



Hi3861 V100 / Hi3861L V100 Upgrade

Development Guide

Issue	01
Date	2020-04-30

Copyright © HiSilicon (Shanghai) Technologies Co., Ltd. 2020. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon (Shanghai) Technologies Co., Ltd.

Trademarks and Permissions



HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided “AS IS” without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road,
Bantian, Longgang District,
Shenzhen 518129 P. R. China

Website: <https://www.hisilicon.com/en/>

Email: support@hisilicon.com



About This Document

Purpose

This document describes how to use the application programming interfaces (APIs) of upgrade.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3861	V100
Hi3861L	V100



Intended Audience

The document is intended for:



- Technical support engineers
- Software development engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
	Indicates a hazard with a medium level of risk which, if not avoided, could result in death or serious injury.



Symbol	Description
 CAUTION	Indicates a hazard with a low level of risk which, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.
 NOTE	Supplements the important information in the main text. NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration.

Change History

Issue	Date	Change Description
01	2020-04-30	This issue is the first official release. <ul style="list-style-type: none">In 1 Introduction, the description of the dual-partition upgrade mode and compression upgrade mode is updated.In Table 2-1 of 2 Development Process, the description of <code>hi_upg_transmit</code> and <code>hi_upg_transmit_finish</code> is updated.In 4 Programming Samples, Kernel Upgrade Sample and FlashBoot Upgrade Sample are updated.
00B03	2020-04-03	<ul style="list-style-type: none">In 1 Introduction, the description about how to select the dual-partition upgrade mode or compression upgrade mode by using OTA Settings in menuconfig is added. The description of the registration callback API is added.In Table 2-1 of 2 Development Process, the description of <code>hi_upg_get_file_index</code> is updated, and the description of <code>hi_upg_register_file_verify_fn</code> is added.In 3 Precautions, the precaution for using <code>hi_upg_register_file_verify_fn</code> is added.In 4 Programming Samples, the description of obtaining the app upgrade file ID in dual-partition upgrade mode is updated.



Issue	Date	Change Description
00B02	2020-02-12	<ul style="list-style-type: none">• In Table 2-1, the description of hi_upg_init is added.• In 3 Precautions, the precaution for using hi_upg_init is added.
00B01	2020-01-15	This issue is the first draft release.



Contents

About This Document.....	i
1 Introduction.....	1
2 Development Process.....	3
3 Precautions.....	5
4 Programming Samples.....	6



1 Introduction

For details about the flash partitions, see section 2.10 "Flash Partitions and Protection" in the *Hi3861 V100/Hi3861L V100 SDK Development Guide*. The kernel or FlashBoot can be upgraded by calling the upgrade APIs.

- Choose **OTA Settings > dual-partition ota support** (dual-partition upgrade mode) in menuconfig. During kernel upgrade, if kernel A is running, the upgrade file of kernel B is obtained and updated to kernel B. After the upgrade, the system reboots from kernel B. If kernel B is running, the upgrade file of kernel A is obtained and updated to kernel A. After the upgrade, the system reboots from kernel A.
- Choose **OTA Settings > compression ota support** (compression upgrade mode) in menuconfig. During kernel upgrade, the compressed upgrade file is updated to kernel B. After upgrade and reboot, decompress the compressed file of kernel B in FlashBoot. The compressed file is updated to kernel A, and the system boots from kernel A.
- During the upgrade of FlashBoot, the obtained FlashBoot upgrade file is updated to the backup FlashBoot area. After the verification is successful, the file is updated to the FlashBoot area. After upgrade and restart, new FlashBoot is used.

The upgrade APIs are classified into the following types based on their functions:

- APIs used to control the upgrade process, including:
 - hi_upg_transmit
 - hi_upg_transmit_finish
 - hi_upg_finish
 - hi_upg_stop
- APIs used to obtain information about upgrade files, including:
 - hi_upg_get_file_index
 - hi_upg_get_max_file_len
 - hi_upg_get_content
- Registration callback API used to control or learn the current upgrade process, including:
 - hi_upg_register_file_verify_fn



This registration API is called during task initialization. When the actual upgrade process is executed, the callback function registered by the user is called.



2 Development Process

Application Scenario

- Kernel upgrade
- FlashBoot upgrade

Function

Table 2-1 describes the upgrade APIs.

