Application Note

SimpleLink™ Wi-Fi® CC3x20、CC3x3x 无线更新

摘要

无线 (OTA) 更新是指通过无线方式将新的软件更新或配置交付到嵌入式器件。利用物联网 (IoT) 中的身份密钥 (IRK) 概念，OTA 提供了一种分发固件更新或升级的有效方式。

本文档介绍了德州仪器 (TI)™ 所提供的 SimpleLink™ Wi-Fi® 系列器件的 OTA 库，并说明了如何准备新的云端就绪更新以供 OTA 库下载。

SimpleLink™ MCU 产品系列提供的单一开发环境包含灵活的硬件、软件和工具选项，便于客户开发有线和无线应用。为了能够在主机 MCU、Wi-Fi®、低功耗 *Bluetooth*®、Sub-1GHz 器件等平台中完全重复使用代码，可根据您的设计选择 MCU 或连接标准。只需一次性投资 SimpleLink™ 软件开发套件 (SDK) 便可重复使用，从而开启创建无限应用的大门。如需了解更多相关信息，请访问 www.ti.com.cn/simplelink/cn。

内容

商标

Wi-Fi® is a registered trademark of Wi-Fi Alliance.
Bluetooth® is a registered trademark of Bluetooth SIG Inc..
所有商标均为其各自所有者的财产。

1 引言

SimpleLink™ Wi-Fi® 系列器件的 OTA 库简化了 MCU 应用的工作，既能以安全且具有失效防护的方式访问云端并下载新的升级版本（例如新的固件应用程序、服务包和用户文件），同时保持系统的完整性。

OTA 库公开了一个简单的 API 集：

- OTA_init()
- OTA_set()
- OTA_get()
- OTA_run()

此 API 集与非 OS 平台和基于 OS 的平台兼容。

1.1 OTA 系统方框图

图 1-1 显示了 OTA 系统的方框图。

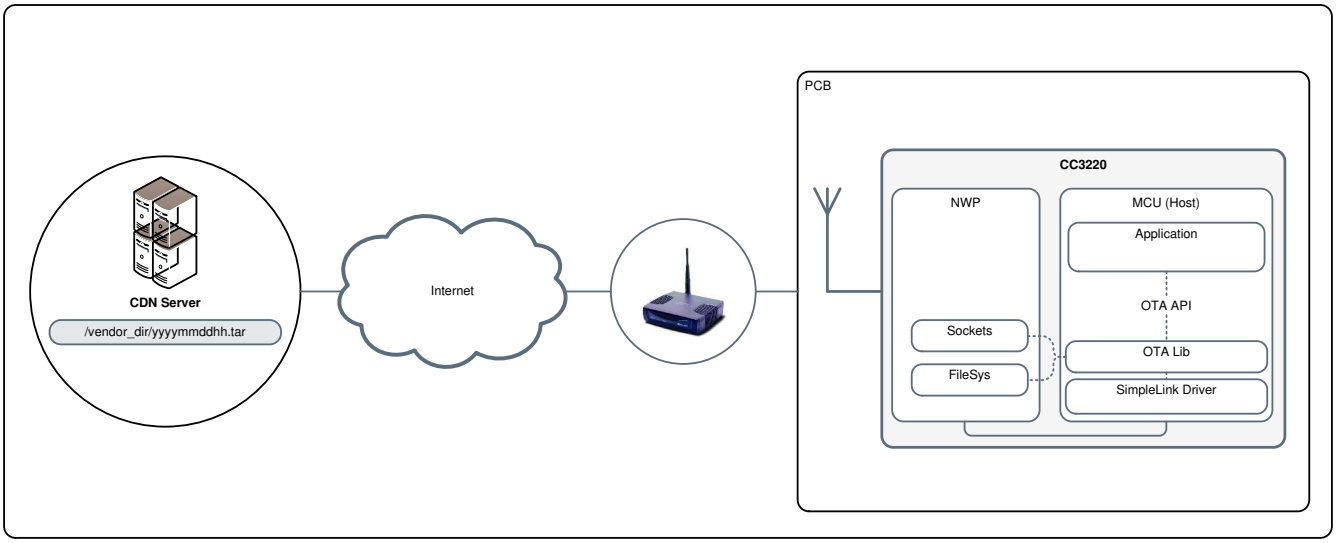

图 1-1. OTA 系统图

OTA 库支持以下 CDN 云供应商：

- Dropbox™
- Github
- 定制（请参阅节 10）

OTA 库会实现一个简单的 HTTP 客户端 (TCP) 来与 CDN 服务器相连。此客户端可以通过主机应用程序进行配置，如下所示：

- 非安全（连接到运行 HTTP 服务器的 CDN）
- 安全（连接到运行 HTTPS 服务器的 CDN）
  - 服务器身份验证（默认需要）
  - 域名验证（默认需要）
  - 无服务器身份验证

软件升级应用程序和用户文件应置于 .tar 存档文件中。默认情况下，所有文件都处于非安全和失效防护模式下。供应商可以在命令文件 (ota.cmd) 中添加条目，以此来更改文件的各种属性（例如安全、签名、证书文件名等等），其中命令文件采用 JSON 格式并包含在 .tar 存档文件中。

1.2 OTA 软件框图

图 1-2 展示了 OTA 软件的框图。

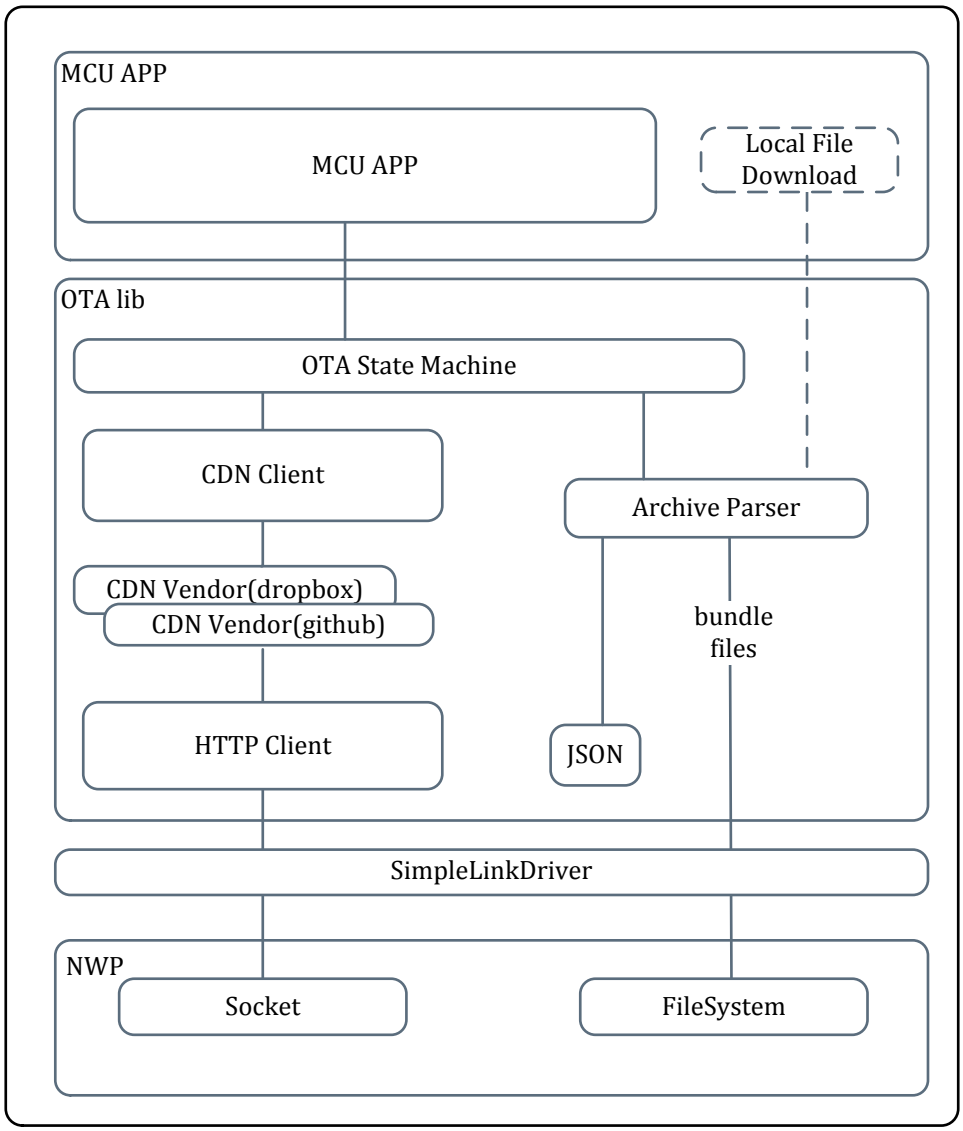

图 1-2. OTA 软件框图

1.3 术语和缩写

表 1-1 列出了本文档中使用的缩略语、缩写词和首字母缩写。

表 1-1. 缩略语、缩写词和首字母缩写

术语	说明
ACM	客户经理
CDN	内容交付网络。在与 OTA 库相关的情形中，CDN 与包含新软件包的服务器有关。
CSP	云存储提供商
HTTP	超文本传输协议
JSON	Javascript 对象标记
OTA	无线
REST	表述性状态转移
URI	统一资源标识符
URL	统一资源定位符

2 添加 OTA 功能的概要说明

通过将 OTA 库和 OTA 示例应用程序作为参考，可以扩展现有主机应用程序以添 OTA 功能。

OTA 示例应用程序会在 CC3220/CC3230/CC3235x 和 CC3120/CC3130/CC3135 器件上作为主机与 MSP432™ 微控制器 (MCU) 搭配使用。

2.1 软件安装与下载

遵循 SDK 入门指南以按如下所示准备开发环境：

1. 下载最新版本的 Code Composer Studio™ (CCS) IDE。
2. 下载最新的 CC32xx SDK 或 CC31xx 插件。
3. 下载最新的 UniFlash 工具，选择相关器件，然后启动内部的 Image Creator 工具。
4. 使用最新的服务包。服务包可以放在 <cc32xx_sdk_install_dir>\tools\cc32xx_tools 中
5. 在手机上安装 SimpleLink™ Starter Pro。

2.2 OTA 过程简介

以下步骤简要介绍了准备和运行 OTA 应用程序所需的步骤。

1. 在选定的 CDN 服务器（Dropbox、Github 或自定义）上创建开发人员帐户。
2. 打开 OTA 示例应用程序并更改选定 CDN 服务器的信息。
3. 对 OTA 示例应用程序进行编译。
4. 使用主机应用程序和所有其他必要的文件来创建 .tar 文件。
5. 将 .tar 文件上传到 CDN 服务器。
6. 打开 UART 终端以查看应用程序日志。
7. 使用 Image Creator，准备包含 OTA 应用程序的映像并对器件进行编程。
8. 当主机应用程序启动 OTA 过程时，下载并运行新的主机应用程序。

备注

将 CDN 服务器中的 .tar 文件替换为较新文件时，应该会重新调用 OTA 下载。

3 运行默认 OTA 示例应用程序

OTA 示例应用程序提供了具有云 OTA 和配置功能的基本应用程序。此应用程序可以用作开发目标应用程序的基础。有关此应用程序的详细信息，请参阅节 5。

云 OTA 示例应用程序支持：

- 操作系统：FreeRTOS 或 TI-RTOS
- CC32xx 器件型号：CC3220R、CC3220S、CC3220SF、CC3230S、CC3230SF、CC3235S、CC3235SF
- 开发工具：CCS 和 IAR Workbench。请参阅 SDK 发行说明，以获取最新受支持的版本

以下各个部分适用于特定的 CC32xx 型号和操作系统；请从 SDK 导入选定器件型号和操作系统对应的相关应用程序工程。对于 CC3220R 器件，您可以使用 CC3220S 器件工程。

备注

OTA 库会生成编译错误。在 Dropbox 或 Github 上开立开发人员帐户，并获取令牌以在 <cc32xx_sdk_install_dir>\source\ti\net\ota\otauser.h 中使用。对于 Dropbox 云服务，请参阅 节 8.1 和 节 4.1。

3.1 为 CC3220SF 器件编译 OTA CCS 云工程

安装 CC32xx SDK 并导入以下 CCS 工程：

- OTA 库工程
- OTA 应用工程

图 3-1 展示了“Project Explorer”选项卡，其中选中了 OTA 云工程。

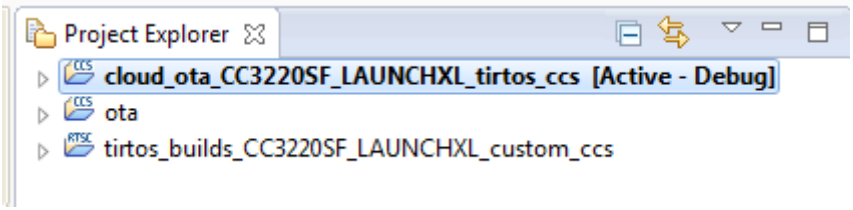

图 3-1. 选中的 OTA 云工程名称

1. 使用 CCS IDE 从以下位置导入 OTA 库工程：

   <cc32xx_sdk_install_dir>\source\ti\net\ota

   备注

   不应将此工程复制到 wCCS 工作区。

2. 根据连接的 CC32xx 器件和所需的操作系统（TI RTOS 或 FreeRTOS），使用 CCS IDE 从以下位置导入 OTA 云工程：
3. 配置云服务器。
   a. 在 Dropbox 或 Github 上开设开发人员帐户。如需相关说明，请参阅 节 8.1 和 节 4.1。
   b. 将提供的云服务器令牌复制到 OTA 库用户配置文件中：

