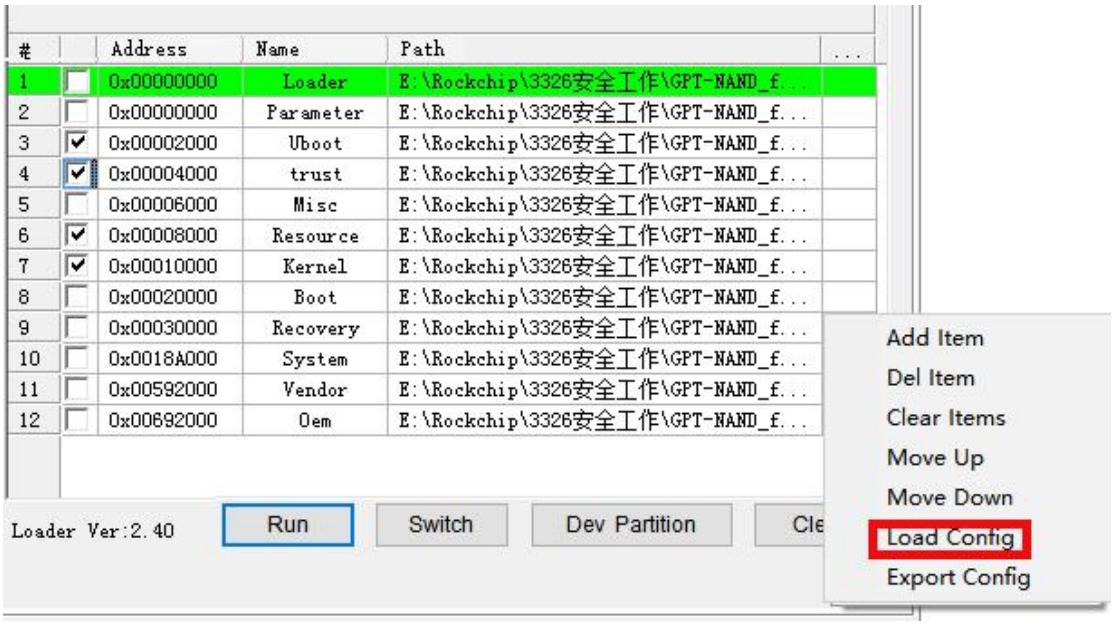
3.3 USB Developer Flashing Tool (RKDevTool)

Note: If the computer being used is burning for the first time, drivers need to be installed first. Please refer to the "USB Driver Installation Instructions" in the USB driver directory.

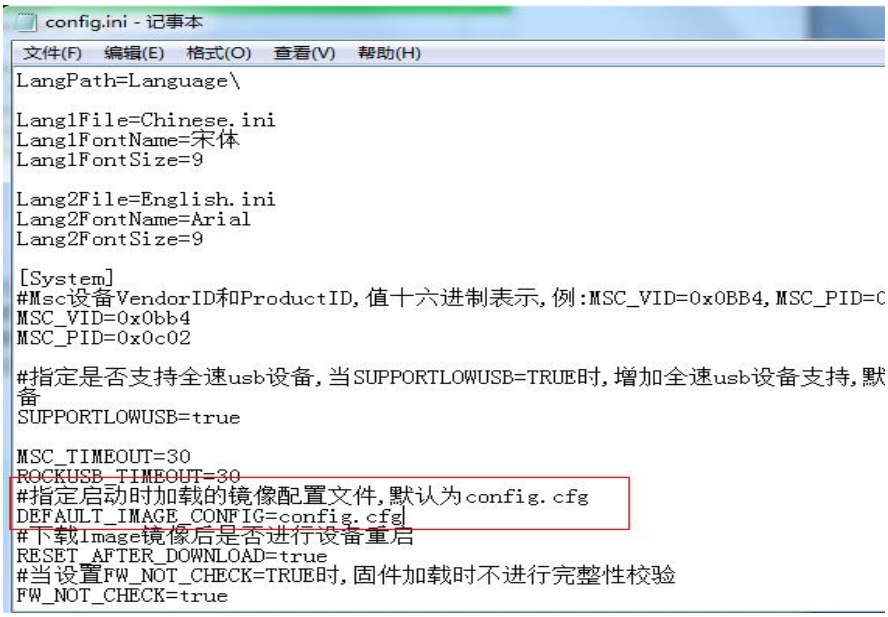


Key Features:

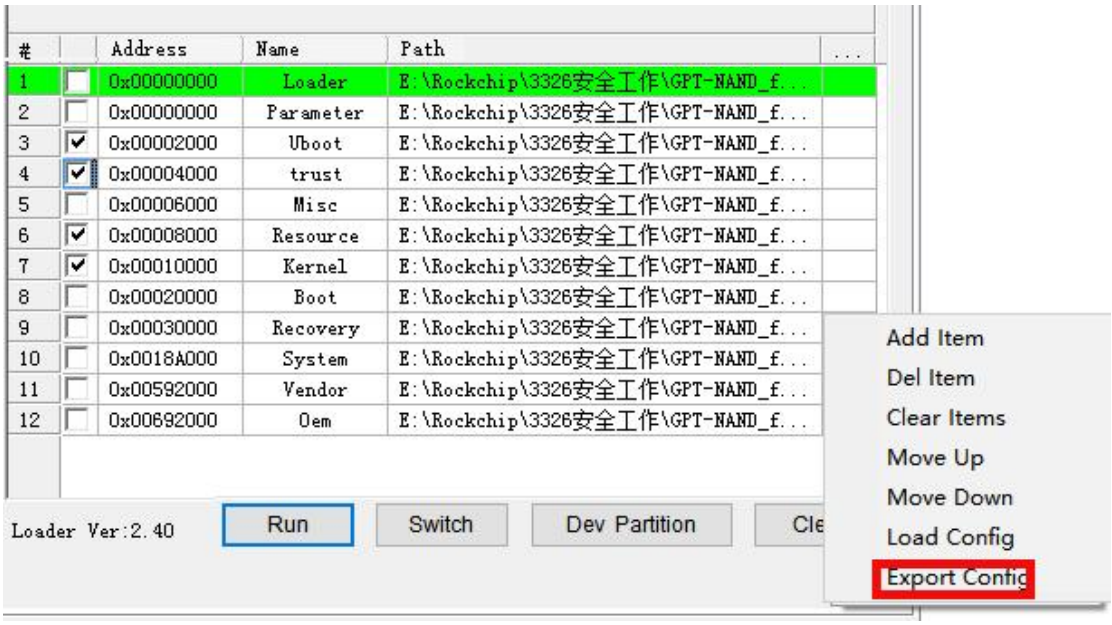
3.3.1 Import Partition Configuration



Often, the partition configuration provided with the new tool differs slightly from the project's configuration. User can right-click in the partition configuration area and choose "Import Configuration" to load a previously saved project partition configuration file. To load a specific configuration file upon tool startup, modify the DEFAULT_IMAGE_CONFIG entry in the Config.ini file.

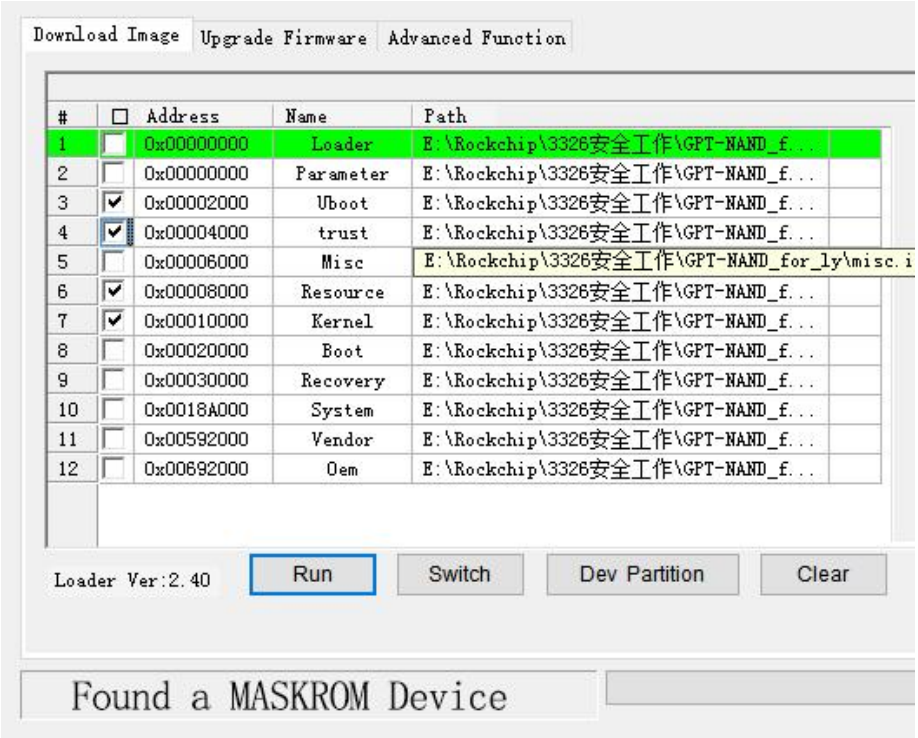


3.3.2 Export Partition Configuration



After configuring partitions, user can save their partition setup by exporting it. Click "Export Configuration" in the blank area of the partition configuration zone. Specify a file name and directory for saving. If replace the config.cfg file in the tool directory, the saved configuration will load when the tool starts.

3.3.3 Burning One or Multiple Partition Images



Steps:

- Connect the device to the PC in Loader or Maskrom mode. In Maskrom mode, select the "Loader" download option.

- Check the desired flashing items. Confirm that the addresses before the flashing items are correct. If an address is zero, load partition information from the "Parameter" partition table.

- Click "Execute."

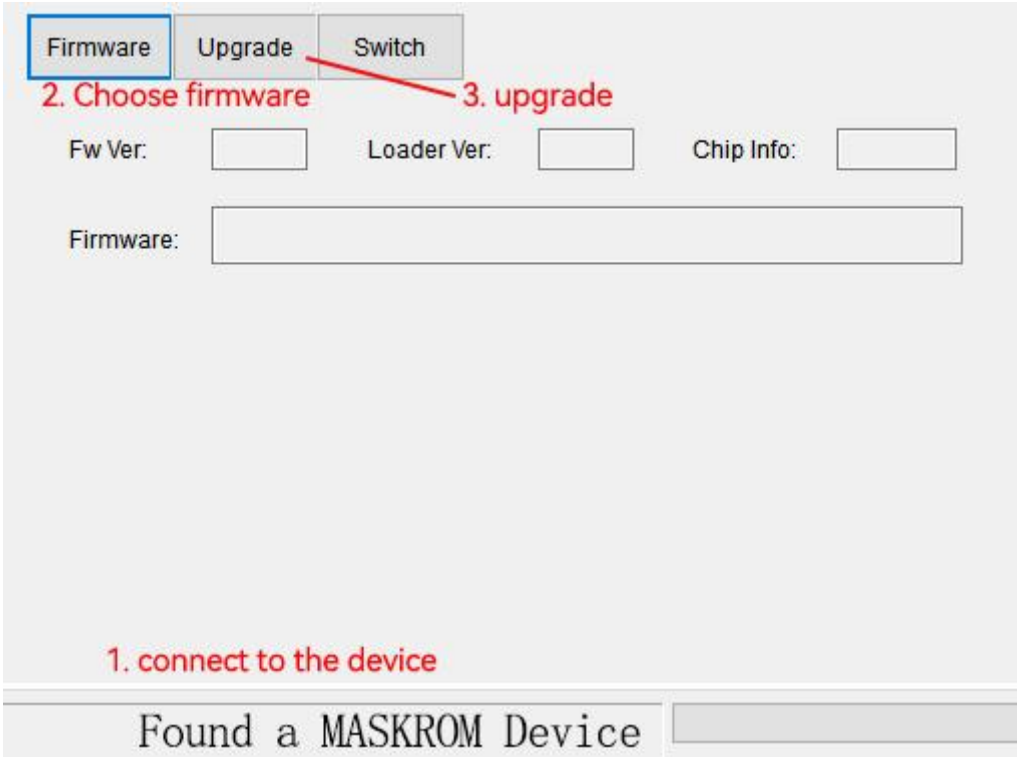
3.3.4 Switching

When Found a MASKROM Device appears, execute "Switch" to enter Loader or Maskrom mode.

3.3.5 Device Partition Table

To use the device's current partition table for flashing, click " Dev Partition". At this point, the partition table is read from the device side. Partition information will be parsed and loaded.

3.3.6 Burning update.img

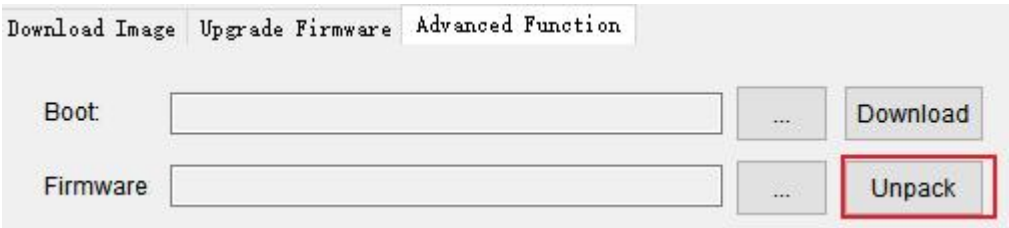


Note: Flashing update.img is only possible in loader and maskrom modes; other modes require switching first. The firmware can be an update.img firmware or a loader file.

3.3.7 Erase Flash

Executing the "Erase Flash" function erases all blocks of the Flash, including the system blocks before the firmware area. If repeated firmware upgrades fail, try performing "Erase Flash" before upgrading again.