Table 2-1 Description of upgrade APIs

API Name	Description
hi_upg_init	Initializes the upgrade status.
hi_upg_transmit	Transfers the upgrade file.
hi_upg_transmit_finish	Finishes upgrade file transmission.
hi_upg_finish	Finishes upgrade.
hi_upg_stop	Stops upgrade.
hi_upg_get_file_index	Obtains the ID of the kernel upgrade file during kernel upgrade in dual-partition upgrade mode. 1: kernel A upgrade file 2: kernel B upgrade file
hi_upg_get_max_file_len	Obtains the maximum length of the upgrade file.
hi_upg_get_content	Obtains the upgrade file content.
hi_upg_register_file_verify_fn	Registers the API for verifying the validity of the user-defined upgrade file. A 32-byte user-defined field is reserved in the upgrade file and can be used together with this API.



Development Process

The development process of upgrade in typical scenarios is as follows:

Step 1 Transmit the upgrade file by calling **hi_upg_transmit**. If the API returns an error, the upgrade is stopped.

Step 2 Notify the upgrade module that the transmission is complete by calling **hi_upg_transmit_finish**. If the API returns an error, the upgrade is stopped.

Step 3 Finish upgrade by calling **hi_upg_finish**.

----End



3 Precautions

- For **hi_upg_transmit**, the length of the first transmission packet must be greater than or equal to 96 bytes.
- For **hi_upg_transmit**, upgrade files must be transmitted based on packages and in sequence.
- For **hi_upg_transmit**, the same packet cannot be transmitted repeatedly.
- **hi_upg_transmit_finish** must be called after the upgrade file is transmitted and cannot be called repeatedly.
- Do not power off the system during FlashBoot upgrade, which cannot be stopped.
- **hi_upg_get_content** must be called after **hi_upg_transmit** and before **hi_upg_transmit_finish**.
- The upgrade APIs cannot be called during interrupts. Apps cannot be called at the same time.
- **hi_upg_init** must be called only after **hi_nv_init**. By default, **hi_upg_init** is called by the app in the SDK. For details, see the **app_main** function.
- **hi_upg_register_file_verify_fn** must be called before the upgrade process is started.



4 Programming Samples

Kernel Upgrade Sample

```
hi_u32 file_size = 0x2000; /* Upgrade file size (obtained from the app) */
hi_u32 max_len;
hi_u8 file_index;

/* 1. Obtain the maximum size of the app upgrade file. */
hi_u32 ret = hi_upg_get_max_file_len(HI_UPG_FILE_KERNEL, &max_len);
if ((ret != HI_ERR_SUCCESS) || (file_size > max_len)) {
    return HI_ERR_UPG_FILE_LEN;
}

/* 2. In dual-partition upgrade mode, obtain the app upgrade file ID to determine the upgrade
file to be transferred. Skip this step if the compression upgrade mode is used.*/
ret = hi_upg_get_file_index(&file_index);
if (ret != HI_ERR_SUCCESS) {
    return ret;
}

/* 3. Load the corresponding upgrade file through the network port or serial port (implemented
by users).
Transmit the upgrade file to the upgrade module by calling hi_upg_transmit. */

/* 4. After the transmission is complete (hi_upg_transmit_finish), if an error is returned, stop
the upgrade. */
ret = hi_upg_transmit_finish();
if (ret != HI_ERR_SUCCESS) {
    /*Stop the upgrade. */
    hi_upg_stop();
    ...
}

/* 5. Finish upgrade (hi_upg_finish). */
hi_upg_finish();
```

FlashBoot Upgrade Sample

```
hi_u32 file_size = 0x2000; /* Actual size of the upgrade file (obtained from the app) */
hi_u32 max_len;

/* 1. Obtain the maximum size of the app upgrade file. */
hi_u32 ret = hi_upg_get_max_file_len(HI_UPG_FILE_BOOT, &max_len);
if ((ret != HI_ERR_SUCCESS) || (file_size > max_len)) {
    return HI_ERR_UPG_FILE_LEN;
}
```



```
/* 2. Load the corresponding upgrade file through the network port or serial port (implemented  
by users).  
Transmit the upgrade file to the upgrade module by calling hi_upg_transmit. */  
  
/* 3. After the transmission is complete (hi_upg_transmit_finish), if an error is returned, stop  
the upgrade. */  
ret = hi_upg_transmit_finish();  
if (ret != HI_ERR_SUCCESS) {  
/*Stop the upgrade. */  
    hi_upg_stop();  
    .....  
}  
/* 4. Finish upgrade (hi_upg_finish). */  
hi_upg_finish();
```