      <cc32xx_sdk_install_dir>\source\ti\net\ota\otauser.h in

```
#define OTA_VENDOR_TOKEN    "<User defined token>"
```

4. 配置 OTA 特殊编译标志。
   - OTA lib – 在 otauser.h 中定义 SL_ENABLE_OTA_DEBUG_TRACES 以打印详细的 OTA 库调试日志
   - OTA project – 在 cloud_ota.c 中定义 DISABLE_OTA_SWITCH_TRIGGER，以在五次 Ping 后启动 OTA 过程，而不等待外部触发（CC32xxLP 按钮）。定义 OTA_LOOP_TESTING，以在返回值为 NO_UPDATE, OLDER_VERSION 时继续运行 OTA 尝试。
5. 完成其他 CCS 配置。
6. 编译工程并生成要由 UniFlash 进行编程的 cloud_ota.bin 文件。
   a. 编译 OTA 库工程 – ota。
   b. 编译 OTA 示例应用工程 – cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs。
   c. 在工程工作区目录 <cc32xx_sdk_workspace_dir>\cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs\Debug\cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs.bin 中输出 .bin 文件。

3.2 通过 UniFlash 工具使用其中的 Image Creator 进行编程

以下说明介绍了如何对器件进行编程。

1. 打开 UniFlash 并选择相关器件，如图 3-2 所示。

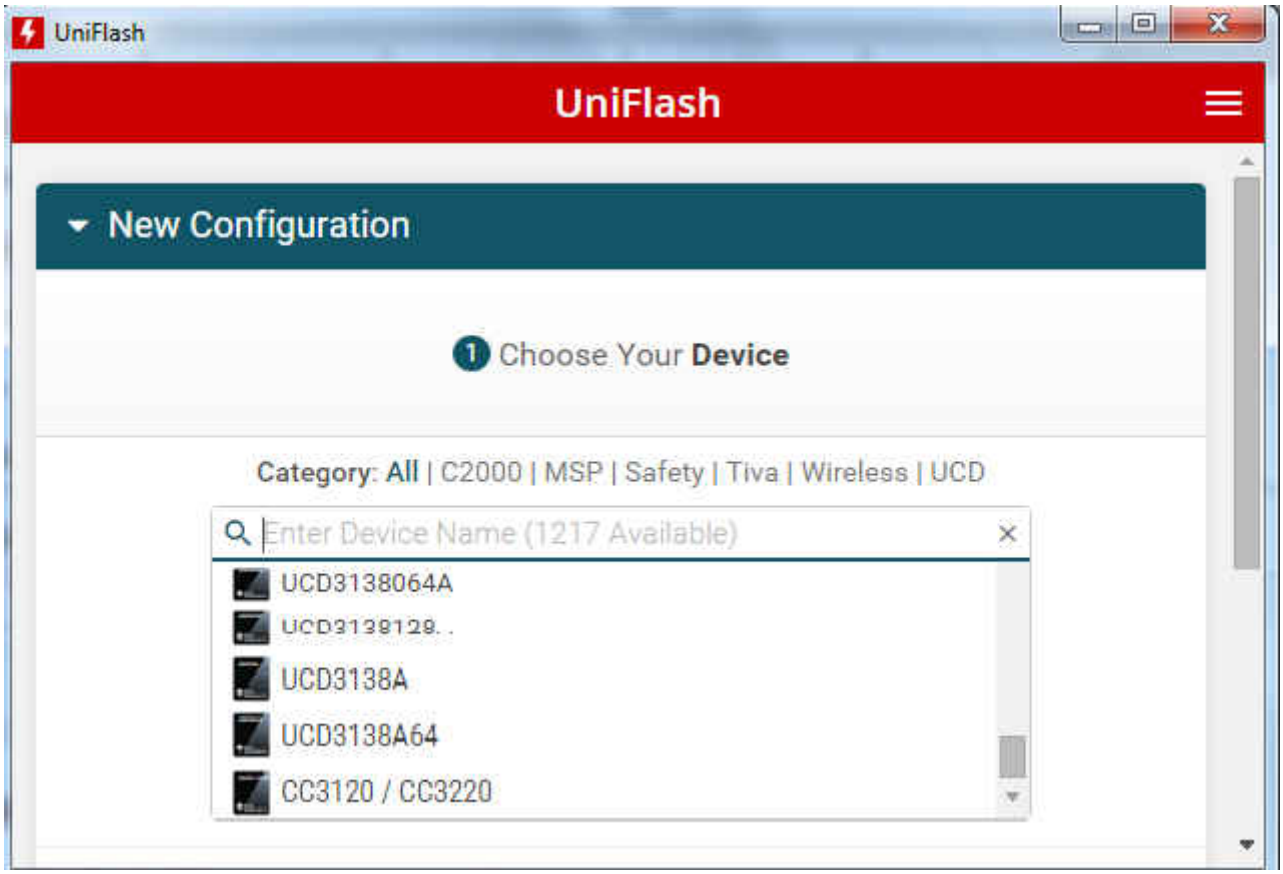

图 3-2. 选择您的器件

2. 启动 Image Creator 并选择“New Project”，填写工程名称，然后点击“create project”（对于调试模式，必须先在开发模式中打开器件）。
3. 进入“Files”→“Trusted Root-Certificate Catalog”并选择 <cc32xx_sdk_install_dir>\tools\certificate-playground\certcatalogPlayGroung20160911.lst（请参阅图 3-3，不要使用默认设置）。

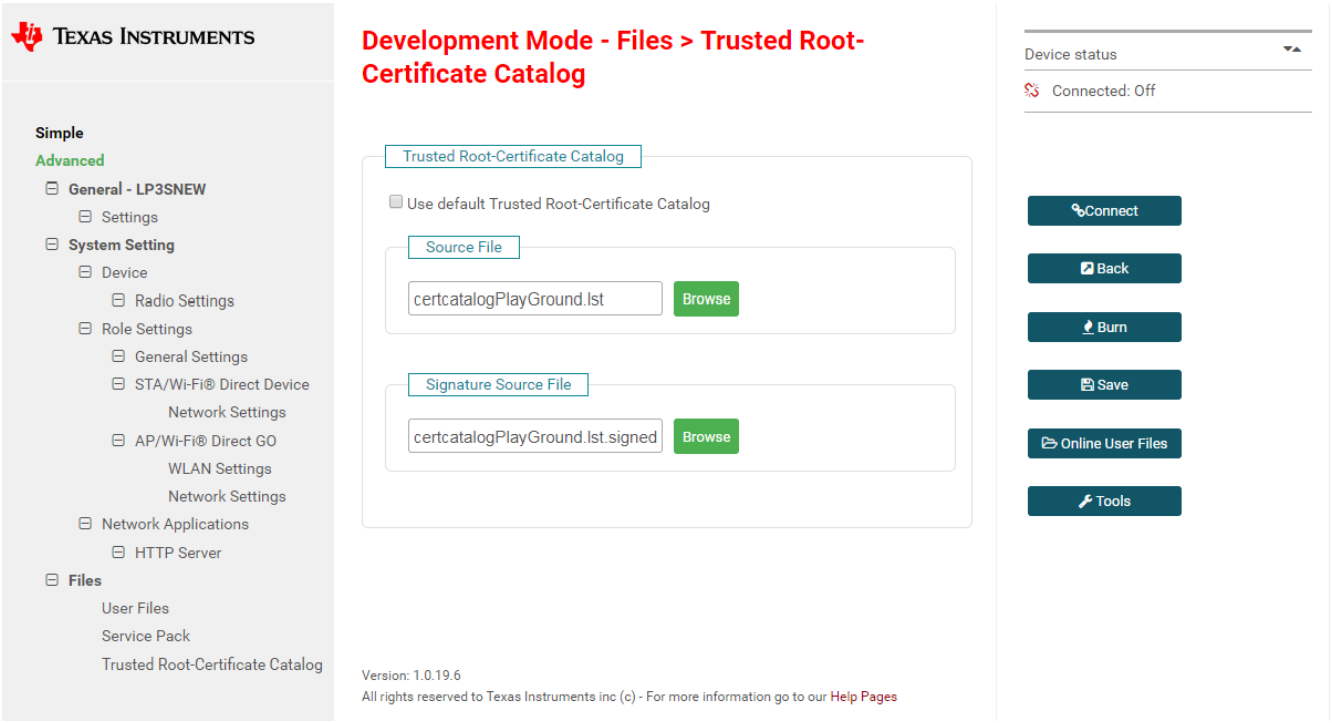

图 3-3. 受信任的根证书目录

4. 进入“Files”→“Service Pack”，并选择最新的联网子系统服务包（请参阅图 3-4）。

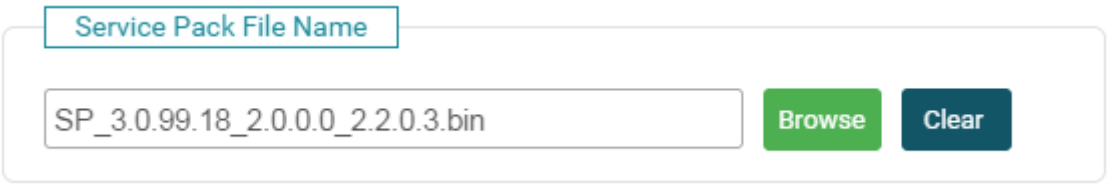

图 3-4. 服务包文件名

5. 进入“Files”→“User Files”，然后选择以下几项（请参阅图 3-5）。

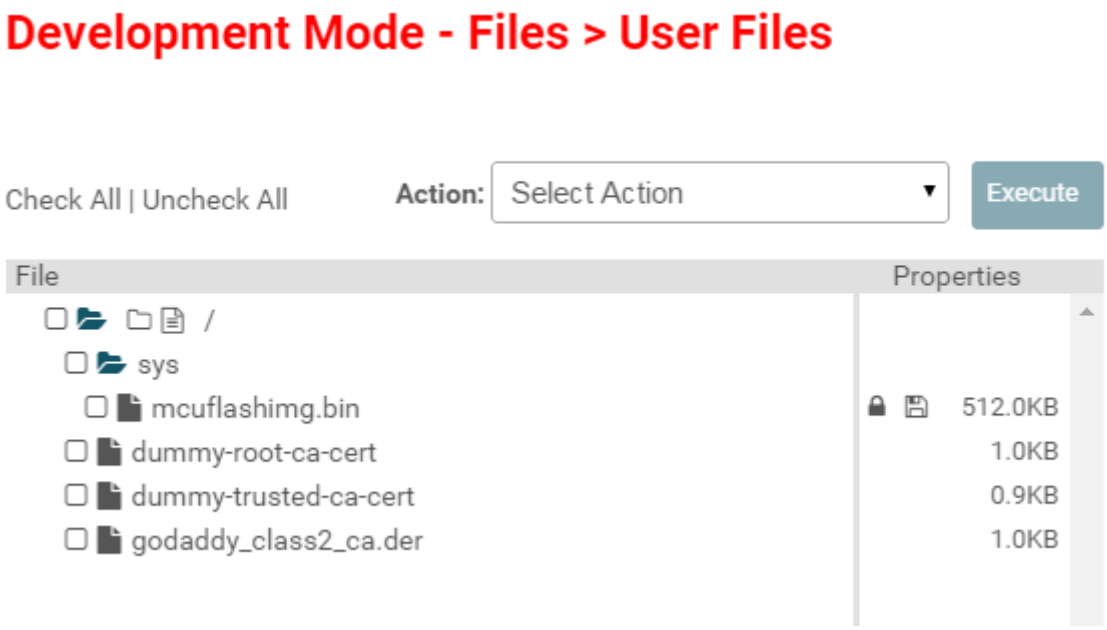

图 3-5. 用户文件

- 证书文件：
  - <cc32xx_sdk_install_dir>\tools\certificate-playground\dummy-root-ca-cert
  - <cc32xx_sdk_install_dir>\tools\certificate-playground\dummy-trusted-ca-cert
- Dropbox 和 Github 的证书：
  - 下载 Dropbox 和 Github 根 CA 证书（即“DigCert_High_Assurance_CA.der”和“GoDaddy_class2_CA.der”）。
- 用户文件 MCU 映像（请参阅图 3-6）：
  - <cc32xx_sdk_workspace_dir>\cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs\Debug\cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs.bin