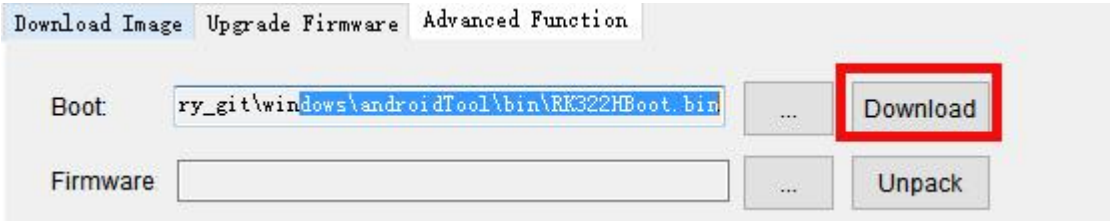
3.3.8 Unpack update.img



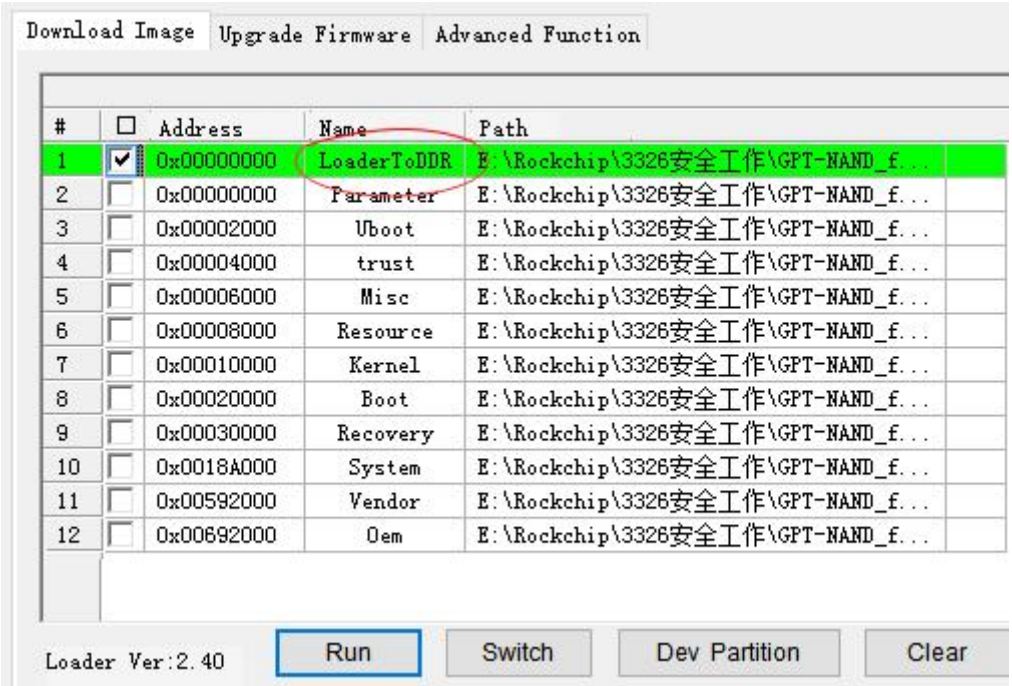
Unpacked files will be saved in the "output" directory under the tool's location.

3.3.9 Download Boot

Downloading the boot initiates DDR initialization and loads the usbplug code into DDR through the loader. User can perform this by the following two ways.



Double-clicking "Loader" and selecting "LoaderToDDR" from the dropdown list.



3.3.10 Download GPT

A GPT partition can be in the format of a parameter file or a partition_table.img file. If it's a parameter file, note:

The parameter file should have the attribute "TYPE: GPT."

```
FIRMWARE_VER:7.1
MACHINE_MODEL:RK3126c
MACHINE_ID:007
MANUFACTURER:rk3126c
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 3126c
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
CMDLINE:mtdparts=rk29xxnand:0x00002000@0x00004000(uboot),0x00002000@0x00006
```

```
6000(atf),0x00038000@0x00008000(boot:bootable),-@0x0040000(rootfs:grow)
```

The last partition should use the "grow" attribute to allocate any remaining space to that partition.

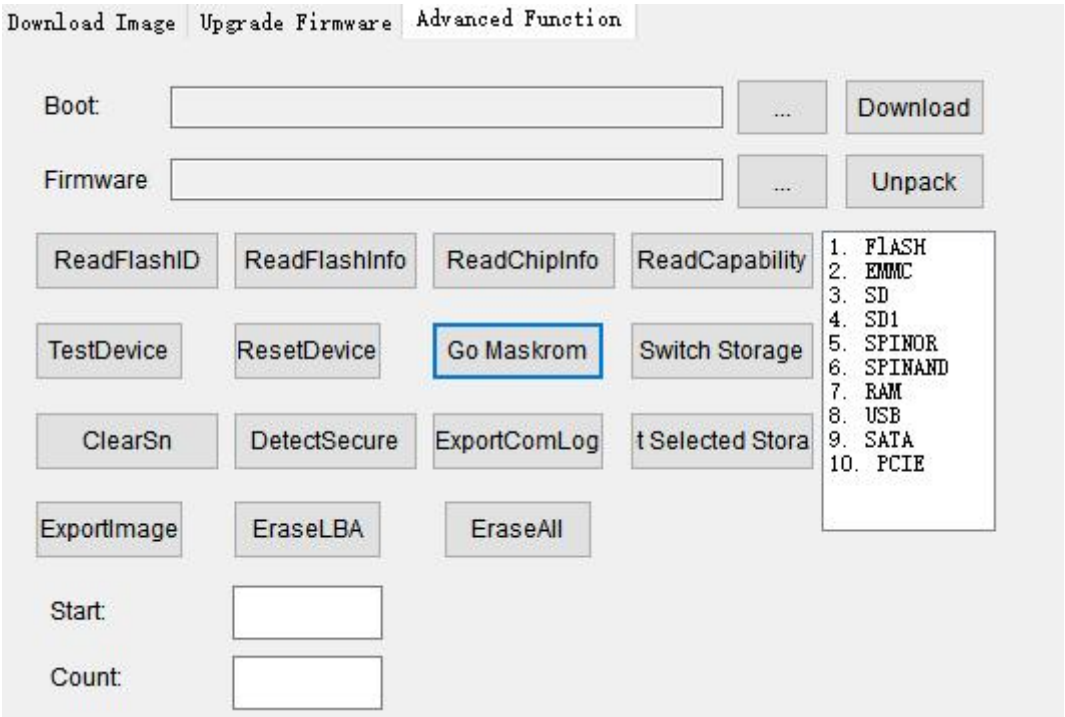
#	☐	Address	Name	Path
1	☑	0x00000000	LoaderToDDR	E:\Rockchip\3326安全工作\GPT-NAND_f...
2	☐	0x00000000	Parameter	E:\Rockchip\3326安全工作\GPT-NAND_for_ly\gpt.txt
3	☐	0x00004000	Uboot	E:\Rockchip\3326安全工作\GPT-NAND_f...
4	☐	0x00006000	trust	E:\Rockchip\3326安全工作\GPT-NAND_f...
5	☐	0x00008000	Misc	E:\Rockchip\3326安全工作\GPT-NAND_f...
6	☐	0x0000A000	Resource	E:\Rockchip\3326安全工作\GPT-NAND_f...
7	☐	0x00012000	Kernel	E:\Rockchip\3326安全工作\GPT-NAND_f...
8	☐	0x00022000	Boot	E:\Rockchip\3326安全工作\GPT-NAND_f...
9	☐	0x00032000	Recovery	E:\Rockchip\3326安全工作\GPT-NAND_f...
10	☐	0x0018C000	System	E:\Rockchip\3326安全工作\GPT-NAND_f...
11	☐	0x00594000	Vendor	E:\Rockchip\3326安全工作\GPT-NAND_f...
12	☐	0x00614000	Oem	E:\Rockchip\3326安全工作\GPT-NAND_f...

Loader: 2.40 Run Switch Dev Partition Clear

3.3.11 Read Device Extended Functions

For GPT burning, the "DirectLBA" in the extended functions must be enabled; otherwise, the tool will prompt that the "current device doesn't support GPT."

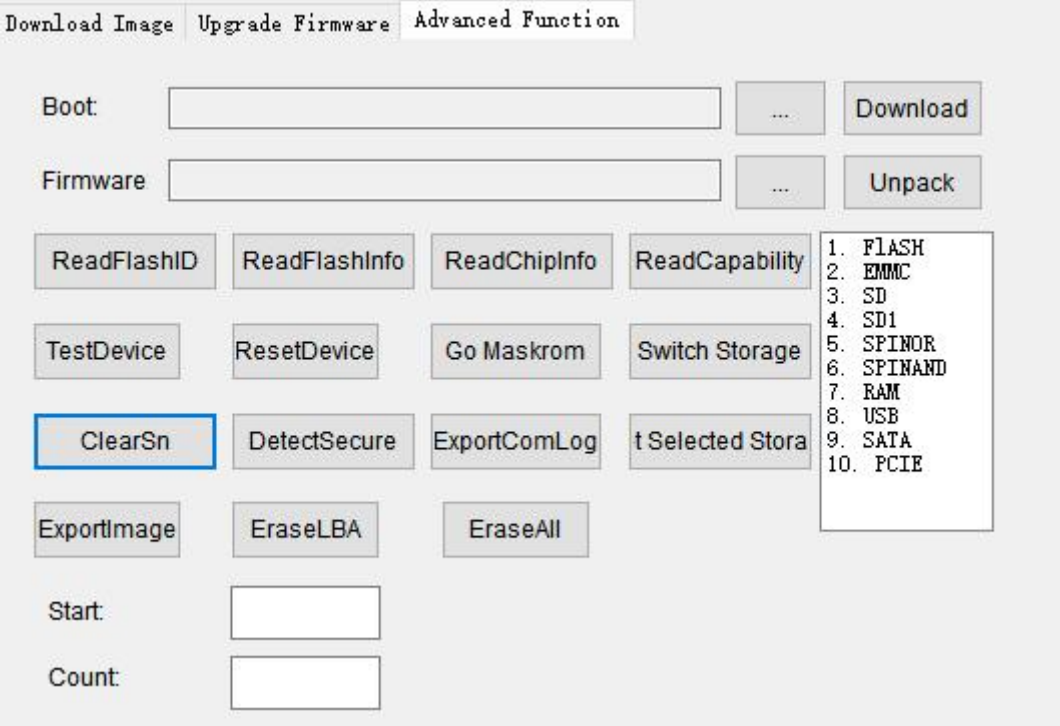
3.3.12 Enter Maskrom



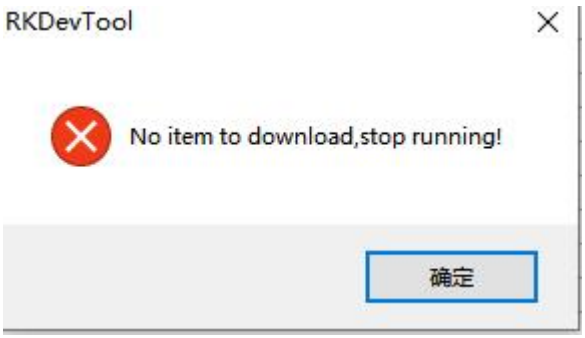
When the device is in Loader mode, can use this function to switch the device to Maskrom mode.

3.3.13 Clear Serial Number

If a serial number was written using a unified dynamic library tool, it can be cleared using this function.



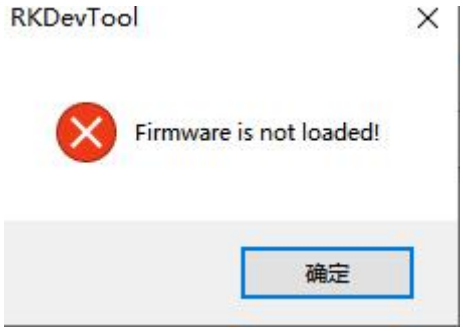
3.3.14 Common Issues:



(1) Missing Download Items

As shown in the image above, it indicates that the download item for the kernel partition doesn't exist.

(2) Failed to Load Firmware



Check the following issues:

- Whether the firmware is being used by other programs.

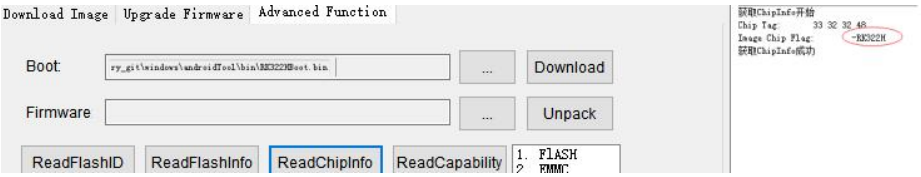
- Incorrect firmware format.

- Corrupted firmware.

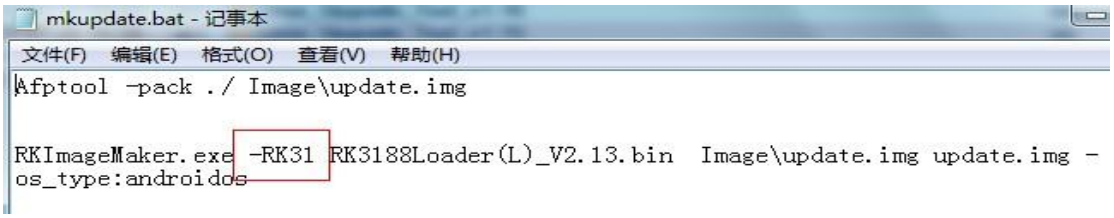
(3) ChIP Verification Failed

This indicates that the loader or firmware doesn't match the device.

Use the "Read ChIP Information" function to confirm the chIP identifier.



Make sure that the chIP identifier is set the same as what's displayed in "Image ChIP Flag" when packaging firmware or loader.



(4) Failed to Download Boot

Check if DDR is properly soldered.

Ensure that the loader matches the device.

(5) Failed to Download Firmware or Partition Image

Update Rockusb driver or the burning tool.

Use a shielded USB cable and connect to rear USB ports.

Check if the flash is loose, damaged, or unsupported.

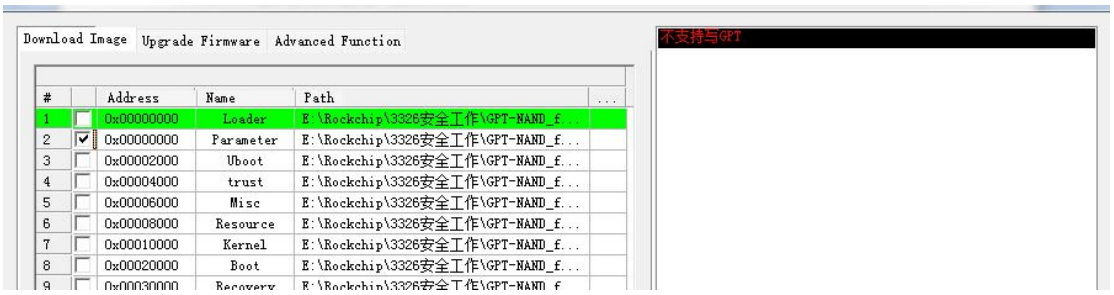
(6) Failed to Verify Firmware or Partition Image

Confirm that the sizes of partitions in the parameter file can accommodate the corresponding image files.