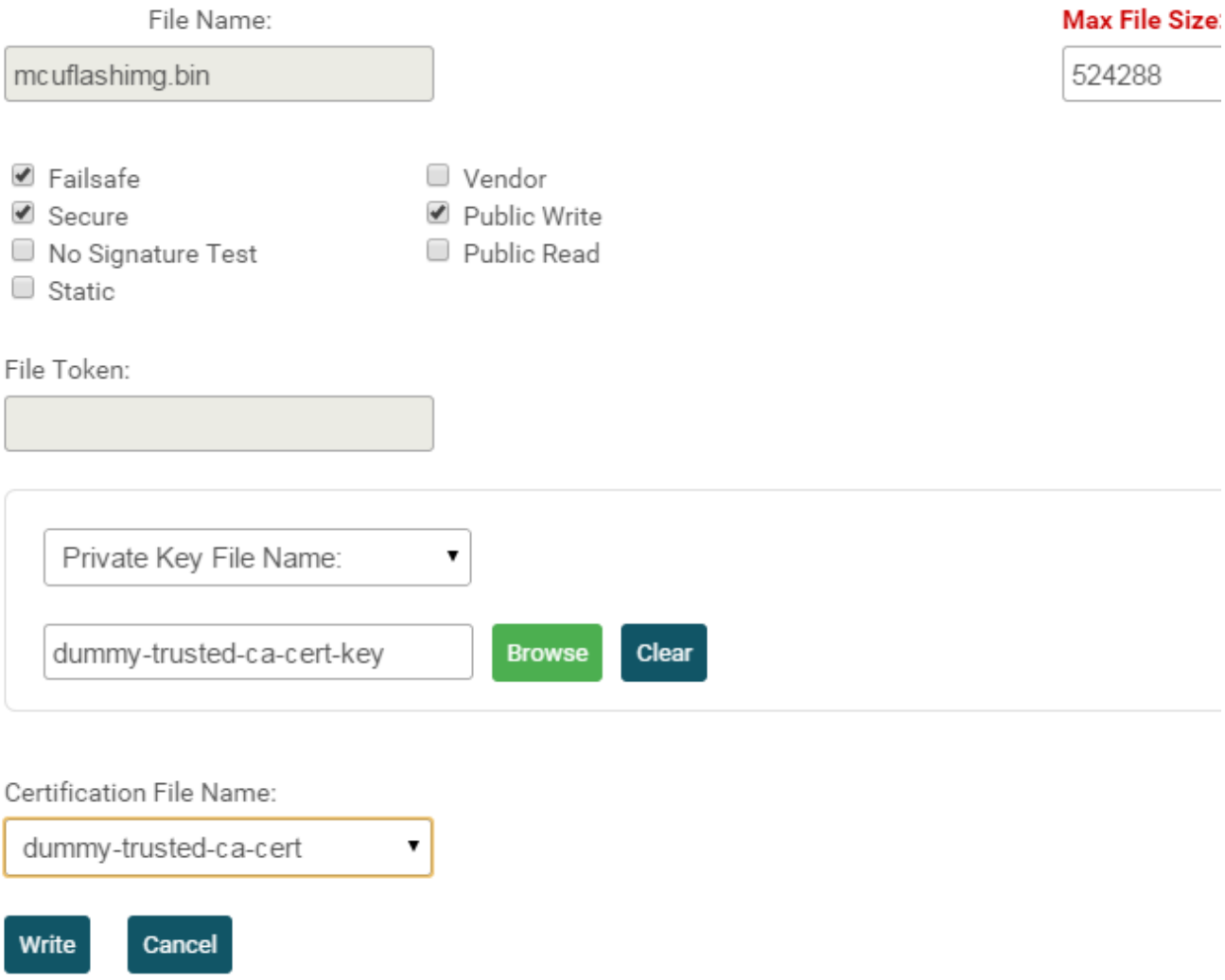

图 3-6. MCU 映像

6. 在“System setting”→“Role Settings”→“General Settings”下，将器件配置为 STA 角色。
7. 点击“Create OTA”以生成要放到云服务器上的 .tar 文件。输出文件 2016091917_CC3220SF_CC3220SF_OTA.tar 应放到云目录中（请参阅图 3-7）。

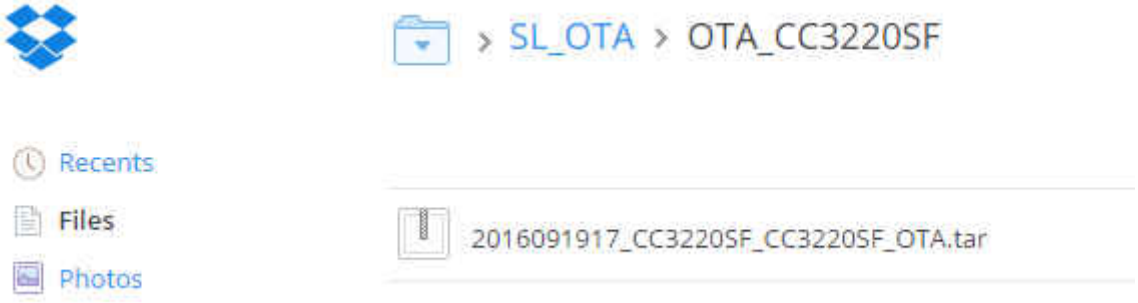

图 3-7. 云目录

8. 在 otauser.h 中进行配置：

```
#define OTA_VENDOR_DIR      "OTA_VENDOR_DIR"
```

9. 点击“Program Image (Create & Program)”，如图 3-8 所示。现已将映像编程到器件。

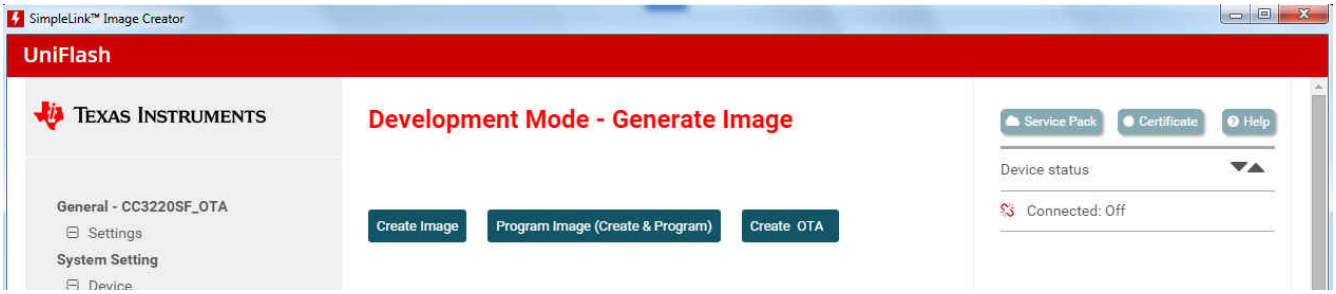

图 3-8. 生成映像

3.3 运行示例应用程序

对 OTA 示例应用程序进行编程并将 OTA tar 文件上传到云端后，点击“Reset”以启动 OTA 过程。示例应用程序执行的主要步骤如下所示：

1. 通过现有的配置文件或通过配置流程，连接到 AP。
2. 发送 ping 并等待点击 CC3220LP 按钮来启动 OTA。
3. 使用 CDN 服务器详情配置 OTA 库。
4. 运行从外部触发器初始化的 OTA 过程（CC3220LP 板上的按钮）
   a. 连接到 CDN HTTPS 服务器。
   b. 请求目录内容，并下载 .tar 存档文件。
   c. 将 .tar 存档文件（包括 mcuflashimg.bin）提取到串行闪存。
5. 下载完成后，设置 IMAGE_TESTING 模式并复位 MCU。
6. 系统测试成功完成后，设置 IMAGE_COMMIT 并继续运行新的 MCU 映像。

备注

用户可以再次初始化新的 OTA 过程。只有在新的 .tar 文件上传到 CDN 后，或者用户通过 cloud_ota.c 文件中的 #define OTA_LOOP_TESTING 将其配置为忽略较新版本时，才会重新调用下载过程。

3.4 为 CC3220SF 编译云 OTA IAR 工程

安装 IAR Workbench 后，遵循快速入门指南（位于 <cc3220_sdk_install_dir>/docs/simplelink_mcu_sdk 中的“QuickStart for IAR IDE”部分），然后完成以下步骤：

1. 配置自定义参数变量：（“Tools”→“Configure Custom Argument Variables”，在“Global”选项卡中，选择“Import”。导航至：SDK_installation_Dir → examples → CC32XX_SDK.custom_argvars，然后打开）
2. 导入工程工作区 cloud_ota.eww
3. OTA 库中没有 IAR 工程。若要编译 IAR 工程，请执行以下操作：
   a. 按照节 4.1 中的说明，编辑 otauser.h 文件。
   b. cd <cc32xx_sdk_install_dir>/source/ti/net/ota
   c. 运行 c:\TI\xdctools_xx_core\gmake.exe
   d. 在 IAR 环境中重新编译 cloud_ota 工程以链接新的 ota.a 库。

4. IAR 调试器：
   - “Project”→“General Options”→“Target”→“Device”，选择“Texas Instruments CC3220SF”
   - “Project”→“Debugger”→“Setup”→“Driver TI XDS”
   - “Project”→“Debugger”→“Download”→“Use flash loader”

4 将 OTA 功能添加到嵌入式软件应用程序中

本应用手册重点介绍了 CC32xx 器件通过支持互联网的 Wi-Fi® 接口接收固件更新及任何相关文件的能力。

4.1 更新 OTA 定义

OTA 库支持以下 CDN 供应商：

- Dropbox 进行安全的无线 (OTA) 传输
- Github

本节介绍了如何正确使用其中一个 CDN 供应商。若要添加对其他 CDN 供应商的支持，请参阅节 10。

若要运行 OTA 库，供应商必须在其中一个受支持的 CDN 开设帐户以分发软件更新。有关开设帐户的更多信息，请参阅节 8。

包括帐户信息在内的 CDN 设置以及选定的目录和令牌会更新到 otauser.h 文件中。

4.1.1 otauser.h

此文件位于以下位置：<cc32xx_sdk_install_dir>/source/ti/net/ota/otauser.h

通过定义 OTA_SERVER_TYPE，选择 OTA 服务器供应商：

```
#define OTA_SERVER_TYPE    OTA_SERVER_DROPBOX_V2
```

定义供应商服务器信息部分：

```
#define OTA_VENDOR_DIR  "OTA_CC3220SF"
/* Custom server info */#define OTA_SERVER_NAME                "api.dropboxapi.com"
#define OTA_SERVER_IP_ADDRESS          0x00000000
#define OTA_SERVER_SECURED             1
/* Custom vendor info */#define OTA_VENDOR_TOKEN               "<User defined Dropbox token>"
#define OTA_SERVER_ROOT_CA_CERT        "DigCert_High_Assurance_CA.der"
#define OTA_SERVER_AUTH_IGNORE_DATA_TIME_ERROR
#define OTA_SERVER_AUTH_DISABLE_CERT_STORE
```

主机应用程序可以在 OTA_SERVER_IP_ADDRESS 中设置服务器 IP 地址，而不是服务器名称。

默认情况下，lib 要求进行服务器身份验证和域名验证。从安全角度来看，之前的要求是基本要求，并且 TI 建议使用此模式。如果定制服务器未受安全保护，或者不要求进行服务器身份验证和域名验证，则 OTA_SERVER_ROOT_CA_CERT 可以为未定义。

4.2 在主要应用程序中使用 OTA 库

图 4-1 展示了运行 OTA 的主机应用程序可能的基本流程。

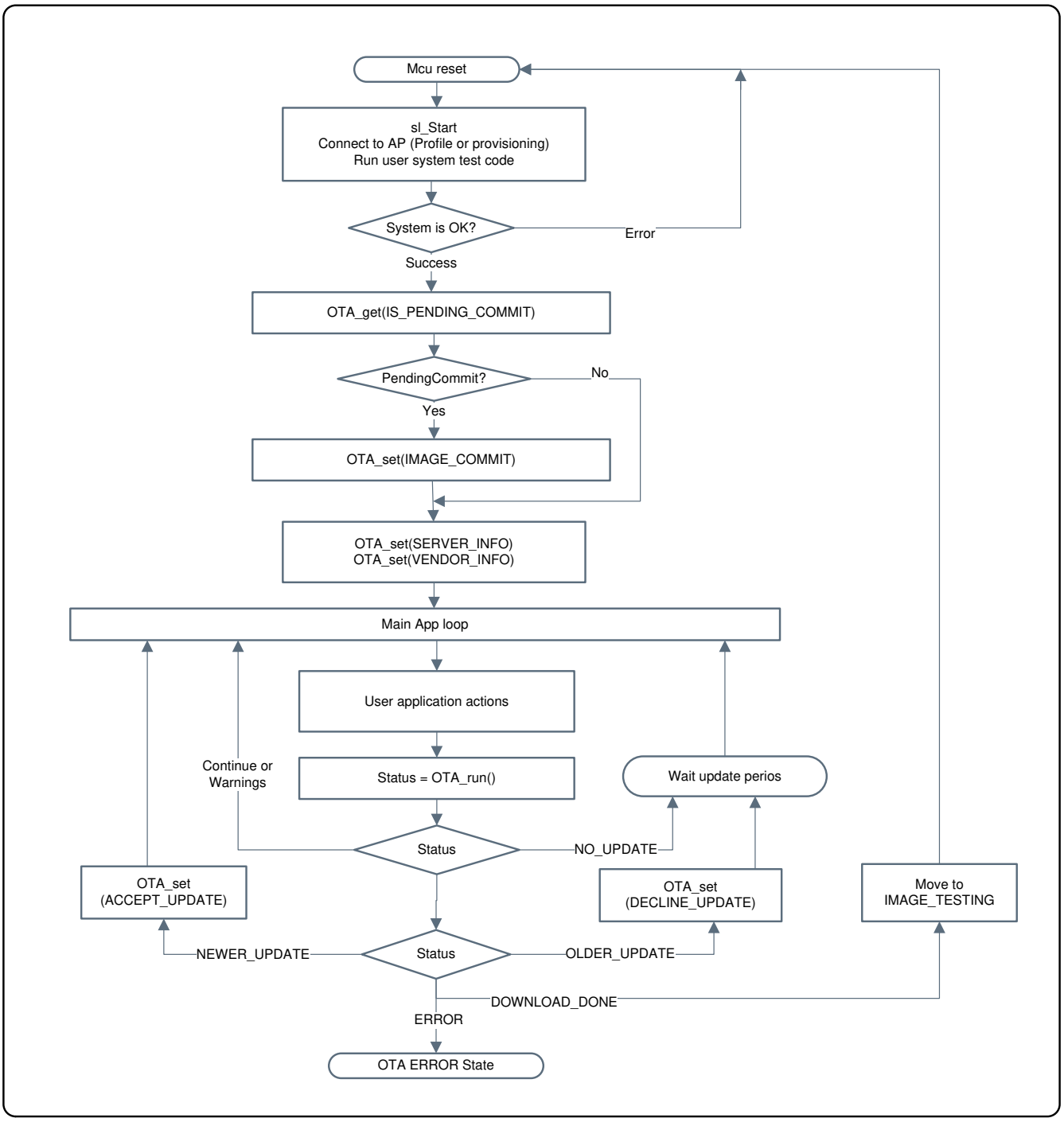

图 4-1. 基本 OTA 应用程序流程图

4.2.1 初始化 OTA 库

主机应用程序必须定义一个全局缓冲区，以便使用 OTA 库，并在初始化 OTA 时提供此缓冲区：

```
#include "ota.h"
#include "otauser.h"
OTA_memBlock otaMemBlock;
_i16 Status;
Status = OTA_init( OTA_RUN_NON_BLOCKING, &taMemBlock, NULL);
if (Status < 0)
{
    /* handle error */
}
```