If there are issues with Flash software mapping, try erasing the flash first.

Check the stability of the DDR device.

(7) GPT Writing Not Supported



Use the "Read Capability" function to confirm if the current device supports DirectLBA functionality.

Writing GPT requires DirectLBA functionality to be enabled.

Important Notes:

- When using the tool in a non-Chinese operating system, ensure that the tool's directory path contains only English characters.

- On Windows 7 and Vista systems, right-click and run the program with administrator privileges.

- After modifying the Config.ini configuration file, need to restart the tool for the changes to take effect.

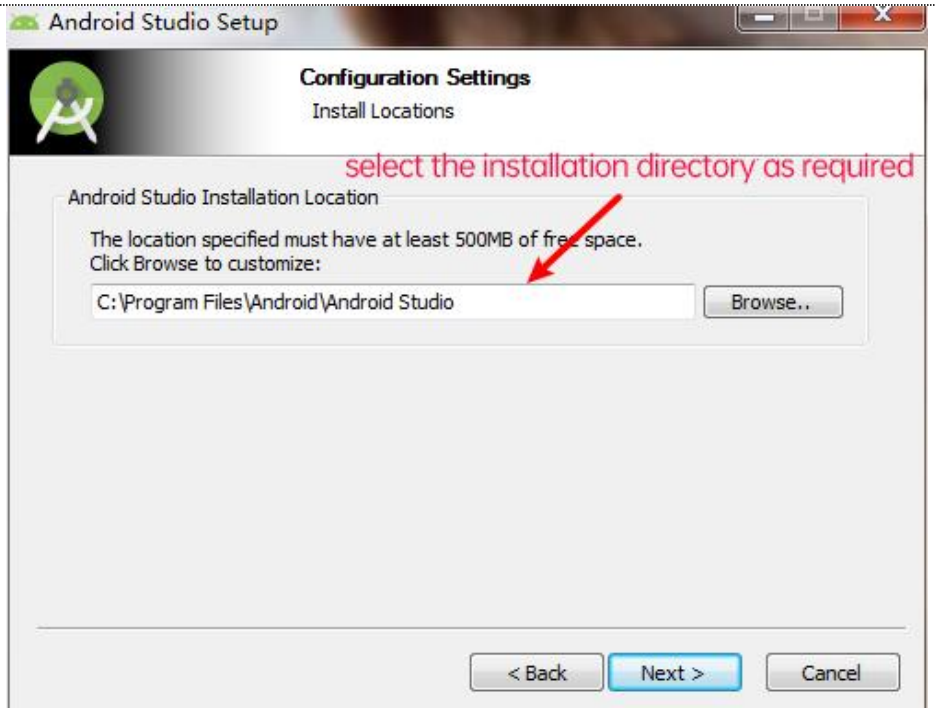
4 Setting Up and Using the Android Studio Development Environment

4.1 Tools/Resource

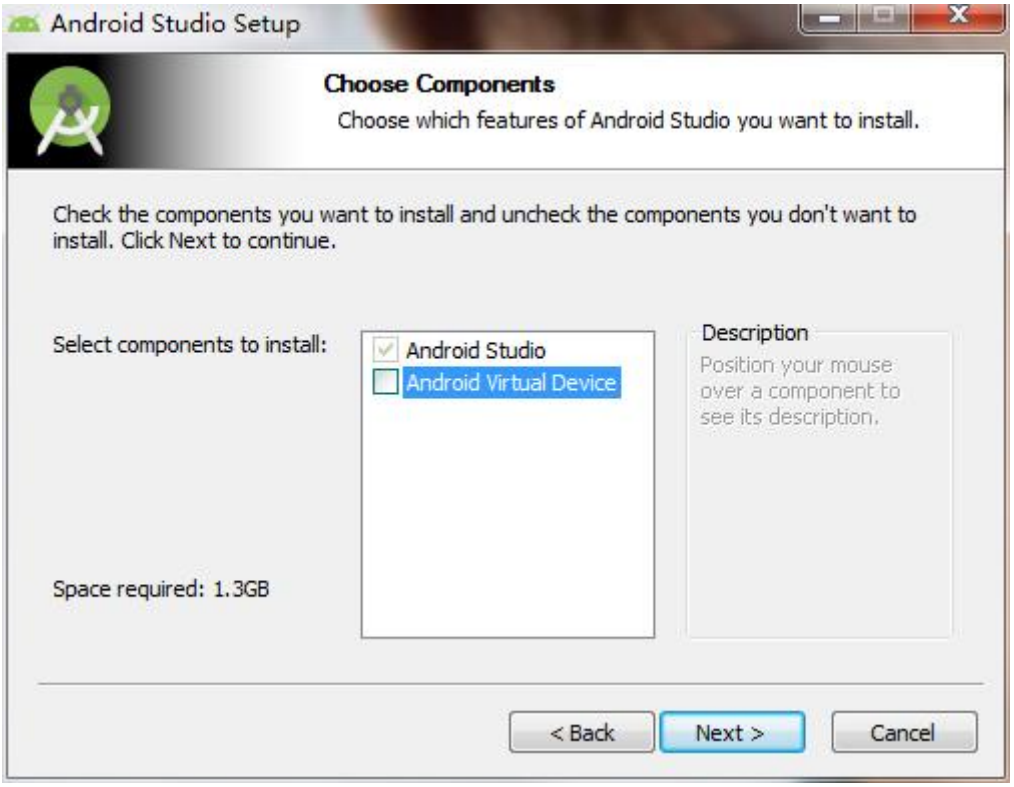
The installation programs can be downloaded from the following link: [https: //developer.android.google.cn/studio/archive] (https: //developer.android.google.cn/studio/archive) (Please download Android Studio 3.6 or later versions)

4.2 Software Installation

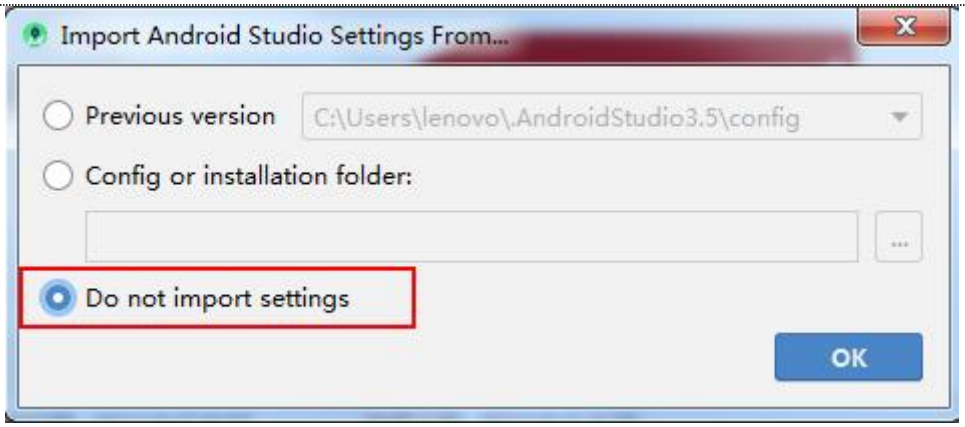
(1) Double-click the installation package to start the installation.



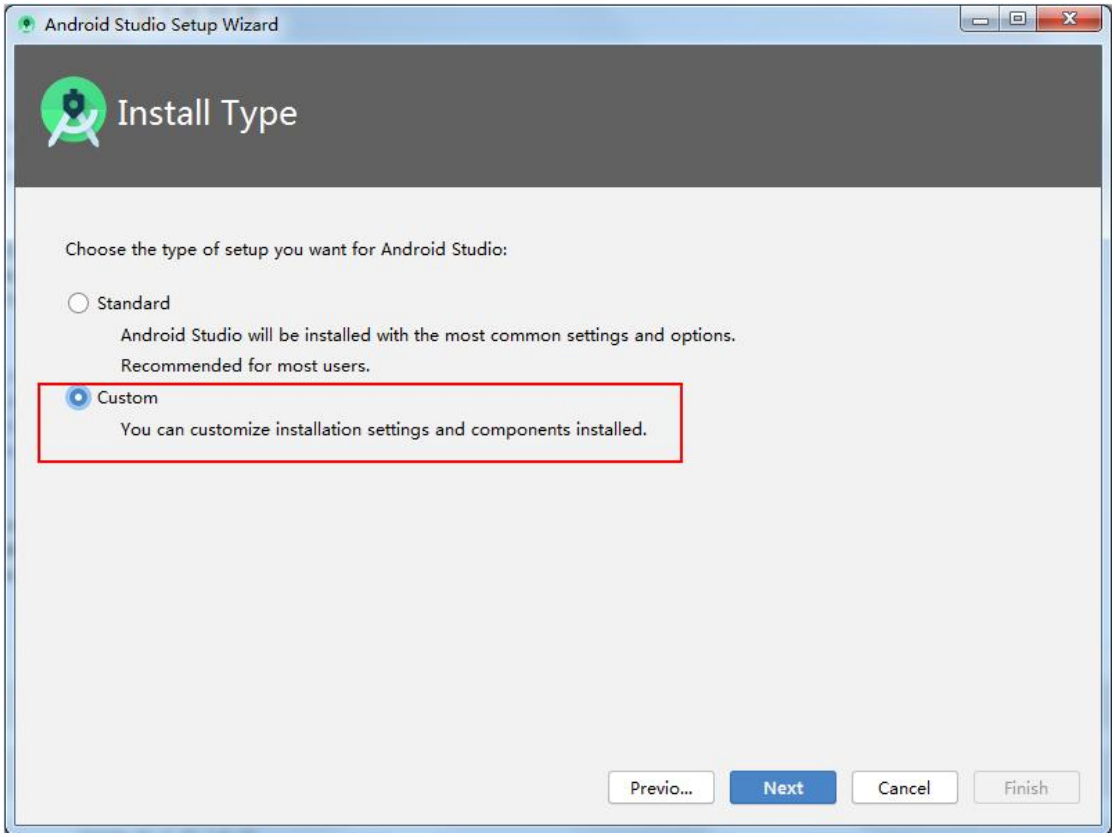
Choose whether to install AVD (Android Virtual Device) based on user's needs.



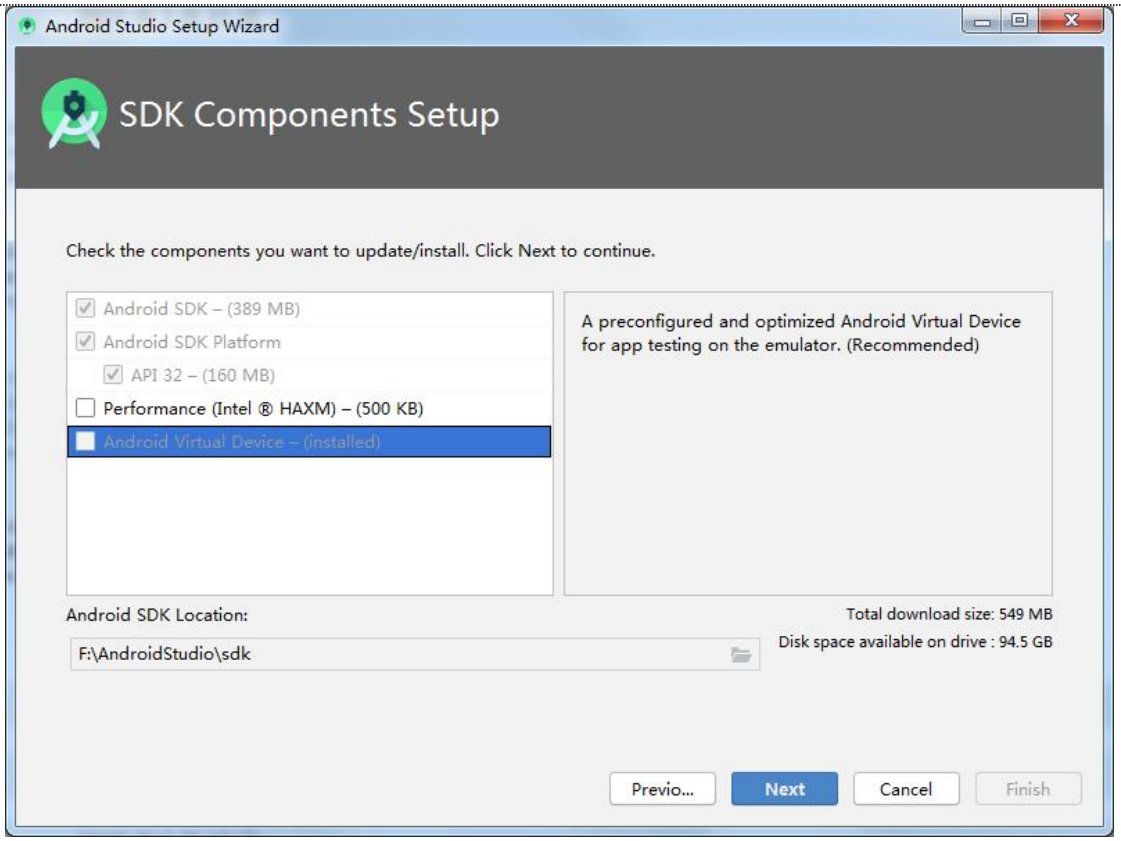
For the first-time use of Android Studio, it's recommended not to import settings.



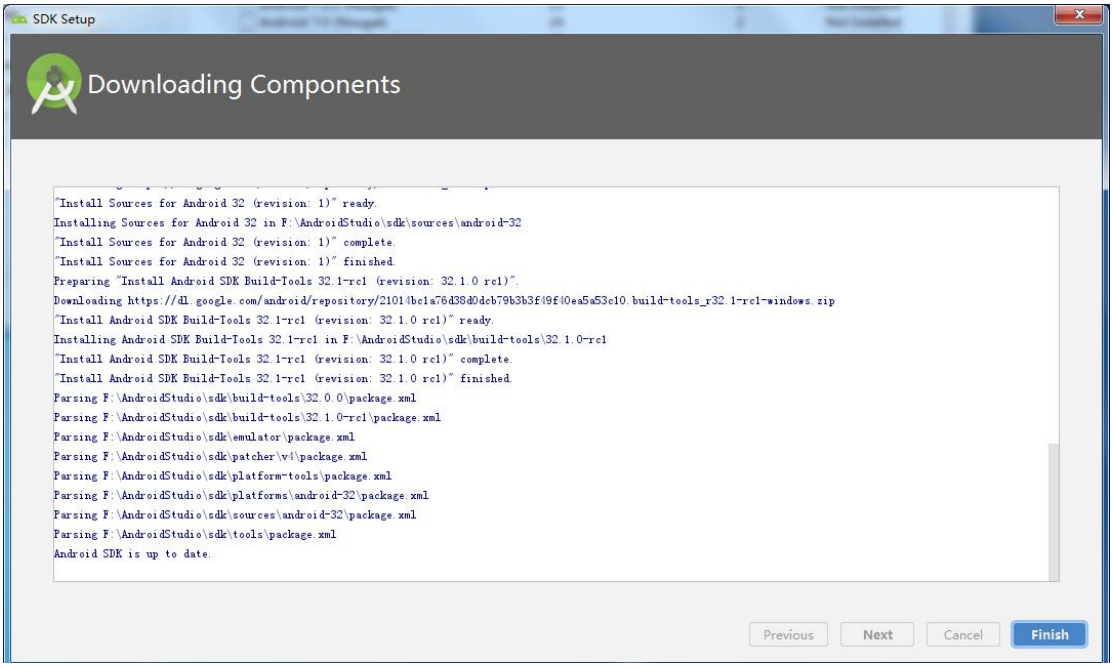
Select the installation type. Choose custom installation.



Select the location for SDK installation.



Wait for the installation to complete.

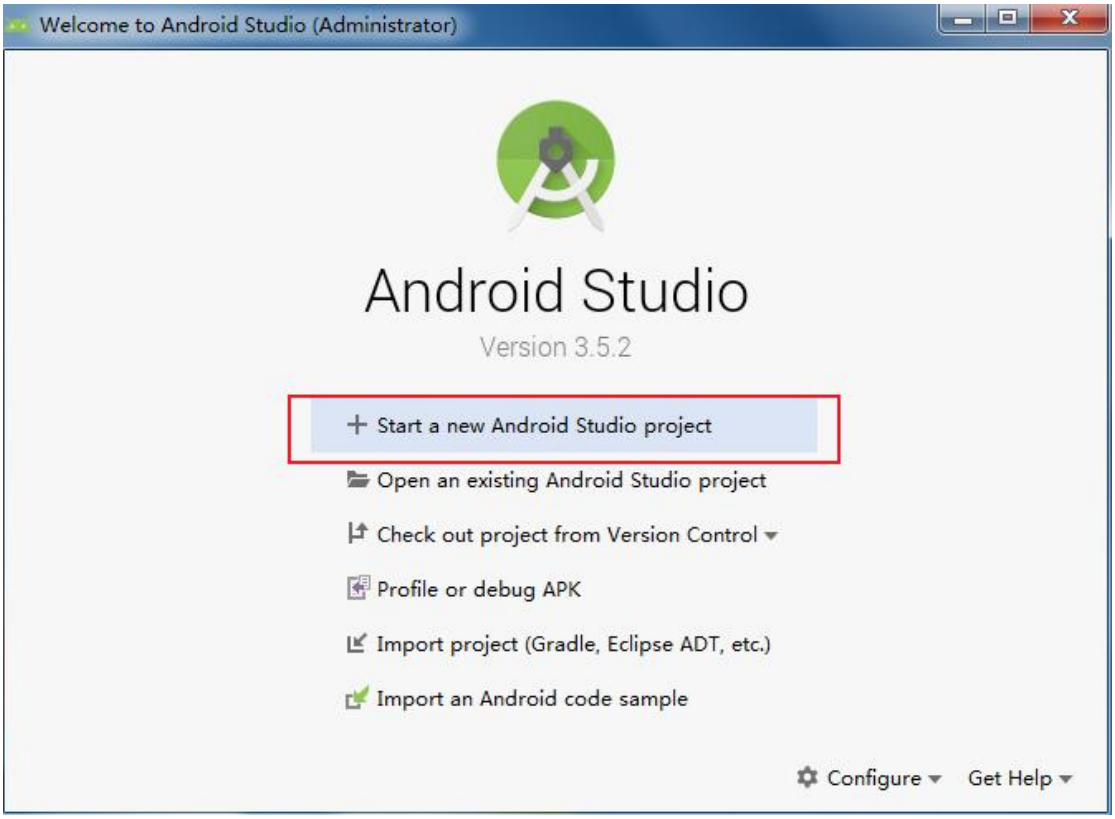


(2) Open the installed Android Studio and create a new project.

The first time to open it, might encounter an error as shown below. Choose [Cancel] to close it.

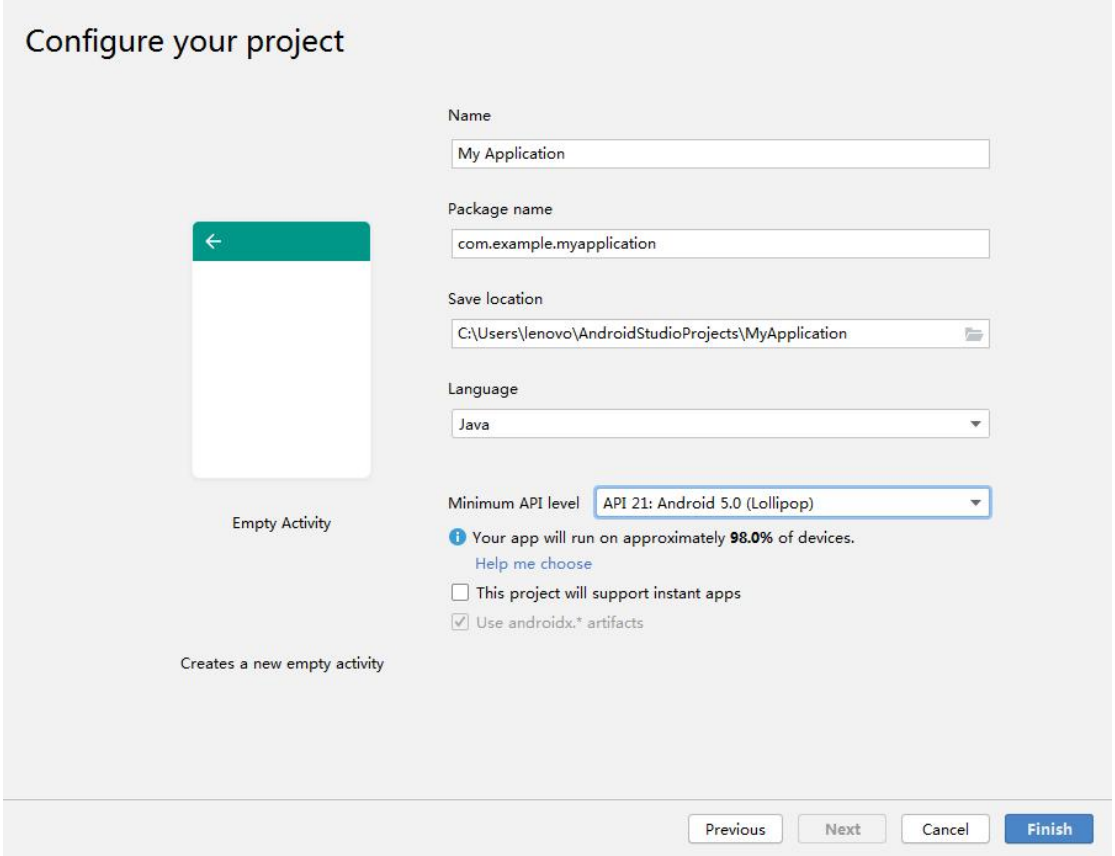
For the first use, Android Studio will perform downloads and automatic environment configuration. Just wait for the process to finish.

4.3 Creating a New Project



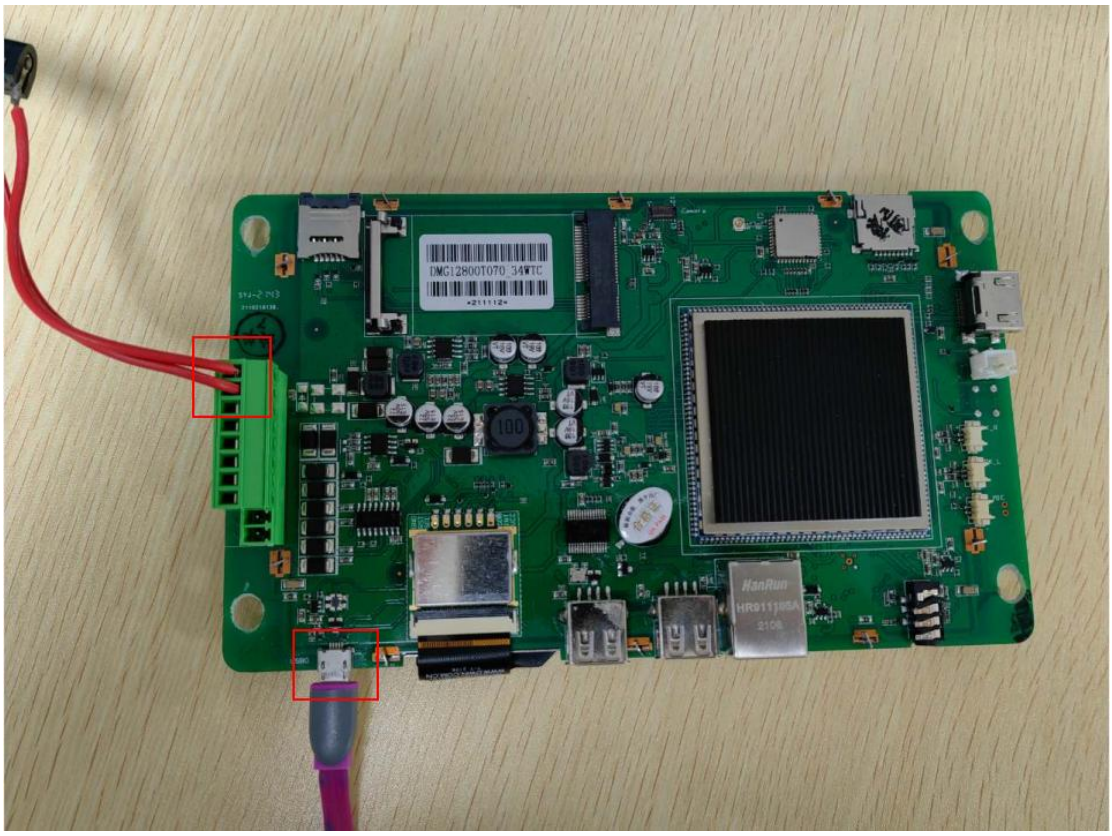
(1) Choose "Start a new Android Studio project."

(2) Fill in the project name, package name, set the storage location, and select the programming language. In this case, we'll use Java and set the minimum compatible Android version to 5.0.



(3) Build and Run the First Project

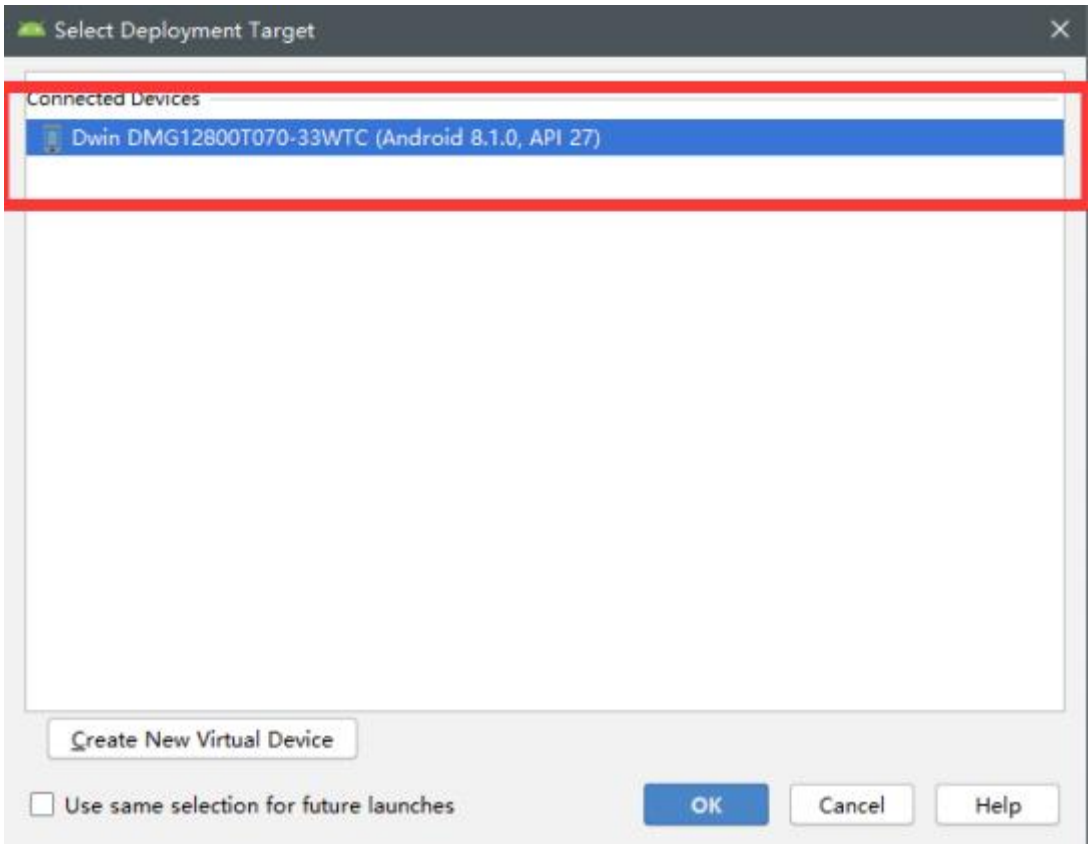
Make sure to connect the Android device to the PC before selecting debug and run.