设置 OTA 服务器信息：

```
OtaOptServerInfo_t g_otaOptServerInfo;
_i16 Status;
g_otaOptServerInfo.IpAddress         = OTA_SERVER_IP_ADDRESS;
g_otaOptServerInfo.SecuredConnection = OTA_SERVER_SECURED;
strcpy((char *)g_otaOptServerInfo.ServerName,  OTA_SERVER_NAME);
strcpy((char *)g_otaOptServerInfo.VendorToken, OTA_VENDOR_TOKEN);
Status = OTA_set(EXTLIB_OTA_SET_OPT_SERVER_INFO,
                 sizeof(g_otaOptServerInfo), (_u8 *)&g_otaOptServerInfo);
if (Status < 0)
{
    /* handle error */
}
```

设置 OTA 供应商 ID：

```
_i16 Status;
Status = OTA_set (EXTLIB_OTA_SET_OPT_VENDOR_ID,
                          strlen(OTA_VENDOR_DIR), (_u8 *)OTA_VENDOR_DIR);
if (Status < 0)
{
    /* handle error */
}
```

4.2.2 运行 OTA 过程

OTA 过程会按步骤与主应用程序并排运行。只要返回值为 OTA_RUN_STATUS_CONTINUE，用户就应继续调用 OTA 过程。

以下示例显示了非操作系统应用程序的简单主循环：

```
_i16 Status;
While (MainStatus != END_OF_MAIN_APP)
{
    /* Run main application step */
    MainStatus = RunMainAppStep();
    if (MainStatus < 0)
    {
        /* handle error */
    }
    /* Run OTA process step */
    Status = OTA_run ();
    /* Handle OTA process step status */
}
```

可能的 OTA 过程返回值如下：

- OTA_RUN_STATUS_CONTINUE – 继续调用 OTA_run。
- RUN_STAT_DOWNLOAD_DONE – 当前 OTA 更新会话已完成。主机应复位 MCU 并测试映像。在 MCU 初始化时，检查系统是否已连接，然后测试捆绑包是否处于待提交状态，并设置 commit 命令。当 MCU 初始化出错时，回滚新的映像捆绑包。
- OTA_RUN_STATUS_NO_UPDATES – 没有映像可供下载；在下个 OTA 期间或收到外部触发条件时重试。
- OTA_RUN_STATUS_CHECK_NEW_VERSION – OTA 找到了较新的更新版本，用户可以通过使用 _ACCEPT_UPDATE 选项调用 OTA_set 来接受新版本，或继续运行，或在继续操作之前再次进行检查。
- OTA_RUN_STATUS_CHECK_OLDEST_VERSION – OTA 找到了较旧的更新版本，用户可以通过使用 _DECLINE_UPDATE 选项调用 OTA_set 来停止 OTA，或继续运行，或在继续操作之前再次进行检查。
- OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_ <reason> – 一组 OTA WARNINGS；出现错误，但 OTA 将重试该过程五次，因此请继续运行 OTA。
- OTA_RUN_ERROR_CONSECUTIVE_OTA_ERRORS – OTA 过程连续失败五次；复位 MCU 以重试，或选择停止 OTA。
- OTA_RUN_ERROR_<reason> – (负值) 一组 OTA 错误；停止 OTA 过程。
- OTA_RUN_ERROR_SECURITY_ALERT – 来自文件系统的安全警告；停止下载当前的 OTA 更新。导致此错误的文件系统错误如下：
  - SL_ERROR_FS_CERT_IN_THE_CHAIN_REVOKED_SECURITY_ALERT
  - SL_ERROR_FS_WRONG_SIGNATURE_SECURITY_ALERT
  - SL_ERROR_FS_CERT_CHAIN_ERROR_SECURITY_ALERT
  - SL_ERROR_FS_SECURITY_ALERT

```
_i32 Status;
OtaOptVersionsInfo_t VersionsInfo;
_i32 Optionlen;
    Status = OTA_run();
    switch (Status)
    {
        case OTA_RUN_STATUS_CONTINUE:
            SignalEvent(APP_EVENT_CONTINUE);
            break;
        case OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_CONNECT_OTA_SERVER:
        case OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_RECV_APPEND:
        case OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_REQ_OTA_DIR:
        case OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_REQ_FILE_URL:
        case OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_CONNECT_FILE_SERVER:
        case OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_REQ_FILE_CONTENT:
        case OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_FILE_HDR:
        case OTA_RUN_STATUS_CONTINUE_WARNING_FAILED_DOWNLOAD_AND_SAVE:
            /* on warning, continue calling OTA_run for next retry */
            SignalEvent(APP_EVENT_CONTINUE);
            break;
        case OTA_RUN_STATUS_NO_UPDATES:
            /* OTA will go back to IDLE and next OTA_run will restart the process */
            SignalEvent(APP_EVENT_OTA_CHECK_DONE);
            break;
        case OTA_RUN_STATUS_CHECK_NEWER_VERSION:
            OTA_set(EXTLIB_OTA_SET_OPT_ACCEPT_UPDATE, 0, NULL);
            SignalEvent(APP_EVENT_CONTINUE);
            break;
        case OTA_RUN_STATUS_CHECK_OLDER_VERSION:
            SignalEvent(APP_EVENT_OTA_CHECK_DONE);
            break;
        case OTA_RUN_STATUS_CHECK_OLDER_VERSION:
            SignalEvent(APP_EVENT_OTA_DOWNLOAD_DONE);
            break;
        case OTA_RUN_ERROR_CONSECUTIVE_OTA_ERRORS:  /* 5 consecutive failures, must stop */
            SignalEvent(APP_EVENT_RESTART);
            break;
        case OTA_RUN_ERROR_NO_SERVER_NO_VENDOR:
        case OTA_RUN_ERROR_UNEXPECTED_STATE:
        case OTA_RUN_ERROR_SECURITY_ALERT:          /* security alert, must stop */
            SignalEvent(APP_EVENT_OTA_ERROR);
            break;
        default:
            break;
```

4.2.3 OTA 提交流程

成功完成 OTA 进程（状态 RUN_STAT_DOWNLOAD_DONE）后，所有软件升级包文件都会以捆绑模式存储在串行闪存上，但仍未被使用。

主机应用程序应通过以下方式将该捆绑包置于 IMAGE_TESTING 模式：调用 sl_Stop() 并将超时设置为 0 以外的值，然后复位 MCU。

```
sl_Stop(200);
    Platform_Reset();
```

复位后，会使用捆绑包文件，同时应用程序应检查新映像是否能正常工作（例如，连接到网络，运行某些网络测试等等）。

系统测试成功完成时，应用程序应检查该映像是否处于 WAIT_FOR_COMMIT 模式，并调用 OTA 库来执行提交。提交后，相关文件会被使用，并且不允许进行回滚。

在系统测试失败时，应用程序必须将 MCU 复位以使所有文件回滚到上个映像：

```
_i32 isPendingCommit;
    _i32 isPendingCommit_len;
    _i32 Status;
    Status = OTA_get(EXTLIB_OTA_GET_OPT_IS_PENDING_COMMIT,

&isPendingCommit_len, (_u8 *)&isPendingCommit);
    if (Status < 0)
    {
        /* handle error */
    }
    if (isPendingCommit)
    {
        Status = OTA_set(EXTLIB_OTA_SET_OPT_IMAGE_COMMIT, 0,
                         NULL);
        if (Status < 0)
        {
            /* handle error */
        }
    }
    Return 0;
```

5 OTA 示例应用程序

该 OTA 示例应用程序由状态机实现。该状态机在收到 sl_Start 后启动，并由 NWP 事件和应用程序事件驱动，如表 5-1 和表 5-2 所示。

表 5-1. OTA 应用状态

OTA 应用状态	说明
APP_STATE_STARTING	等待从 NWP 收到 INIT_COMPLETE 事件（调用 sl_Start 后）。
APP_STATE_WAIT_FOR_CONNECTION	等待从 NWP 收到 WLAN_CONNECTED 事件 - 如果器件上保存了配置文件，则连接运行。
APP_STATE_WAIT_FOR_IP	等待从 NWP 收到 IPV4_ACQUIRED 事件
APP_STATE_PROVISIONING_IN_PROGRESS	等待 PROVISIONING_SUCCESS 事件。如果没有配置文件，或连接失败，则配置流程启动。
APP_STATE_PROVISIONING_WAIT_COMPLETE	等待 PROVISIONING_STOP 事件，配置停止表示配置连接的整个循环结束。
APP_STATE_PINGING_GW	这是主应用程序状态。在此状态下，应用程序会向网关发送 Ping 请求。 等待从 NWP 收到 PING_COMPLETE 事件并初始化另一个 ping 请求 (sl_NetAppPing)。
APP_STATE_OTA_RUN	继续调用 OtaRunStep 并检查返回状态。 下载完成后，复位 MCU 以测试新的映像。
APP_STATE_ERROR	致命错误状态 – 停止

表 5-2. NWP 事件

NWP 事件	转换为 OTA APP 事件
GeneralEvent	*SL_ERROR_LOADING_CERTIFICATE_STORE – Ignored(*) 否则：SignalEvent(APP_EVENT_ERROR)
WlanEvent	SL_WLAN_EVENT_CONNECT：SignalEvent(APP_EVENT_CONNECTED) SL_WLAN_EVENT_DISCONNECT：SignalEvent(APP_EVENT_DISCONNECTED) 否则：SignalEvent(APP_EVENT_ERROR) SL_WLAN_EVENT_PROVISIONING_STATUS： 根据状态： • SignalEvent(APP_EVENT_PROVISIONING_STARTED) • SignalEvent(APP_EVENT_PROVISIONING_SUCCESS) • SignalEvent(APP_EVENT_PROVISIONING_STOPPED) • SignalEvent(APP_EVENT_ERROR)
FatalErrorEvent	SignalEvent(APP_EVENT_ERROR)
NetAppEvent	SL_NETAPP_EVENT_IPV4_ACQUIRED：SignalEvent(APP_EVENT_IP_ACQUIRED) *SL_NETAPP_EVENT_IPV4_LOST：SignalEvent(APP_EVENT_DISCONNECT) *SL_NETAPP_EVENT_DHCP_IPV4_ACQUIRE_TIMEOUT：SignalEvent(APP_EVENT_DISCONNECT) 否则：SignalEvent(APP_EVENT_ERROR)
HttpServerEvent	SignalEvent(APP_EVENT_ERROR)
SockEvent	SL_SOCKET_TX_FAILED_EVENT：SignalEvent(APP_EVENT_RESTART) 否则：SignalEvent(APP_EVENT_ERROR)