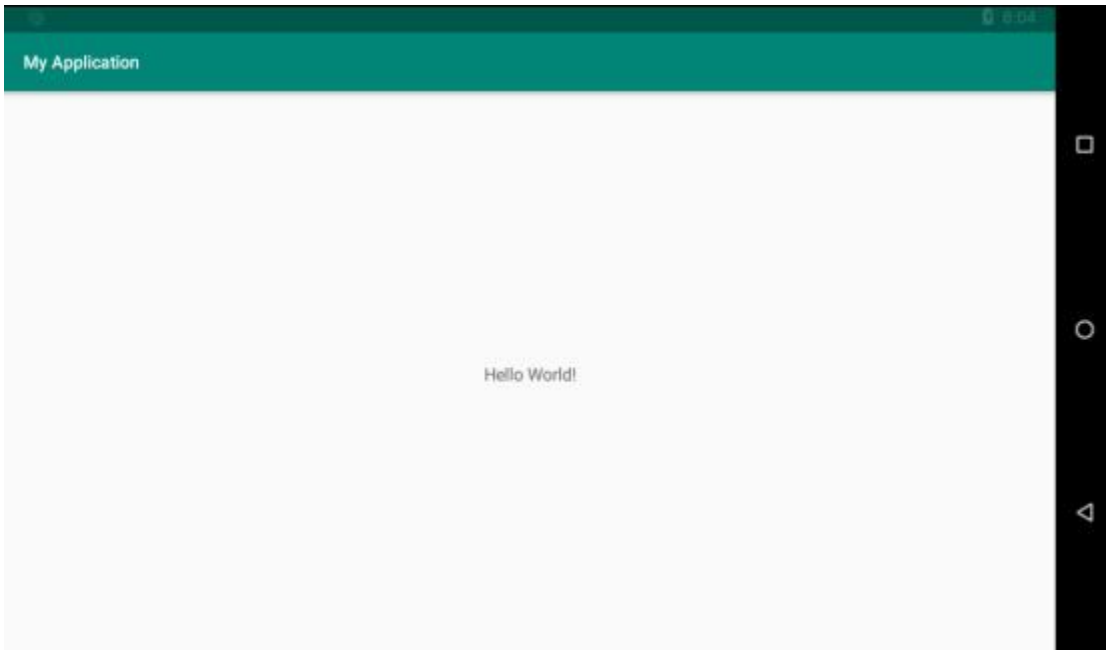


As shown, connect the Android device to the computer via USB and power on the device. (The layout of

each Android screen may vary, the following image is for reference only.)

Click on the DWIN screen in the designated area, then click "Run." The example program will run as shown below.

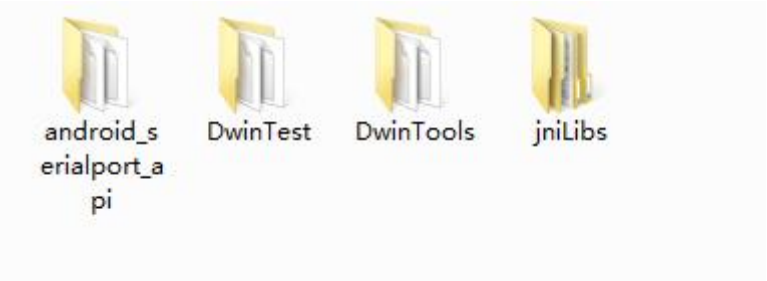




5 Using DWIN Android Screen Serial Port Tools

5.1 DWIN Android Tools Package Instruction

DWIN provides an Android testing software source code and a Java library file for serial communication. The provided files include:



- jniLibs: Pre-packaged shared object libraries (so files)

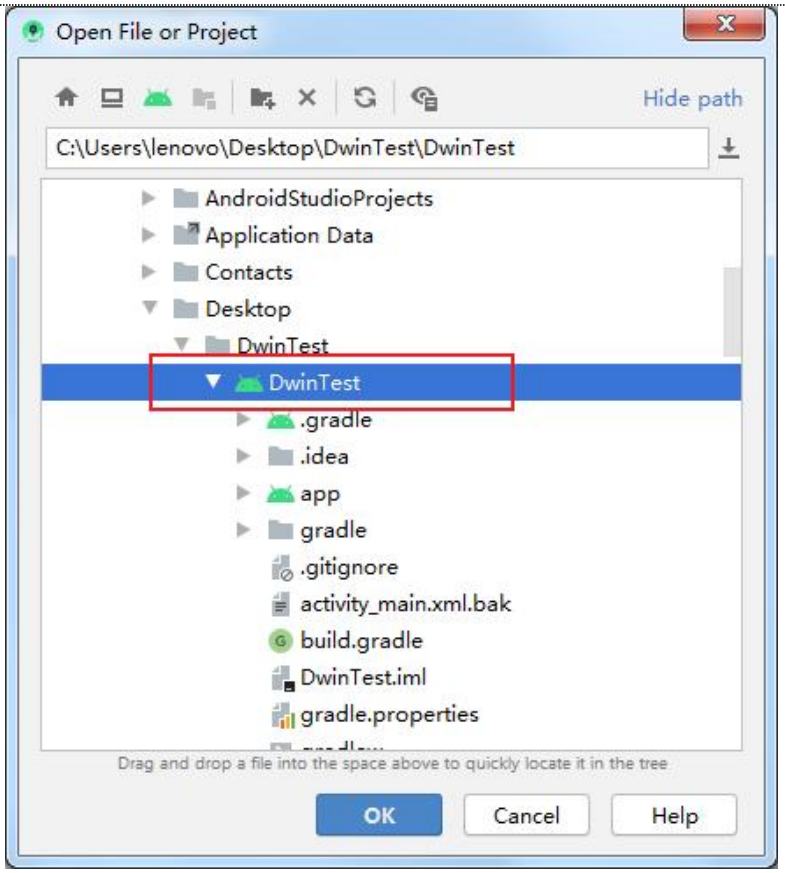
- android_serialport_api: Library utility classes for calling

- DwinTools: Complete project source code for Dwin tools

- DwinTest: Dwin test software

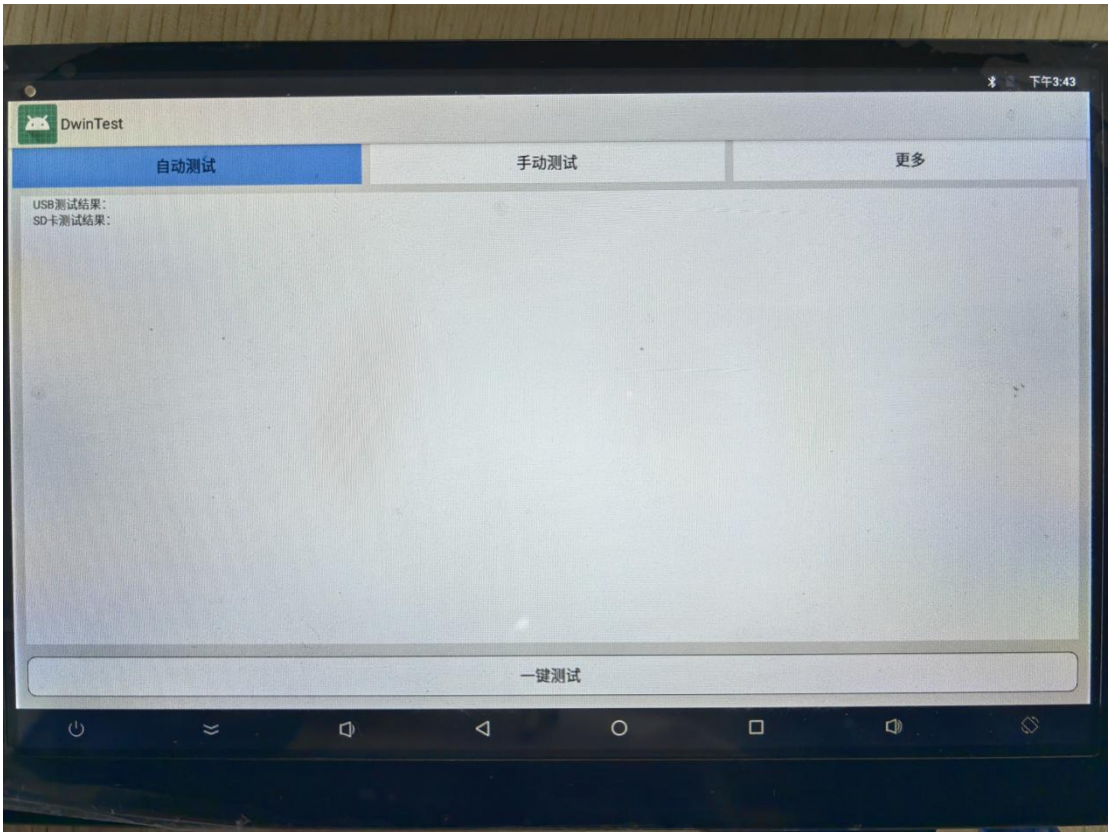
5.2 Importing Dwin Test Project Source Code into Android Studio

(1) In Android Studio, click on "File" in the top left corner, then click "Open." Choose the directory of the source code.



(2) Power up the motherboard and connect the Android screen to the PC using a USB cable.

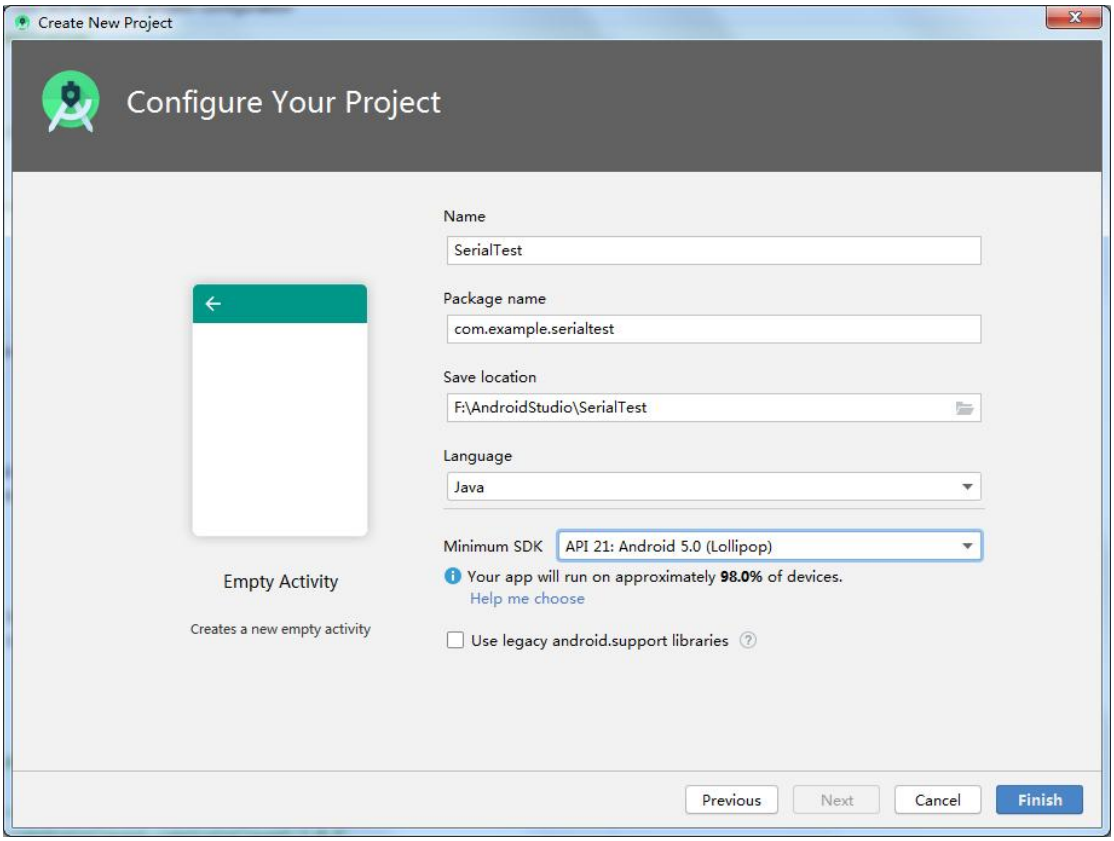
(3) Select the appropriate Android screen, click "Run", and download DwinTest onto the Android screen.



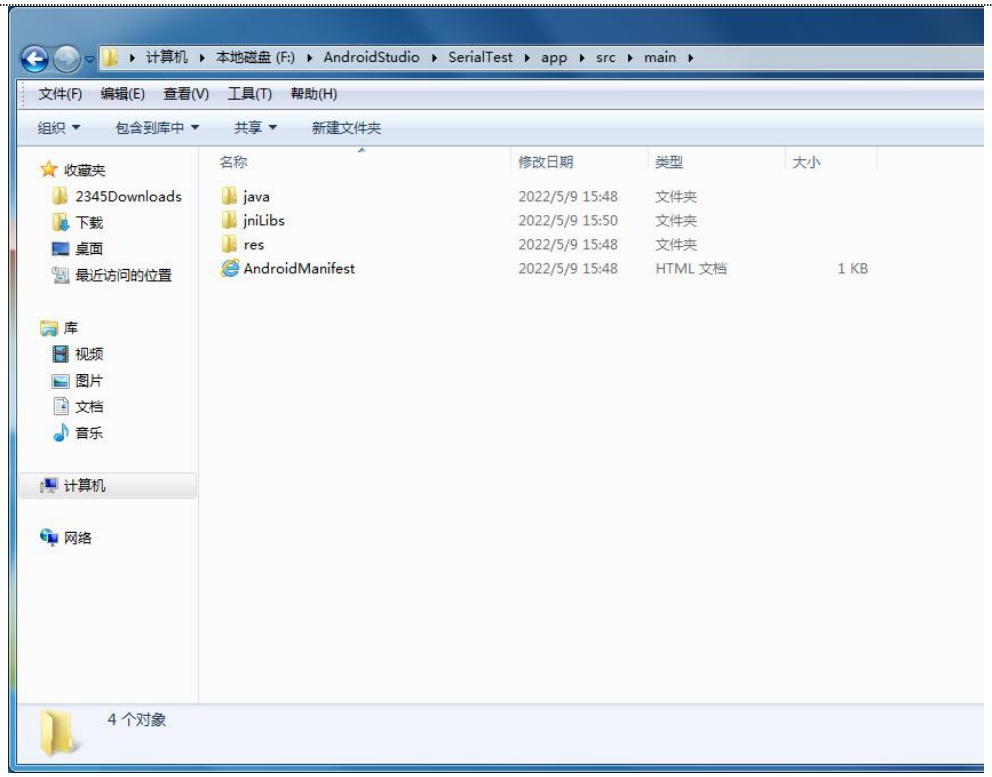
Click [More] -> [Instructions] to see how to use the software for basic testing.