5.1 应用状态机

图 5-1 展示了应用状态机示例。

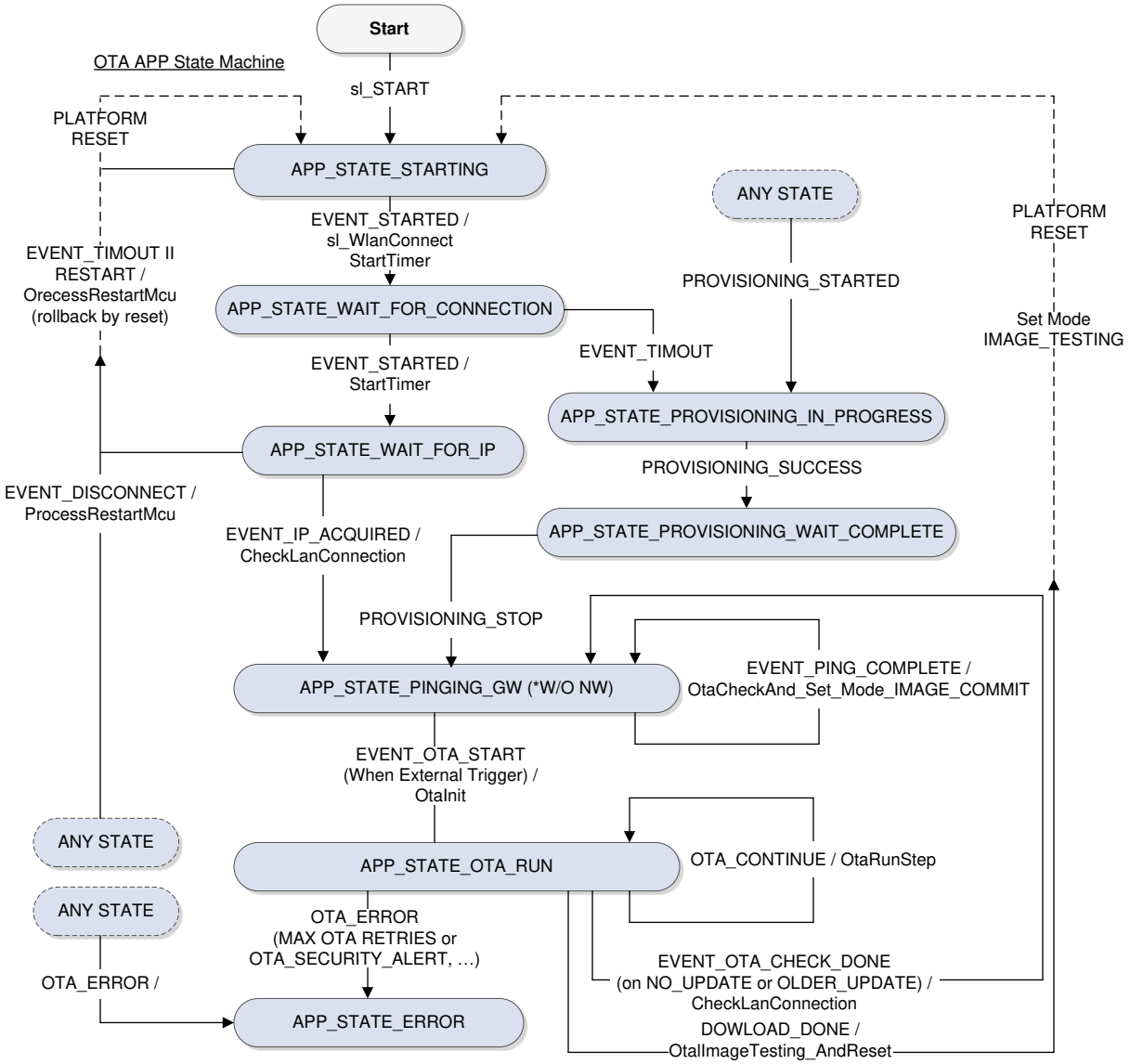

图 5-1. 应用状态机示例

5.2 配置连接

主机应用程序使用现有的配置文件来与 AP 相连。如果没有配置文件，或者主机应用程序无法与现有的配置文件相连，则主机应用程序可以选择通过 TI 配置流程添加新的配置文件。

如果没有连接，配置流程会自动开始。用户应使用 Mobile Provisioning 应用程序来添加新的配置文件。

更多关于配置的信息可以在“配置应用程序报告”中找到。

更多关于该移动应用程序的信息可以在“配置用户指南”中找到。

5.3 OTA 外部触发器

建立连接后，主机应用程序会一直检查与 AP 的连接 (PING)。

若要启动 OTA 过程，请初始化外部触发器。外部触发器会根据使用的不同平台来定义。

- CC32xx – 开关 2
- MSP432 – 开关 P1.1

外部触发器初始化后，主机应用程序会启动 OTA 过程。

6 创建 OTA 软件升级包

软件升级包是一个包含所有相关文件的 .tar 存档文件（未压缩）。文件应当按照图 6-1 所示以目录层次结构的形式放置。

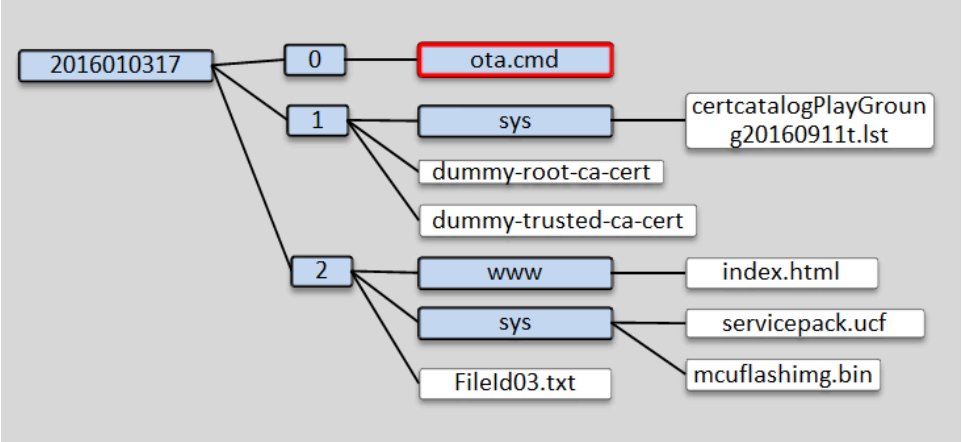

图 6-1. 目录层次结构

根目录的名称应当包含软件包版本号。此名称应该包含可与现有软件包版本号相提并论的数位，并且用户可以根据该名称来确定是否需要升级。图 6-1 展示了以下格式的日期和时间数字：YYYYMMDDHH。

目录 0、1 和 2 用于设置软件包文件下载顺序：

- 目录 0 – 包含名为 ota.cmd 的元数据软件包文件；安全软件包中的每个文件都必须在元数据文件中存在一个对应条目，以及可选的签名和证书文件名。以默认属性打开的文件（非安全、失效防护、捆绑模式）不应在元数据文件中存在条目。
- 目录 1 – 包含必须下载到对应文件的证书。
- 目录 2 – 使用目录 0 中元数据、目录 1 中证书的文件或其他文件。

文件名和目录名称必须与串行闪存中的名称相同。对于根目录中的文件，应移除前导斜杠（证书文件名限制）。

基本 .tar 文件应当使用 UniFlash 工具并点击“Create OTA”来生成，如图 6-2 所示。此基本 .tar 文件仅具有 MCU 映像和服务包。

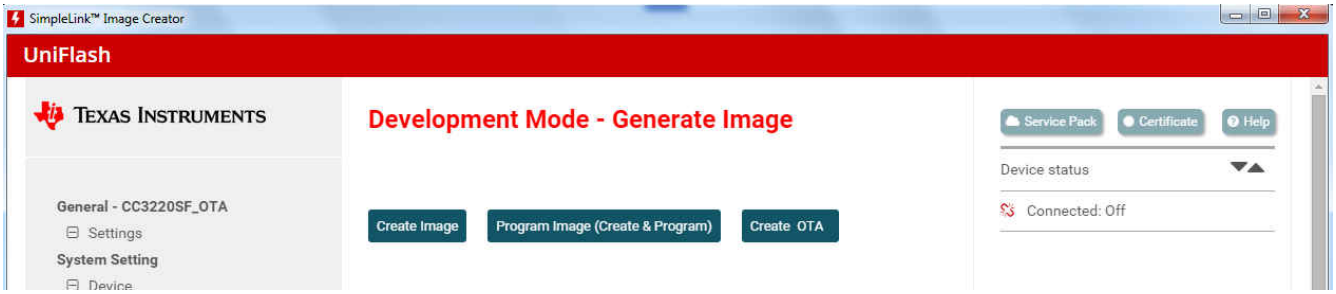

图 6-2. 创建 OTA

用户可以通过以下方法在现有 .tar 文件中添加用户文件：将文件解压到本地目录，添加用户文件，然后将这个目录归档为 .tar 文件。

限制：

- 仅支持 .tar 文件类型：5-目录；0-文件
- 文件名包含完整路径（含子目录），并且上限为 100 个字节。
- 证书存储 (certstore.lst)：MaxRequestSize = 7000，安全、失效防护、公共写入
- 服务包：对于 ROM（补丁）：MaxRequestSize = 131072，失效防护
- 服务包：对于 TDFLASH：MaxRequestSize = 262144，无失效防护、安全，公共写入

7 准备 ota.cmd 元数据文件

ota.cmd 文件是 OTA 存档文件中的第一个文件，并采用 JSON 格式。每个 JSON 对象均代表存档文件中的一个文件（未被使用）和一个具有默认属性的文件（失效防护、非安全和捆绑模式）。具有默认属性的存档文件不需要在 ota.cmd 元数据文件中存在 JSON 对象。对于非捆绑文件（例如没有镜像空间的大型文件），则无法保证系统完整性。

该元数据支持以下 JSON 令牌：

- filename – 文件的名称，.tar 文件和目标 SFLASH 中具有相同的名称
- signature_base64 – Base64 格式的文件签名，大小为 345 个字节
- certificate – 证书文件名
- secured – 如果文件受到保护，则为 1（默认为非安全）
- maxsize – 最大文件大小，仅在新建文件时才相关（默认为实际文件大小）
- bundle – 如果文件是捆绑包的一部分，则为 1（默认为捆绑 1）

以下代码示例包含一个带有签名和证书的安全文件 FileId03.txt 的对象。第二个对象是一个带有签名的服务包文件。

```
[
    {
        "filename": "/local/FileId03.txt",
        "signature_base64": "kc8XfFOfMfr4HBJiPxTRHyb99d2uOoICme0AYU94+...",
        "certificate": "dummy-trusted-ca-certcert",
        "secured": 1,
        "bundle": 0
    },
    {
        "filename": "/sys/servicepack.ucf"
        "signature_base64": "EEC6GZG1Oq6Agigmb2f9ny9rNK2Mg9hFClpgMhd4jCW/...",
        "certificate":"",
        "secured": 1,
        "bundle": 1
    },
    {
        "filename": "/sys/mcuflashimg.bin",
        "signature_base64": "dRTARlzLFKAog34ZUareCmo9j2lrHnvc+v3qqW9C/...",
        "certificate": "dummy-root-ca-certcert",
        "secured": 1,
        "bundle": 1
    }
]
```