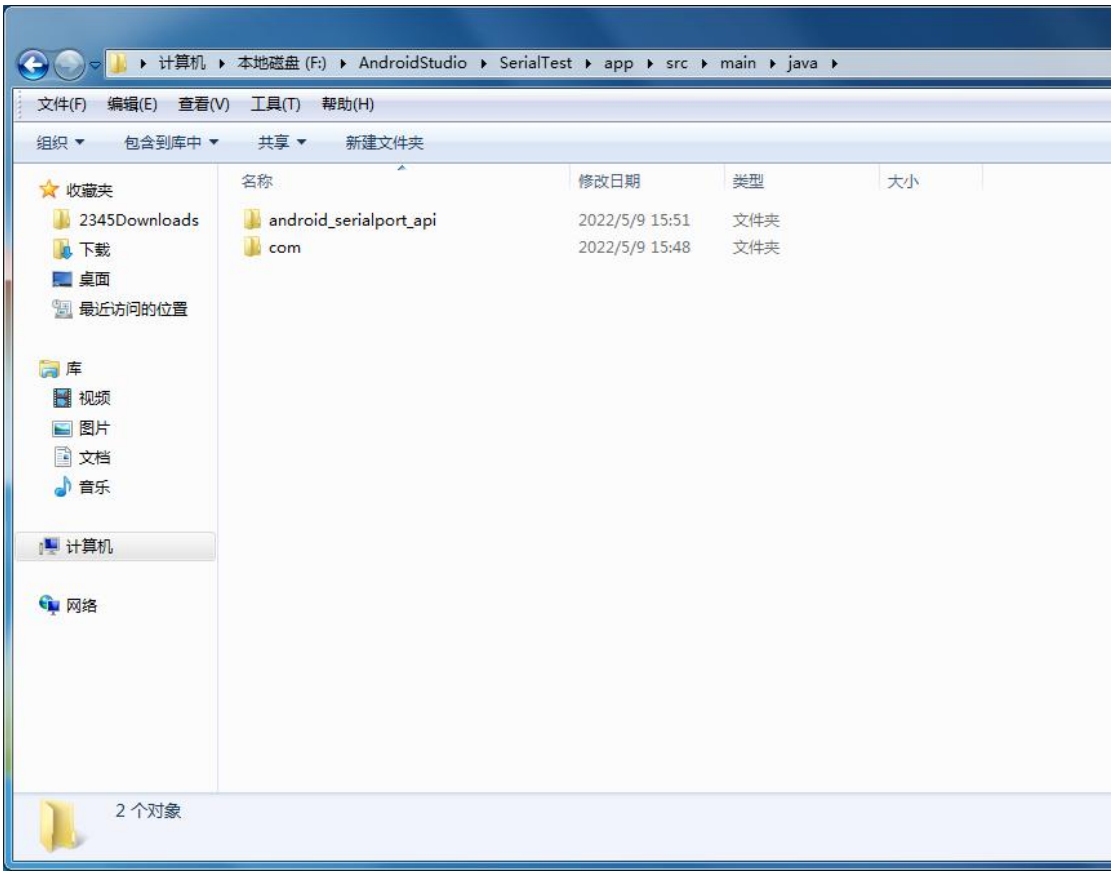
5.3 Using Dwin Java Serial Libraries to Create a New Project

(1) Create a new project. In this example, the project is named "SerialTest, " saved in the path "F:\AndroidStudio\SerialTest, " and the programming language is Java.



(2) Copy the "jniLibs" from the Dwin Android tools package to "F:\AndroidStudio\SerialTest\app\src\main." Also, copy the "android_serialport_api" folder to "F:\AndroidStudio\SerialTest\app\src\main\java." Ensure that the directoy is named "android_serialport_api" and remains unchanged.





(3) After completing the steps, operate the serial port. Specific operations can be referenced in the activities within the DwinTools project source code at "DwinTools\app\src\main\java\com\dwin\DwinTools\serial." Modify and use these according to the project.

-Application.java: It integrates `android.app.Application` and is primarily used to generate SharedPreferences. The configuration of baud rate and serial port devices is completed using SharedPreferences. Generally, no modification is required.

-SerialPortActivity.java: Used to create input/output streams and threads for operating serial port read/write operations. Users can inherit the Activity class for reading and writing.

-SerialPortPreferences.java: This class scans available serial port devices like ttyS* and writes their configurations to SharedPreferences. Users can modify this class based on their own app's requirements.

-ConsoleActivity.java: It is integrated from `SerialPortActivity` and is used for character-based serial communication.

-HexConsoleActivity.java: It is integrated from `SerialPortActivity` and is used for hexadecimal-based serial communication.

-SerialMainMenu.java: This is the main menu activity that facilitates navigation to specific activities.

-Sending01010101Activity.java: This activity constantly sends the sequence "0101" to test the serial waveform.

-LoopbackActivity.java: This activity is used to test whether serial communication for loopback is functioning correctly.

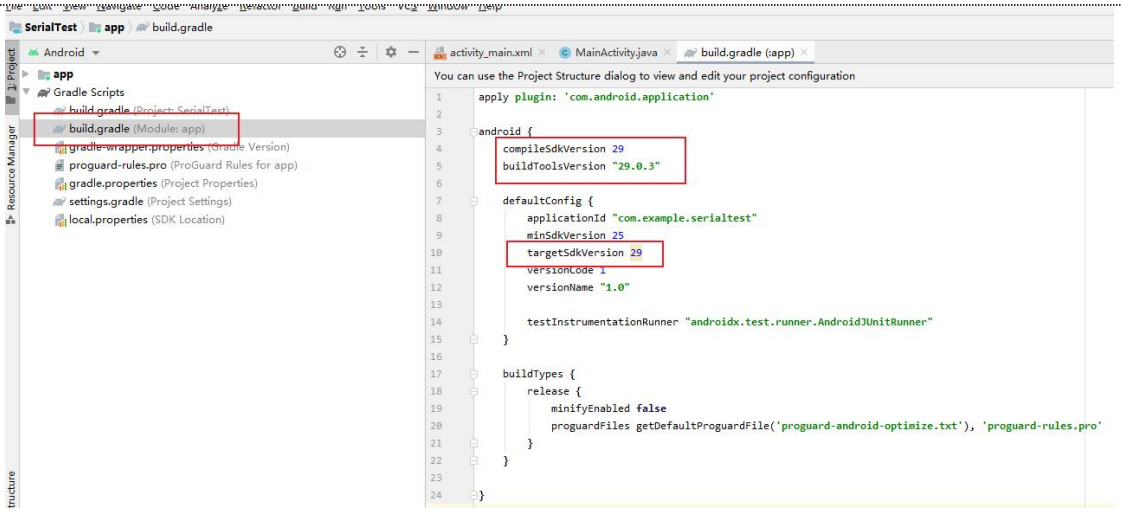
-HexHelper.java: This is a series of utility classes used for converting between characters and hexadecimal representations.

Before using these classes, make sure to update the configuration in the `build.gradle (Module: app)` file as shown in the image.

compileSdkVersion 29

buildToolsVersion "29.0.3"

targetSdkVersion 29



5.4 Example of Using Common Functions of DWIN Android Screen

Most of the special functions of the DWIN Android screen are operated in the form of sending broadcasts.

5.4.1 Dynamic Restart

Just send broadcast android.intent.action.dwin_reboot

Intent sr = **new**

Intent (**"android.intent.action.dwin_reboot"**);

sendBroadcast (sr);

5.4.2 Dynamic Return and the Implementation of Returning Home

Return to send broadcast android.intent.action.dwin_input_back_key

Intent sr = **new** Intent (**"android.intent.action.dwin_input_back_key"**);

sendBroadcast (sr);

home send boardcast android.intent.action.dwin_input_home_key

Intent sr = **new** Intent (**"android.intent.action.dwin_input_home_key"**);

sendBroadcast (sr);

5.4.3 Read chip_id

Use the getChipId () method in the tool class ChipUtil in the DwinTools source code to return, and the source code has been given.

```
case R.id.btn_get_chipid:
    //获取Chip ID
    String chipId = ChipUtil.getInstance().getChipId();
    mTextViewChipId.setText("Chip ID : " + chipId);
    break;
```

5.4.4 Rotate the Screen Through the App

Use the form of sending broadcast to operate. The code is in the RotationActivity class in DwinTools. After the sending is successful, the system will restart and modify the screen rotation.

```
@Override
public void onClick(View v) {
    switch (v.getId()){
        case R.id.btn_rotation_0:
            setRotation("0");
            break;
        case R.id.btn_rotation_90:
            setRotation("90");
            break;
        case R.id.btn_rotation_180:
            setRotation("180");
            break;
        case R.id.btn_rotation_270:
            setRotation("270");
            break;
    }
}
```

```
/**
 * 通过发送广播，系统则会进行屏幕旋转 The system rotates the screen by sending Broadcast
 * @param value 旋转的值 the value of rotation
 */
private void setRotation(String value){
    Log.v(TAG, msg: "Dwin test sendBroadcast!");
    Intent sr = new Intent(DWIN_ROTATION);
    sr.putExtra( name: "message", value);
    sendBroadcast(sr);
}
```

5.4.5 Permanently Hiding the Navigation Bar via App

To permanently hide the navigation bar using the app, user can utilize broadcast messages. The relevant code can be found in the `NavigationBarActivity` class within the `DwinTools` project.

When user send the broadcast successfully, the system will reboot, and the navigation bar status will be modified accordingly.

```
send 0 to show navigation bar, send 1 to hide navigation bar
//发送"0"则显示导航栏，发送"1"则隐藏导航栏
public static final String DWIN_NAVIGATION_BAR_SHOW_VALUE = "0";
public static final String DWIN_NAVIGATION_BAR_HIDE_VALUE = "1";
```

```
/**
 * 通过发送广播，系统则会进行相应的隐藏\显示导航栏 The system hides/shows navigation bar by sending Broadcast
 * @param value 是否显示导航栏的值
 wether to display the value of the navigation bar
 */
private void setNavigationBar(String value){
    Log.v(TAG, msg: "Dwin test sendBroadcast!");
    Intent sr = new Intent(DWIN_NAVIGATION_BAR);
    sr.putExtra( name: "message", value);
    sendBroadcast(sr);
}
```

5.4.6 Setting Up Auto Start on Boot

There are generally two ways to achieve auto-start on boot: one is by receiving the boot broadcast and starting the app, and the other is by setting the APK as the launcher. In the former method, the app starts after booting into the system desktop, while in the latter method, the app bypasses the system desktop and starts directly.

(1) Enter the program after booting the d android interface

When Android starts, it broadcasts a system message with the content `ACTION_BOOT_COMPLETED`, represented as `Android.intent.action.BOOT_COMPLETED`. By capturing this message in the app, user can trigger the app to start. This is typically done by implementing a BroadcastReceiver.

- Interface Activity, MainActivity .java document

```
public class MainActivity extendsActivity {
/** Called when the activity is first created. */
@Override
public void onCreate (Bundle savedInstanceState) {
super.onCreate (savedInstanceState);
// without title
requestWindowFeature (Window. FEATURE_NO_TITLE );
// full screen
getWindow ().setFlags (WindowManager.LayoutParams. FLAG_FULLSCREEN ,
WindowManager.LayoutParams. FLAG_FULLSCREEN );
setContentView (R.layout. activity_main );
}
}
```

This code is very simple. When the Activity starts, the TextView will be displayed, use it to display the words user want to display.

- Receive broadcast messages

```
public class BootBroadcastReceiver extends BroadcastReceiver {
static final String action_boot = "android.intent.action.BOOT_COMPLETED";
```

```
@Override

public void onReceive (Context context, Intent intent) {

if (intent.getAction ().equals ( action_boot )) {

Intent ootStartIntent = new Intent (context, MainActivity. class);

ootStartIntent.addFlags (Intent. FLAG_ACTIVITY_NEW_TASK );

context.startActivity (ootStartIntent);

}

}

}
```

This class inherits from BroadcastReceiver. In the overridden method onReceive, check whether the received Intent conforms to BOOT_COMPLETED. If so, start the MainActivity.

- Configuration file

AndroidManifest.xml :

```
<?xml version= "1.0" encoding= "utf-8" ?>

<manifest xmlns: android= "http: //schemas.android.com/apk/res/android"

package= "com.ajie.bootstartdemo"

android: versionCode= "1"

android: versionName= "1.0" >

<uses-sdk

android: minSdkVersion= "8"

android: targetSdkVersion= "17" />

<application

android: allowBackup= "true"
```

```
android: icon= "@drawable/ic_launcher"

android: label= "@string/app_name"

android: theme= "@style/AppTheme" >

<activity

android: name= "com.dwin.bootstartdemo.MainActivity"

android: label= "@string/app_name" >

<intent-filter>

<action android: name= "android.intent.action.MAIN" />

<category android: name= "android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<receiver android: name= "com.dwin.bootstartdemo.BootBroadcastReceiver" >

<intent-filter>

<action android: name= "android.intent.action.BOOT_COMPLETED" />

<category android: name= "android.intent.category.HOME" />

</intent-filter>

</receiver>

</application>

<uses-permission android: name= "android.permission.RECEIVE_BOOT_COMPLETED" >

</uses-permission>

</manifest>
```

Please note the part highlighted in red. This section registers a receiver with the system. The sub-node intent-filter indicates that it is set to receive the android.intent.action.BOOT_COMPLETED message.

Additionally, the android.permission.RECEIVE_BOOT_COMPLETED permission must be configured.

After completion, compile the apk package and install it on the Android screen. Shut down and restart, and the page displayed by the MainActivity will be displayed. DWIN Android broadcast receiving is operated by the default interface of the Android system, and uses the standard Android API function library. For specific usage methods, please refer to the Android official website API Guides.

(2) Boot to user-selected android interface or application

To achieve boot self-starting is mainly to modify the application program to have the authority of laucher.

Replace the <intent-filter> of the first started activity in the configuration file

AndroidManifest.xml:

```
<?xml version= "1.0" encoding= "utf-8" ?>
<manifest xmlns: android= "http: //schemas.android.com/apk/res/android"
package= "com.ajie.bootstartdemo"
android: versionCode= "1"
android: versionName= "1.0" >
<uses-sdk
android: minSdkVersion= "8"
android: targetSdkVersion= "17" />
<application
android: allowBackup= "true"
android: icon= "@drawable/ic_launcher"
android: label= "@string/app_name"
android: theme= "@style/AppTheme" >
<activity
```

```
android: name= "com.dwin.bootstartdemo.MainActivity"

android: label= "@string/app_name" >

<intent-filter>

<action android: name= "android.intent.action.MAIN" />

<category android: name= "android.intent.category.HOME" />

<category android: name= "android.intent.category.DEFAULT" />

<category android: name= "android.intent.category.MONKEY" />

</intent-filter>

</activity>

</application>

</manifest>
```

After running the program, click on the Home button. A Launcher selection box will pop up. Select user developed APK and click "Always."

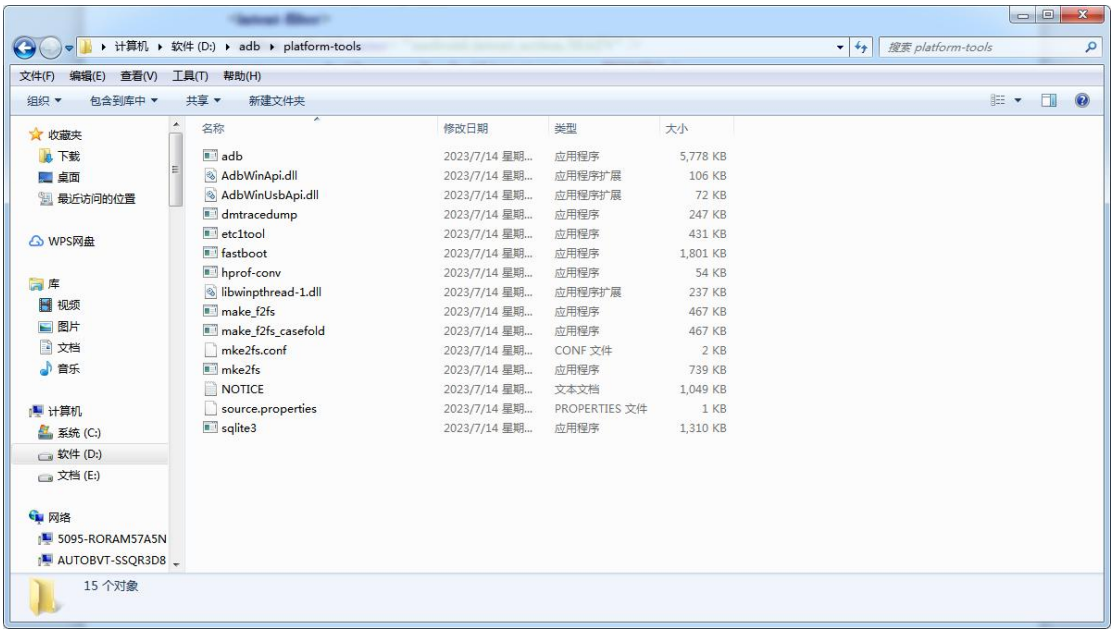
6 ADB Installation and Usage

6.1 Installation on PC

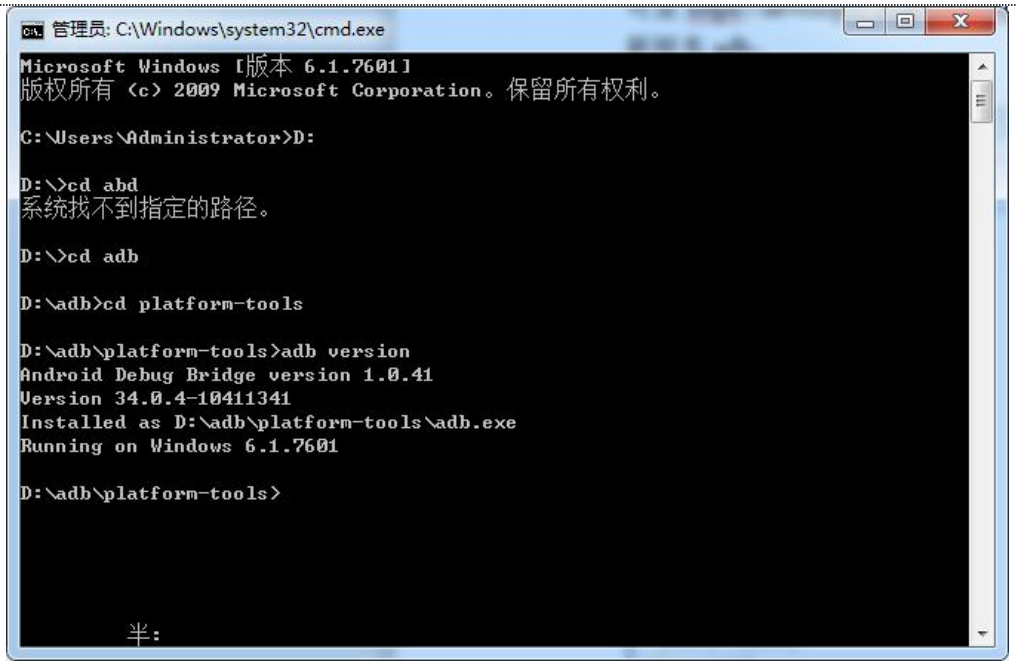
Download the platform-tools adb version tool.

User can obtain the latest version of adb from this website: https://developer.android.google.cn/studio/releases/platform-tools?hl=zh-cn.

Once downloaded, extract it to the local directory.



Then, open the Run dialog (Win+R), type "cmd, " and press Enter. Use commands to perform operations. Input "adb version" to check the adb version and confirm if the installation was successful.

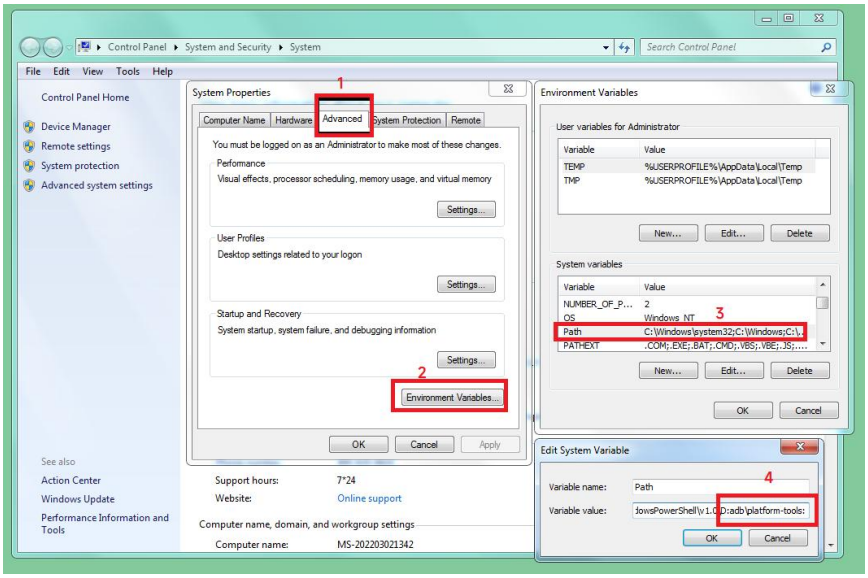


6.2 Debugging Using Android Studio

Add Environment Variable:

- Open [My Computer] → Click on [Properties] → Select [Advanced system settings] → Click on [Environment Variables].

- Edit the system variable "Path" and Add the adb installation path after the variable value as shown in the image below:



This will enable user to use the adb command for debugging within the Android Studio Terminal.

7 New Firmware

The 32 series' firmware underwent a unified functional upgrade on September 4, 2024, providing a series of SDKs and interfaces that facilitate secondary development for customers. These interfaces cannot be called in old firmware. If you need to update to the new firmware, please contact your sales person for assistance.

There are several ways to check your firmware version: 1. Check the product model label. Products with a production date after September 4, 2024 are equipped with new firmware products; 2. Check the firmware compilation date in the software. The firmware with a compilation date of 20240902 is the new version firmware.

This chapter will introduce the content of the new firmware.

Firmware Update Records		
Firmware Name	Release Date	Content
Update.img	Before 2024.9.4	Include some simple example such as how to hide the navigation bar
Update.img	2024.9.4	Include common Android functions such as: example app, GPIO, how to use API etc.
Update.img	2024.11.3-2024.11.13	Update DWINTEST and resolve the problem of boot animation stuttering

7.1 SDK

The following development can refer to the DEMO named DWAndroidLibraryDemo. Please contact the salaes or technical support team to obtain it.

7.1.1 Boot Logo

The boot logos are divided into ubootlogo and kernellogo. Generally, they are replaced simultaneously. The image format is BMP. Please note the following two points:

-When replacing and compiling the boot logo in the system source code or replacing it using API interface, the image resolution should not exceed the screen resolution, and it should be controlled within 2M as much as possible.

-When using FWFactoryTool to modify the firmware package to replace the boot logo, the image resolution should not exceed 600*600 pixels, and the quality should be less than 1MB.

```
 * @param path  Logo path
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setBootLogo (String path);
```

Note: Re-burning firmware or restoring factory settings will not delete the logo. If you need to delete the logo during burning, please use the "erase" function of the burning tool first.

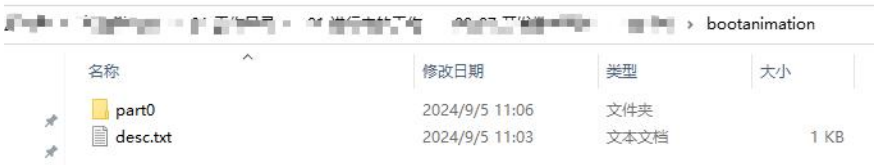
```
/**
 * Delete boot logo
 *
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int deleteBootLogo ();
```

7.1.2 Boot Animation (Multiple pictures)

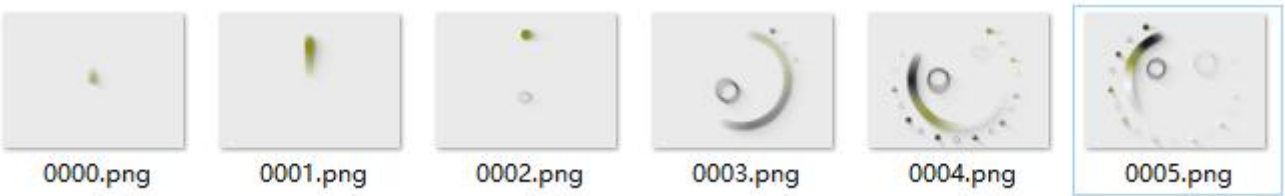
```
 * @param path  Boot animation path
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setBootAnimation (String path);
```

7.1.2.1 Create Bootanimation.zip

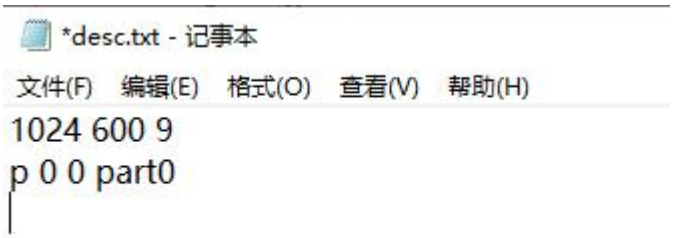
Bootanimation.zIP contains part0 folder and desc.txt, or also can contain 2 folders as part0, part1, here we only take part0 as example.



Create a new folder and name it as part0, put the animation pictures in it (.png format), naming pictures with 000.png, 0001.png, 0002.png and so on. Please note that the image resolution should not exceed the screen resolution, and the memory for a single image should be controlled within 2M. If the images are too large or too many, it may cause the startup animation to be unsmoothly or freezed.



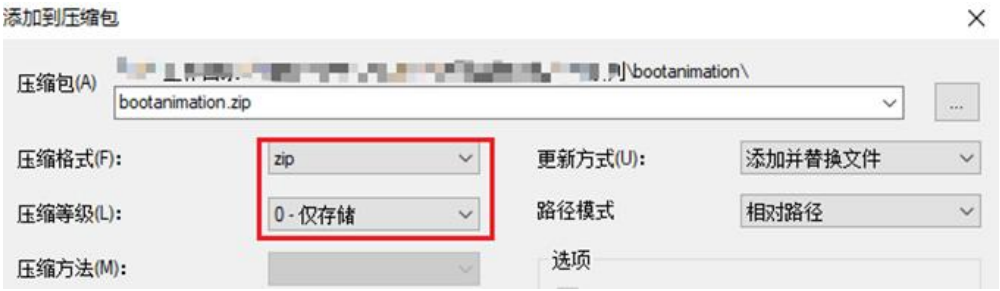
Create a new txt document and name it as desic.txt, Manually input the following content. Please note that after the last line, enter a blank line and save the file. Otherwise, the device will not be able to parse it.



First line: 1024 600 9: 1024 and 600 means resolution is 1024*600p, 9 represent display 9 pictures every second;

Second line: pis a fixed starting point, The first number represent repeated times, 0 is infinite playback (if 3, it will play three times); The second 0 represents the interval time/frame rate between the two frames ; The last part0 represents the animation folder.

Add part0 and desc.txt together to the compressed file", compressed file format"zip", compression method select "Storage". After compression is complete, please re-open it and ensure there is no bootanimation folder inside.



7.1.2.2 Delete Bootanimation

Note: Re-burning firmware or restoring factory settings will not delete the boot animation. If you need to delete the bootanimaton during burning, please use the "erase" function of the burning tool first.

```
/**
 * Delete bootanimation (multiple pictures)
```

```
     *
     * @return Return value of interface calling. Refer to DWErrorCode
     */
    int deleteBootAnimation ();
```

7.1.3 Startup Video

Boot Video Format: Change the file extension of the video to .ts. The video could be approximately ten seconds long.