8 通过云服务分发软件升级

8.1 Dropbox™ 云交付网络 (CDN) 入门

以下步骤介绍了如何使用 Dropbox 服务器来运行云 OTA 应用程序。在按照后续说明操作之前，必须先创建 Dropbox 帐户。

1. 在主页上，打开 Developers 选项卡，如图 8-1 所示。

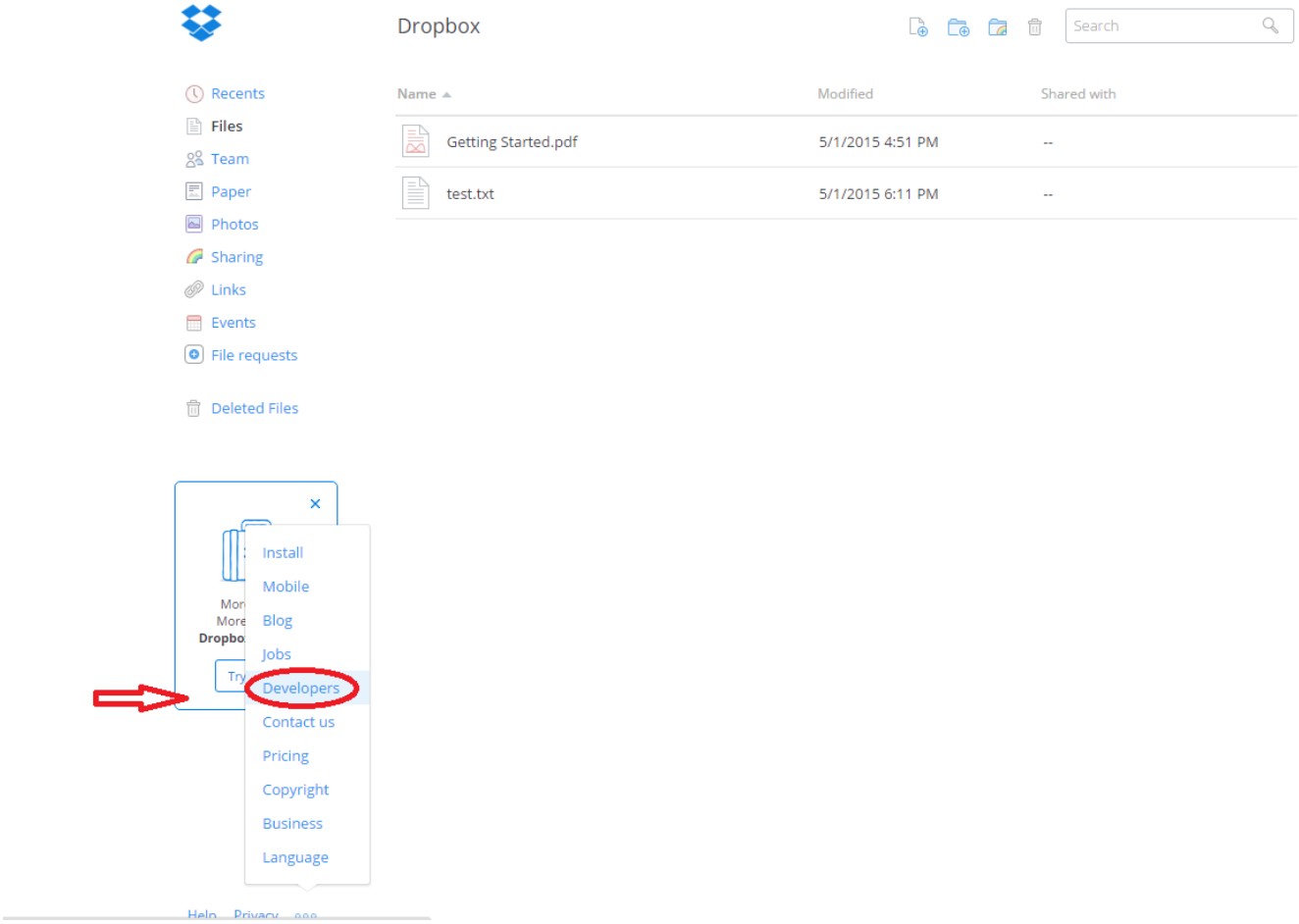

图 8-1. “Developers”选项卡

2. 选择“Create your app”，如图 8-2 所示。

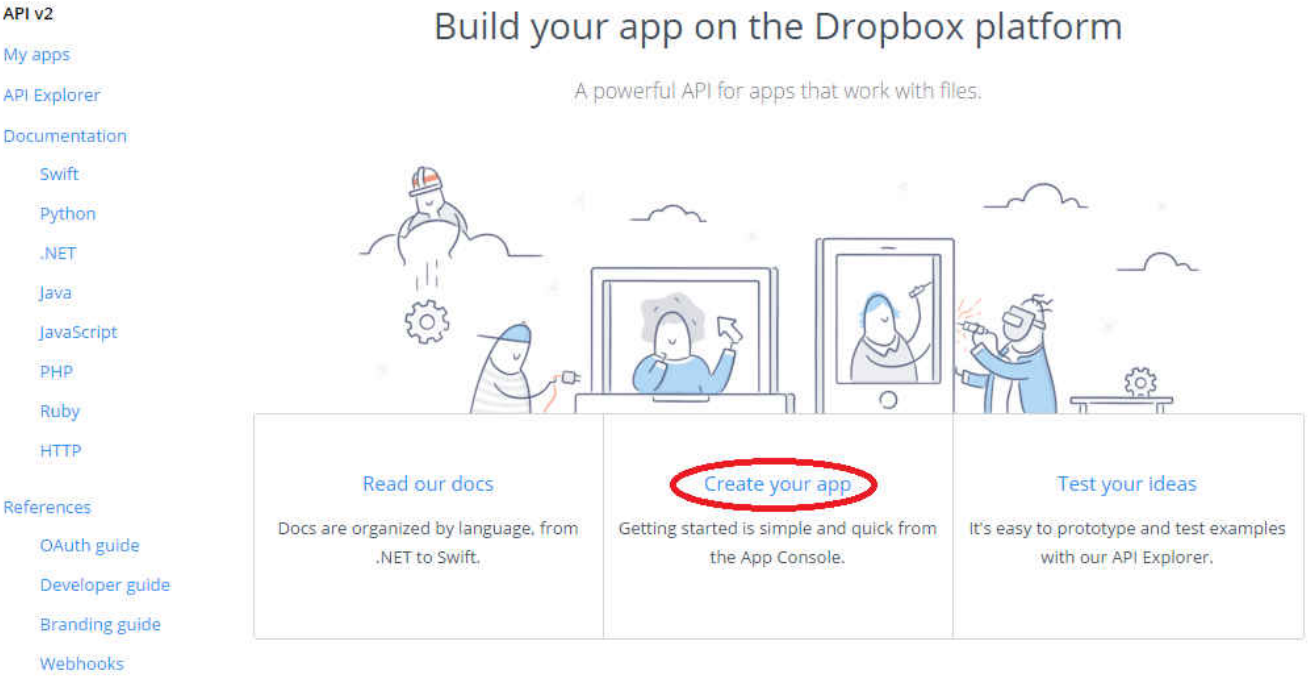

图 8-2. 创建您的应用

3. 选择“Dropbox API”。对于访问类型，请选择“App folder”，如图 8-3 所示，并命名应用程序（例如 OTA_R2），然后点击“Create app”。

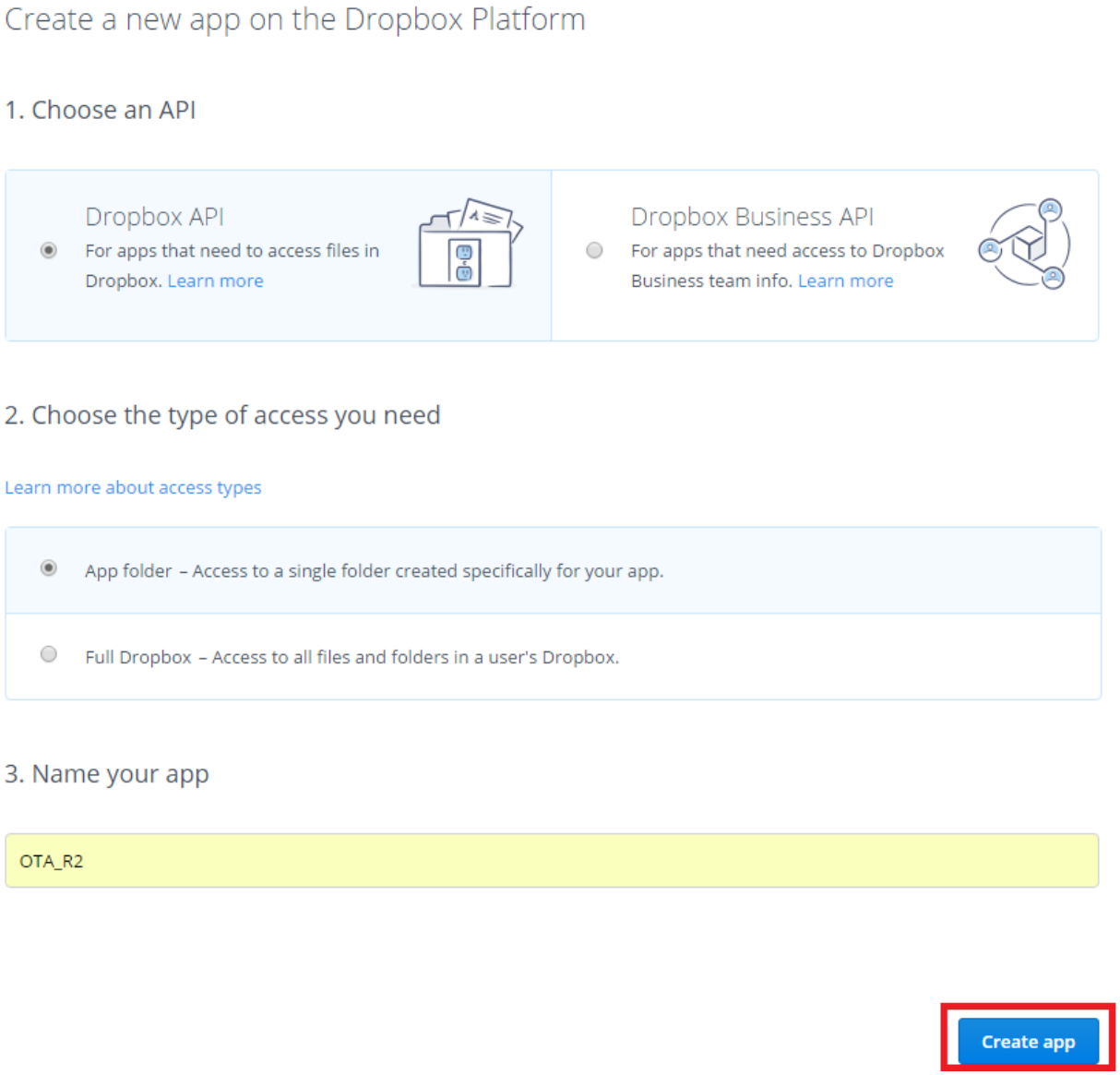

图 8-3. 创建应用

4. 选择“App key”和“App secret”，如图 8-4 所示。

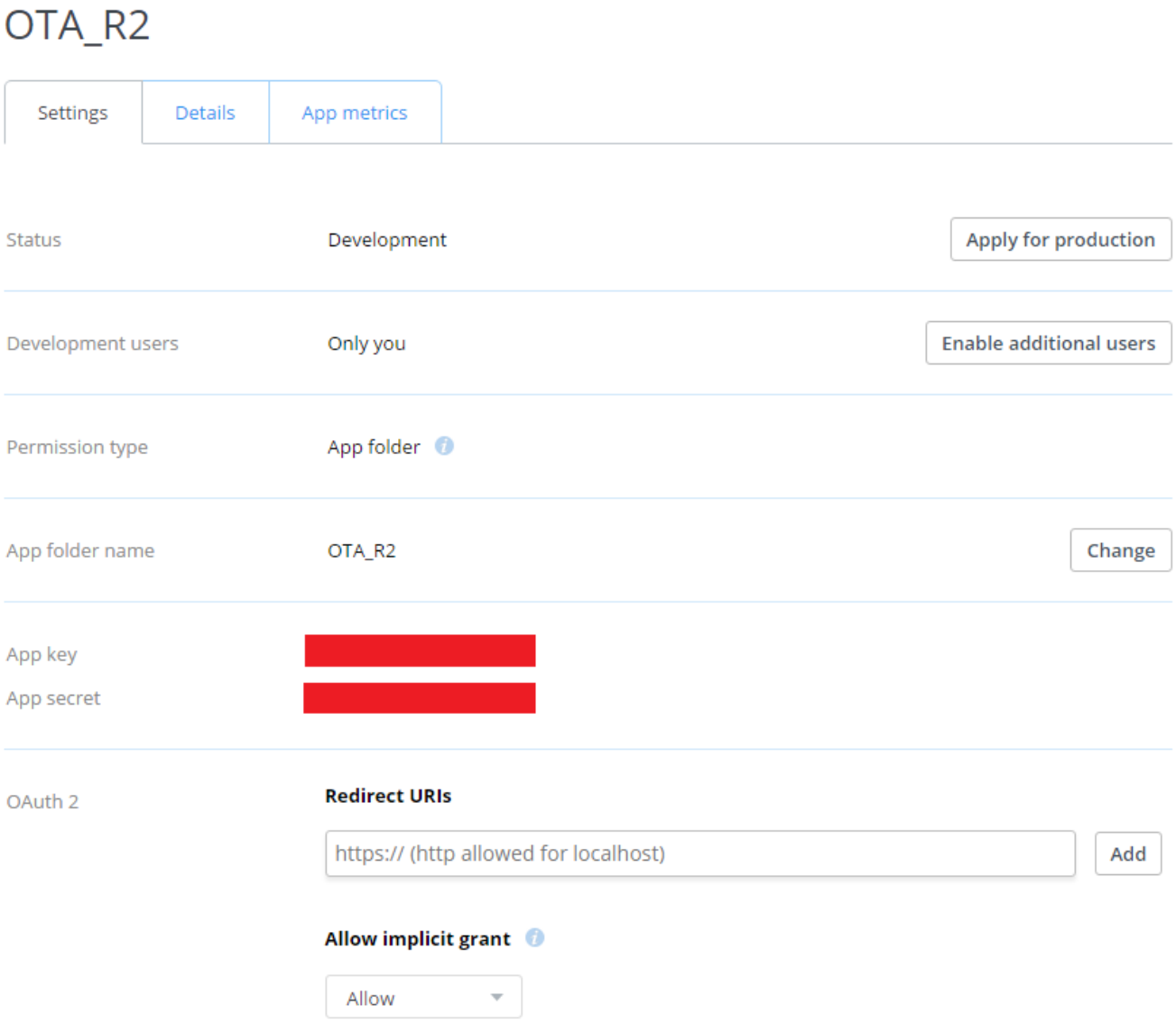

图 8-4. 应用密钥

5. 生成访问令牌，如图 8-5 所示。

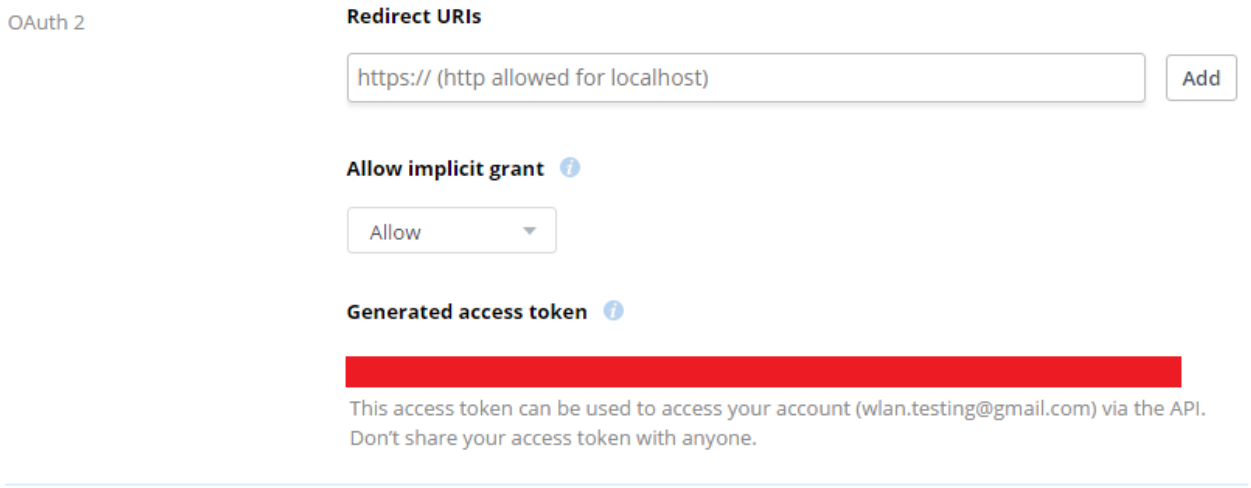

图 8-5. 生成访问令牌

备注

此分步指南介绍了如何基于开发人员 REST 客户端 API-v1 使用 Dropbox 服务器。如需更多信息，请参阅：https://www.dropbox.com/developers-v1/core/docs。

6. 使用前面生成的令牌、应用密钥和密码来授权应用。
7. 授权应用后，保存授权持有者令牌密钥。这个是 OTA 供应商令牌，应保存在 otauser.h 文件中。
8. 主页上会显示一个 Apps 文件夹，如图 8-6 所示。