If both a boot animation (multiple images) and a start-up video are present, the system will preferentially display the boot video.

```
     * @param path  Startup video path
     * @return  Return value of interface calling. Refer to DWErrorCode
     */
    int setBootVideo (String path);
```

Delete the startup video as follows:

```
        * Delete the startup video
         *
         * @return  Return value of interface calling. Refer to DWErrorCode
         */
        int deleteBootVideo ();
```

7.1.4 Monitoring and Reception of Boot Broadcast

Regarding boot monitoring, the broadcast is used to monitor self-starting apps.

```
    public class SelfStartReceiver extends BroadcastReceiver {

    @Override

    public void onReceive(Context context, Intent intent) {

        //Android  A boot broadcast will be sent when the device is powered on: "android.intent.action.BOOT_COMPLETED"

        if (Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) {

            Log.d("SelfStartReceiver", "The boot broadcast has been detected");
```

```
            Intent i = new Intent(context, MainActivity.class);

            i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

            context.startActivity(i);

        }

    }

}
```

Please refer to the following code for broadcast settings and permission configurations:

MainActivity.java | AndroidManifest.xml | SelfStartReceiver.java

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.dwin.selfstartdemo">

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.SelfStartDemo">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver
            android:name=".SelfStartReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter android:priority="100">
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

7.1.5 Top Status Bar

Can display or hide the top status bar, read the status of the top status bar.

```
/**
 * Display the top status bar
 *
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int showStatusBar ();

/**
 * Hide the top status bar
 *
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int hideStatusBar ();

/**
 * Read the status of the top status bar, display or hide
 *
 * @return Return value of interface calling.1: display, 0: hide
 */
int isStatusBarShow ();
```

7.1.6 Bottom Navigation Bar

Allows for showing, hiding and status checking of bottom navigation bar and the integrated screenshot button.

```
/**
* Display bottom navigation bar
*
* @return  Return value of interface calling. Refer to DWErrorCode
*/
int showNavigationBar ();

/**
* Hide bottom navigation bar
*
* @return  Return value of interface calling. Refer to DWErrorCode
*/
int hideNavigationBar ();

/**
* read the status of the bottom navigation bar, display or hide
```

```
 *
 * @return  Return value of interface calling.1: display, 0: hide
 */
int isNavigationBarShow ();

/**
 * Display the screenshot button in the bottom navigation bar
 *
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int showScreenShotButton ();

/**
 * Hide the screenshot button in the bottom navigation bar
 *
 * @return  Return value of interface calling. Refer to DWErrorCode
/**
int hideScreenShotButton ();

/**
  * Read the status of the bottom navigation bar, display or hide
  *
  * @return  Return value of interface calling.1: display, 0: hide
  */
     int isScreenShotShow ();
```

7.1.7 Top Dropdown Box

Display or hide the top dropdown box, read the status of the top dropdown box.

```
/**
  * Display the top dropdown box
  *
  * @return  Return value of interface calling. Refer to DWErrorCode
  */
int showDropDownMenu ();

/**
  * Hide the top dropdown box
  *
  * @return  Return value of interface calling. Refer to DWErrorCode
  */
int hideDropDownMenu ();

/**
  * Read the status of top dropdown box, display or hide
```

```
 *
 * @return  Return value of interface calling. 1: display, 0: hide
 */
int isDropDownMenuShow ();
```

7.1.8 Install APK Silently

```
/**
 * Install APK silently
 *
 * @param path  APK path
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int installApp (String path);
```

7.1.9 Read External Storage Path

```
/**
 * Read external storage path
 *
 * @return  Return path collection
 */
ArrayList<StorageBean> readExternalStoragePath ();
```

7.1.10 Set System Language

```
/**
 * Set system language
 *
 * @param locale  Language format Locale.SIMPLIFIED_CHINESE, Locale.ENGLISH etc.
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setLanguage (Locale locale);
```

7.1.11 Set System Time

```
/**
 * Set system time
 *
 * @param hour  Hour
 * @param minute  Minute
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setTime (int hour, int minute);

/**
 * Set system time
 *
 * @param year  Year
 * @param month  Month
 * @param day  Day
 * @param hour  Hour
 * @param minute  Minute
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setTime (int year, int month, int day, int hour, int minute);

/**
 * Enable automatic synchronization of time
 *
 * @param auto  1: Automatically synchronizing time, 0: Not automatically synchronizing time
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setTimeAuto (int auto);
```

7.1.12 Set System Date

```
/**
 * Set system date
 *
 * @param year  Year
 * @param month  Month
 * @param day  Day
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setDate (int year, int month, int day);
```

7.1.13 Time Zone and Time Setting

Set time zone, enable automatic time zone synchronization, check if automatic synchronization function is enabled, set 12 or 24-hour clock.

```
/**
 * Set time zone
 *
 * @param timeZone  Time Zone Name, such as Asia/Shanghai
 */
void setTimeZone (String timeZone);

/**
 * Enable automatic time zone synchronization
 *
 * @param auto  1: Automatically synchronize time zones, 0: Not automatically synchronize time zones
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
   int setTimeZoneAuto (int auto);

/**
 * Check if automatic time zone synchronization is enabled
 *
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int checkTimeZoneAuto ();

/**
 * Set "12" or "24" hour clock
 *
 * @param value  "12" or "24"
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setTime_12_24 (String value);
```

7.1.14 Set Ethernet MATIC Network Parameters and DHCP

```
/**
 * Set ethernet MATIC network parameters
 *
 * @param address  IP
 * @param mask  Subnet mask code
 * @param gateway  Gateway
 * @param dns1  DNS1
 * @param dns2  DNS2
```

```
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setEthernetStaticConfig (String address, String mask, String gateway, String dns1, String dns2);

/**
 * Set Ethernet DHCP
 *
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setEthernetDynamicConfig ();
```

7.1.15 Set Wi-Fi STATIC Network Parameters and DHCP

```
/**
 * Set WiFi STATIC parameters
 *
 * @param address  IP
 * @param mask  Mask
 * @param gateway  Gateway
 * @param dns1  DNS1
 * @param dns2  DNS2
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setWiFiStaticConfig (String address, String mask, String gateway, String dns1, String dns2);

/**
 * Set WiFi DHCP
 *
 * @return  Return value of interface calling. Refer to DWErrorCode
 */
int setWiFiDynamicConfig ();
```

7.1.16 Get IP, Gateway, DNS, Network Mask

```
/**
 * Get IP
 *
 * @return  Return to IP collection
 */
ArrayList<String> getNetWorkIP ();

/**
 * Get gateway
 *
```

```
     * @return  Return to gateway collection
    */
    ArrayList<String> getNetWorkGateway ();

    /**
     * Get DNS
     *
     * @return  Return to DNScollection
     */
    ArrayList<String> getNetWorkDns ();

    /**
     * Get Network Mask
     *
     * @return Return to Network Mask collection
     */
    ArrayList<String> getNetworkMask ();
```

7.1.17 Reboot System

```
    /**
     * Reboot System
     *
     * @return Return value of interface calling. Refer to DWErrorCode
     */
        int rebootSystem ();
```

7.1.18 Return Value of Interface Calling

Interface calling return value reference: `DWErrorCode`, only for Android SDK.

```
    /**
     * Interface calling return code
     */
    public class DWErrorCode {

    /**
     * Succeed
     */
    public final static int DW_COMMON_SUCCESS = 0;

    /**
     * unknown error
     */
    public final static int DW_COMMON_ERROR_UNKNOUN = -1;
```

```
/**
 * Android SDK do not support it（perhaps due to firmware mismatch）
 */
public final static int DW_COMMON_ERROR_SDK_NOT_SUPPORT = -2;

/**
 * Settings.Global.putXXX Writing value failed
 */
public final static int DW_COMMON_ERROR_SET_SETTINGS_GLOBAL = -3;

/**
 * Settings.Global.getXXX Reading value failed
 */
public final static int DW_COMMON_ERROR_GET_SETTINGS_GLOBAL = -4;

/**
 * Settings.System.putXXX Writing value failed
 */
public final static int DW_COMMON_ERROR_SET_SETTINGS_SYSTEM = -5;

/**
 * Settings.System.getXXX Reading value failed
 */
public final static int DW_COMMON_ERROR_GET_SETTINGS_SYSTEM = -6;

/**
 * Path is empty or null
 */
public final static int DW_COMMON_ERROR_PATH_INVALID = -7;

/**
 * Path is not exist
 */
public final static int DW_COMMON_ERROR_PATH_NOT_EXIST = -8;

/**
 * Copy file error
 */
public final static int DW_COMMON_ERROR_COPY_FILE = -9;

/**
 * Deleting file failed
 */
public final static int DW_COMMON_ERROR_DELETE_FILE = -10;
```

```
/**
 * bootlogo is not a .BMP file
 */
public final static int DW_BOOT_LOGO_ERROR_NOT_BMP_FILE = -100;

/**
 * bootanimation is not a .zip or .ts file
 */
public final static int DW_BOOT_ANIMATION_ERROR_NOT_ZIP_OR_TS_FILE = -101;

/**
 * the installed file is not an APK file
 */
public final static int DW_INSTALL_APP_ERROR_NOT_APK_FILE = -200;

/**
 * installation failed
 */
public final static int DW_INSTALL_APP_ERROR = -201;

/**
 * Failed to read external file
 */
public final static int DW_EXTERNAL_STORAGE_READ_ERROR = -300;

/**
 * Failed to set system language
 */
public final static int DW_LANGUAGE_SET_ERROR = -400;

/**
 * Failed to set system time or date
 */
public final static int DW_TIME_OR_DATE_SET_ERROR = -500;

/**
 * Failed to configure Ethernet STATIC network
 */
public final static int DW_ETHERNET_STATIC_SET_ERROR = -600;

/**
 * Failed to configure Ethernet DHCP network
 */
public final static int DW_ETHERNET_DYNAMIC_SET_ERROR = -601;
```

```
    /**
     * Failed to configure WiFi STATIC network
     */
    public final static int DW_WIFI_STATIC_SET_ERROR = -602;

    /**
     * Failed to configure WiFi DHCP network
     */
    public final static int DW_WIFI_DYNAMIC_SET_ERROR = -603;

    /**
     * System restart failed
     */
    public final static int DW_REBOOT_SYSTEM_ERROR = -600;
}
```

7.2 SerialPort

Serial port calling method: SerialManage.getInstance ().xxx, such as SerialManage.getInstance ().init

7.2.1 Serial Port Initialization

```
    /**
     *Serial port initialization
     *
     * @param serialInter Serial callback
        */
       public void init (SerialInter serialInter)
```

7.2.2 Open Serial Port

```
    /**
     * Open serial port
     *
     * @param devicesPath      Serial port address
     * @param isReadLoop       Whether to continuously monitor the data returned by the serial port
     * @param baudrate         Baud-rate
     * @param dataBits         Data Bits
     * @param stopBits         Stop Bits
     * @param parity           Parity
     * @param readTimePeriod   ReadTimePeriod
     * @return                 Open successfully or not
     */
```

```
    public void open (String devicesPath, boolean isReadLoop, int baudrate, int dataBits, int stopBits, int parity, long
readTimePeriod)
```

7.2.3 Send Serial Port Data

```
    /**
     * Send serial port data
     *
     * @param msg Serial port data
     */
        public void send (byte[] msg)
```

7.2.4 Close Serial Port

```
    /**
     * Close serial port
     */
        public void close ()
```

7.2.5 Release Resources

```
    /**
     * Release resources
     */
    public void release ()
```

7.2.6 Obtain Supported Serial Ports

```
    /**
     * Obtain supported serial ports
     *
     * @return Return to serial port collection

     */
    public String[] getDriverList ()
```

7.2.7 Serial callback

```
    /**
     * Serial callback
     */
    public interface SerialInter {

    /**
     * Connection result callback
```

```
     * @param path  Serial port address ( (when multiple serial ports need to be processed uniformly, addresses can be used
to distinguish them)
     * @param isSuccess  Connection successful or not
     */
    void connectMsg (String path,boolean isSuccess);

    /**
     * Loop read, callback the data read
     * @param path  Serial port address ( (when multiple serial ports need to be processed uniformly, addresses can be used
to distinguish them)
     * @param bytes  Read data
     * @param size  Data Length
     */
    void readData (String path, byte[] bytes, int size);
```

7.3 GPIO

Gpio interface calling method: Gpio.xxxx, such as Gpio.initGpio

7.3.1 Initialize GPIO

```
    /**
     * Initialize GPIO
     *
     * @param gpioArr  GPIO port that needs to be configured, just a numerical number is enough
     * @param value  0:low, 1:high
     * @param direction  In:input, out:output
     * @param context  Above and follow context
     * @param gpioInitCallBack  Initialize callback interface
     */
    public static void initGpio (int[] gpioArr, int value, String direction, Context context, GpioInitCallBack
gpioInitCallBack)
```

7.3.2 Set Value of GPIO

```
    /**
     * Set value of GPIO
     *
     * @param gpio  GPIO port, such as gpio100, value is 100
     * @param value  Set the voltage level, 0: low, 1: high
     * @return  Return value, 0: success, <0:fail
     */
    public static native int setGpioValue (int gpio, int value);
```

7.3.3 Set GPIO Direction

```
/**
 * Set GPIO direction
 *
 * @param gpio  GPIO port, such as gpio100, value is 100
 * @param direction  Set gpiodirection, 0:in; 1:out
 * @return  Return value, 0: success, <0:fail
 */
private static native int setGpioDirection (int gpio, int direction);
```

7.3.4 Read GPIO Direction

```
/**
 * Read GPIO  directio
 *
 * @param gpio  GPIO port, such as gpio100, value is 100
 * @return  Return value GPIO direction (in/out)
*/
private static native int getGpioDirections (int gpio);
```

7.3.5 Read GPIO Value

```
/**
 * Read GPIO value
 *
 * @param gpio  GPIO port, such as gpio100, value is 100
 * @return  Return value
 */
public static native int getGpioValue (int gpio);
```

7.3.6 Initialization CallBack

```
public interface GpioInitCallBack {
/**
 * Initialization callBack method
 *
 * @param success Initialization succeed
 * @param gpio Collection of successfully initialized GPIO ports
 */
void initState (boolean success, List<Integer> gpio);
}
```