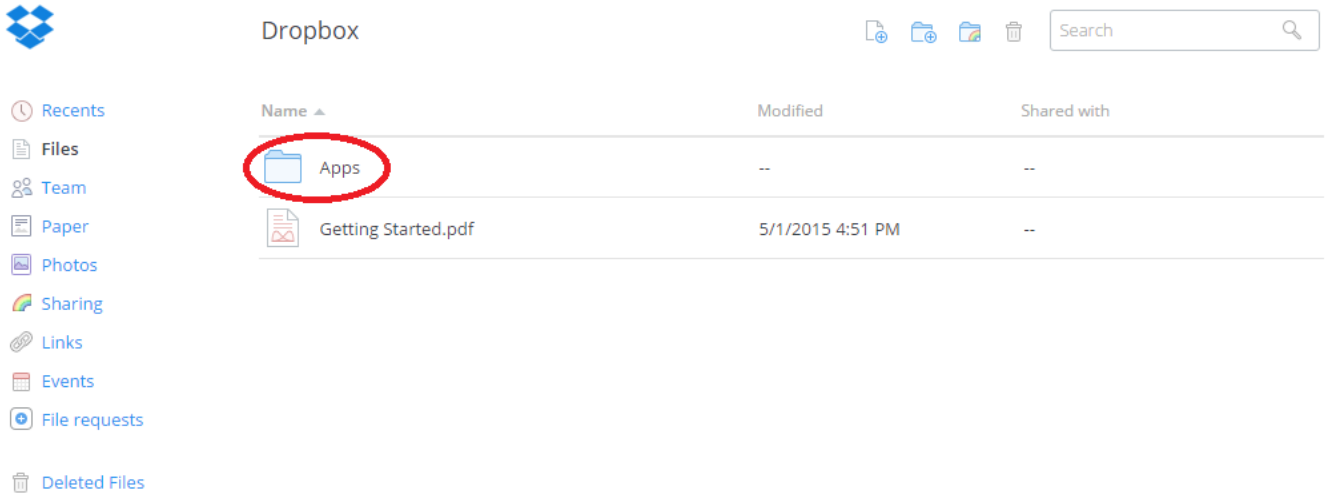

图 8-6. Apps 文件夹

9. 打开“Apps”。这时应该会显示前面创建的文件夹。打开该文件夹，如图 8-7 所示。

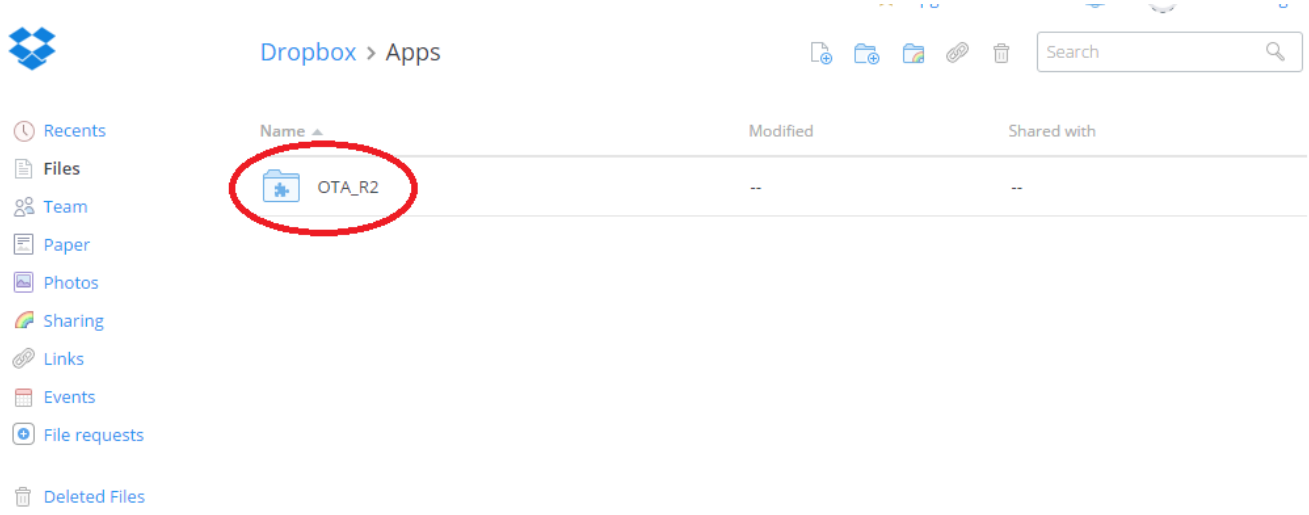

图 8-7. OTA 供应商目录

10. 创建新文件夹并为它指定 otauser.h 文件中 #define OTA_VENDOR_DIR 下显示的相同名称（请参阅图 8-8）。将 .tar 文件置于此处。有关如何创建 .tar 文件的信息，请参阅节 6。

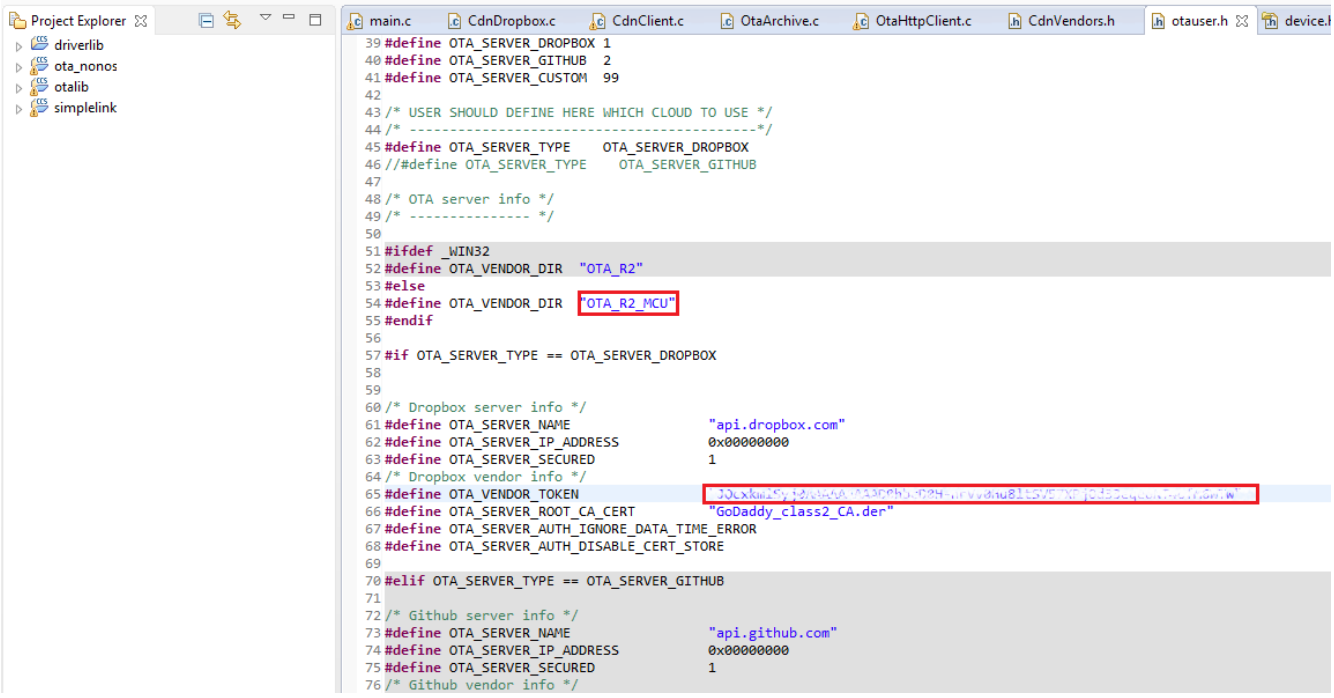

图 8-8. 编辑标出的字段

11. 编译云 OTA 工程并运行。

8.2 GitHub CDN 入门

以下步骤介绍了如何使用 GitHub CDN。

1. 在 gitHub 上创建帐户。
2. 点击“Create new...”按钮，以创建新的存储库（请参阅图 8-9）。

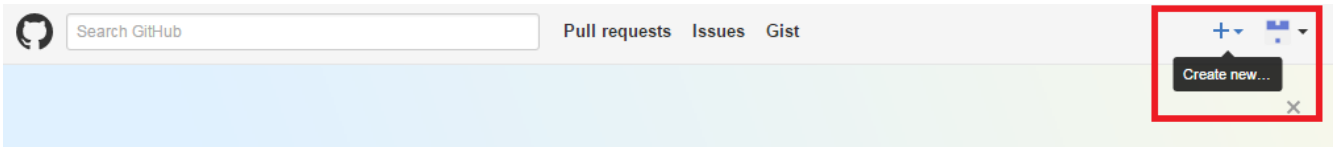

图 8-9. 新建

3. 选择“New repository”（请参阅图 8-10）。

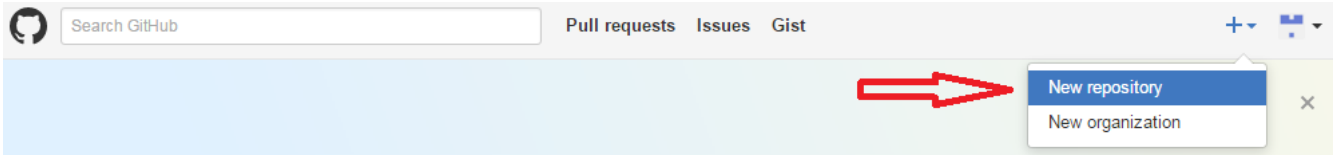

图 8-10. 新存储库

4. 命名存储库，选择公共或私有访问类型，然后点击“Create repository”，如图 8-11 所示。

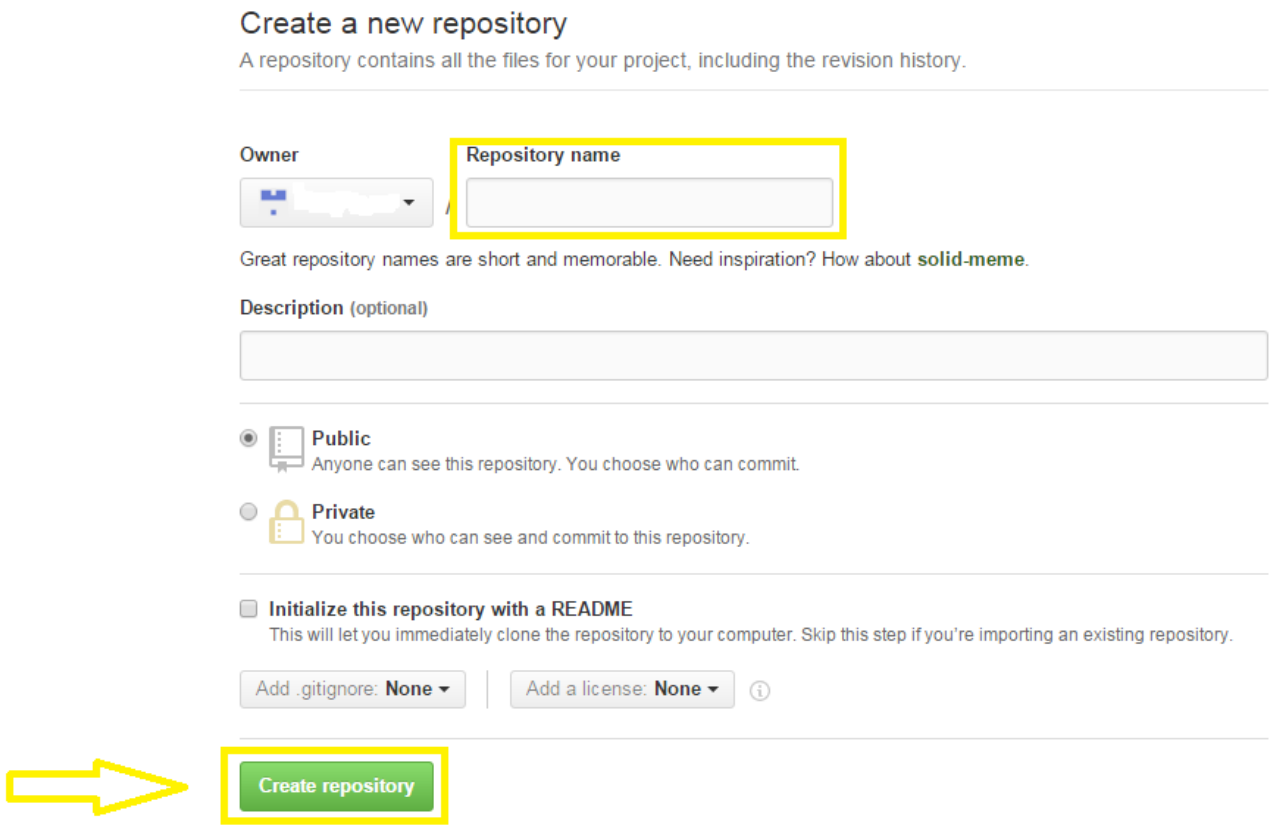

图 8-11. 创建新存储库

5. 保存存储库 URL，如图 8-12 所示。示例为：https://github.com/<username>/<repository name>。

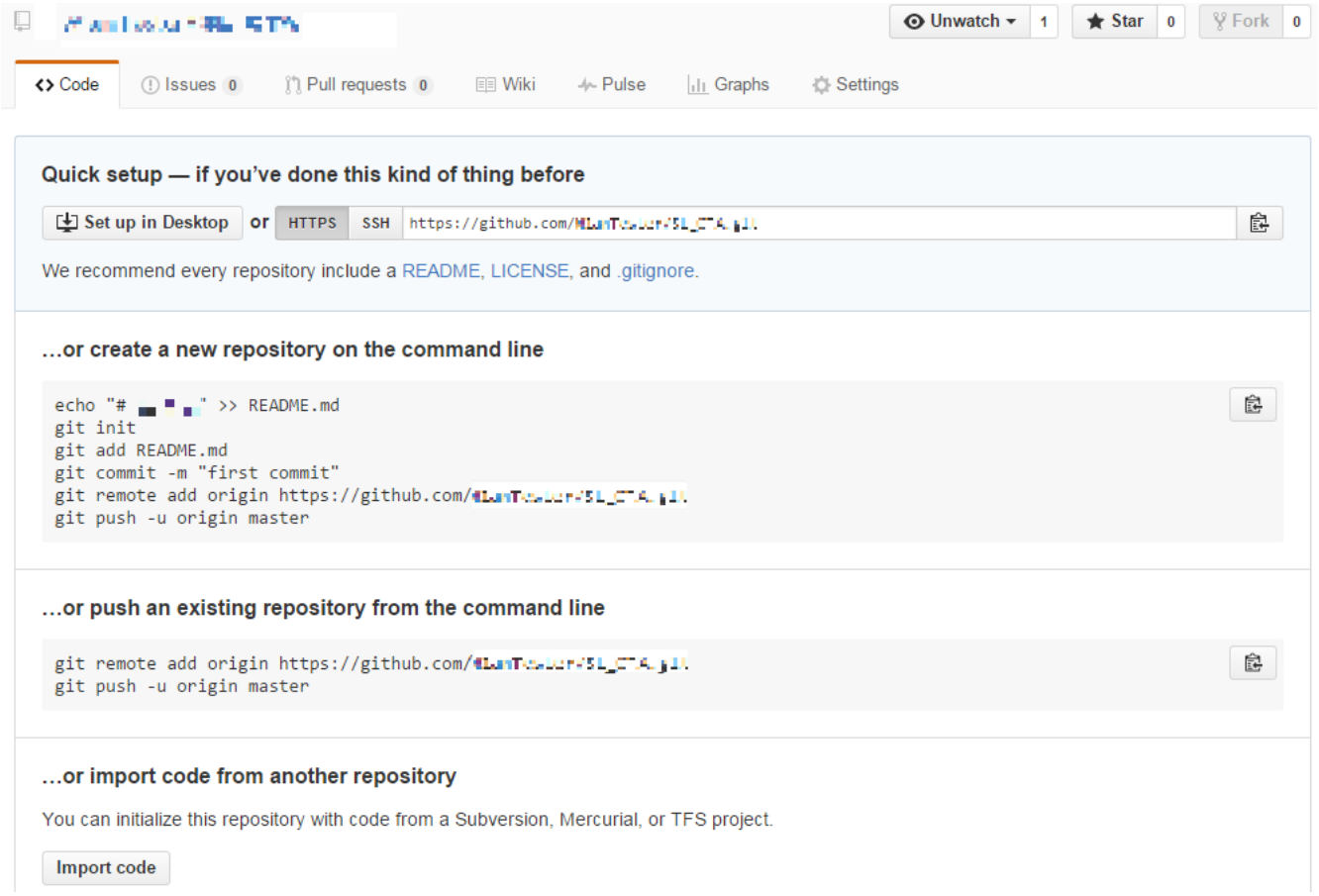

图 8-12. 保存存储库

以下说明显示了如何通过在用户的 PC 上使用 Git GUI 来激活 GitHub 存储库。从以下位置下载 Git GUI：https://git-for-windows.github.io/。

6. 在 PC 上创建一个目录。

7. 右键点击以打开 Git 选项，然后选择“Git Bash Here”，如图 8-13 所示。

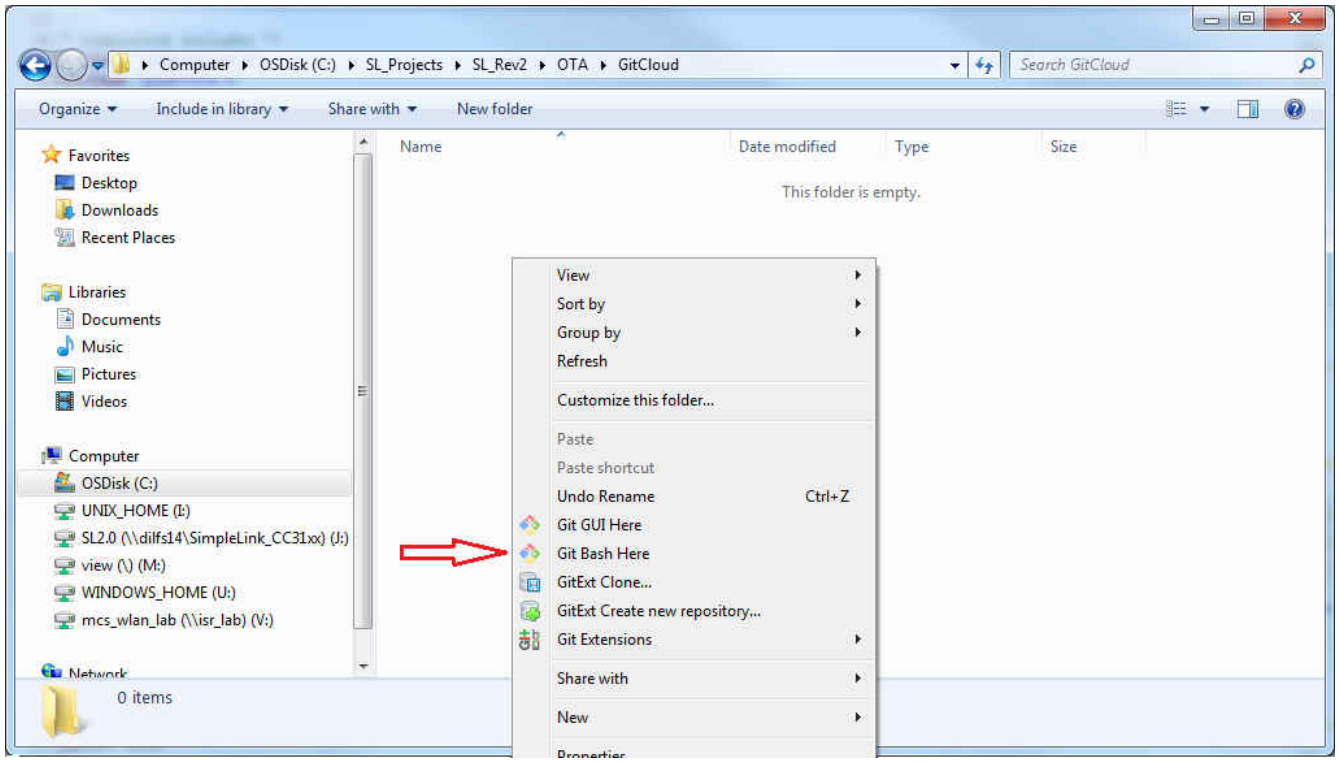

图 8-13. Git Bash Here

8. 在 Git Bash 窗口中键入以下内容：

```
git init
git clone https://github.com/<username>/<repository>
git add cd <user directory>
git add <user tar file>
git commit -a -m 'add OTA …'
git remote
git push origin master
```

9. 输入用户名和密码。现在，可以在 github 中看到该 SP .tar 文件。
10. 激活 GitHub CDN 存储库后，打开云 OTA 示例应用并编辑 otauser.h 文件，如图 8-14 所示。
    - 取消注释 OTA_SERVER_GITHUB：

```
42
43 /* USER SHOULD DEFINE HERE WHICH CLOUD TO USE */
44 /* ------------------------------------------*/
45 //#define OTA_SERVER_TYPE    OTA_SERVER_DROPBOX
46 #define OTA_SERVER_TYPE    OTA_SERVER_GITHUB
47
48 /* OTA server info */
49 /* --------------- */
50
```

图 8-14. 在 otauser.h 中设置服务器名称

    - 编辑图 8-15 中所示的字段。

```
70 #elif OTA_SERVER_TYPE == OTA_SERVER_GITHUB
71
72 /* Github server info */
73 #define OTA_SERVER_NAME                 "api.github.com"
74 #define OTA_SERVER_IP_ADDRESS           0x00000000
75 #define OTA_SERVER_SECURED              1
76 /* Github vendor info */
77 #define OTA_VENDOR_ROOT_DIR             "/repos/<user_name>/<OTA_Dir>"
78 #define OTA_VENDOR_TOKEN                "user_name"
79 #define OTA_SERVER_ROOT_CA_CERT         "DigCert_High_Assurance_CA.der"
80 #define OTA_SERVER_AUTH_IGNORE_DATA_TIME_ERROR
81 #define OTA_SERVER_AUTH_DISABLE_CERT_STORE
82
```

图 8-15. 在 otauser.h 中设置目录

11. 编译相关云 OTA 应用程序，然后运行。

9 本地链接支持

本地链接上的 OTA 是从本地移动设备完成的，而不是从云端。OTA 库的一部分可用于为本地链接网络上的 OTA 提供支持。应用程序使用 NetApp API 从本地移动设备获取存档文件数据块，并仅使用 OtaArchive 和 OtaJson 模块来处理存档文件数据块并将它们储存在 NWP 文件系统中所请求的文件中。

```
 #include "OtaArchive.h"
    OtaArchive_t gOtaArcive;
    _u8  gPayloadBuffer[1400];
    _i32 processedBytes=0;
    _i32 unprocessedBytes=0;
    _i32 accumulatedLen;
    _i32 chunkLen;
    /* Init the Tar parser module */
    OtaArchive_Init(&gOtaArcive);
    while (NOT_END_OF_ARCHIVE_SIZE)
    {
        /* copy the unprocessed part to the start of the buffer */if (unprocessedBytes > 0)
        {
            COPY_UNPROCCESSED(&gPayloadBuffer[0], &gPayloadBuffer[processedBytes],
unprocessedBytes);
        }
        /* read file chunk */
        READ_LOCAL_LINK(&chunkLen, &gPayloadBuffer[unprocessedBytes], &flags);
        otaChunkLen = chunkLen + unprocessedBytes;
        /* process the chunk */
        status = OtaArchive_Process(&gOtaArcive, gPayloadBuffer,
                                            otaChunkLen, &processedBytes);
        unprocessedBytes = otaChunkLen - processedBytes;
    }
    if (status == 0) /* Download done.Need to reset the MCU */
    {
        cc3200Reboot();
    }
```

重新启动后，检查以提交新的映像。

```
/* Check if OtaArchive is in SL_FS_BUNDLE_STATE_PENDING_COMMIT */if
(OtaArchive_GetPendingCommit())
    {
        /* Commit and continue */
        OtaArchive_Commit();
    }
```

10 支持新的 CDN 供应商

OTA 库支持两个 CDN 供应商：Dropbox 和 Github。供应商还可以使用另一个 CDN 服务器。本章介绍了如何定义此定制 CDN。

10.1 otauser.h

将 CDN 添加至列表并将其定义为选定服务器：

```
#define OTA_SERVER_DROPBOX 1
#define OTA_SERVER_GITHUB  2
#define OTA_SERVER_DROPBOX_V2 3
#define OTA_SERVER_CUSTOM  99
#define OTA_SERVER_TYPE    OTA_SERVER_CUSTOM
```

添加定制供应商定义部分：

```
#define OTA_VENDOR_DIR  "OTA_CC3220SF"
/* Custom server info */#define OTA_SERVER_NAME                "api.custom.com"
#define OTA_SERVER_IP_ADDRESS          0x00000000
#define OTA_SERVER_SECURED             1
/* Custom vendor info */#define OTA_VENDOR_TOKEN               "<Custom server access token>"
#define OTA_SERVER_ROOT_CA_CERT        "<Custom server ROOT CA cert file>"
#define OTA_SERVER_AUTH_IGNORE_DATA_TIME_ERROR
#define OTA_SERVER_AUTH_DISABLE_CERT_STORE
```

主机应用程序可以跳过 GetHostByName 并在 OTA_SERVER_IP_ADDRESS 中定义服务器 IP 地址。如果定制服务器不支持服务器身份验证和域名验证，则 OTA_SERVER_ROOT_CA_CERT 应为未定义。

10.2 ota/source/CdnVendors/Custom.c

添加该文件并实现以下函数（请参阅 Dropbox.c 和 Github.c 中的示例）：

1. CdnCustom_SendReqDir – 使用定制服务器名称、定制供应商目录和定制供应商令牌，发送目录树请求。
2. CdnCustom_ParseReqDir – 解析定制服务器的目录树回复并查找每个文件的以下两个令牌：
   - custom_file_name – 列表中的文件名
   - custom_file_size – 文件大小

   备注

   OTA 库仅使用前四个文件来搜索 .tar 文件。
3. CdnCustom_SendReqFileUrl – 使用服务器名称、请求的文件名和定制服务器令牌，向定制服务器发送文件 URL 请求。
4. CdnCustom_ParseRespFileUrl – 解析定制服务器的文件 URL 响应，并查找 URL 令牌：

   custom_file_url – 文件 URL

   备注

   OTA 库仅使用前四个文件。
5. CdnCustom_SendReqFileContent – 从定制文件服务器发送文件内容请求。

10.3 ota/source/CdnVendors/CdnVendors.h

添加定制的供应商宏：

```
#define CdnVendor_SendReqDir            CdnCustom_SendReqDir
#define CdnVendor_ParseRespDir          CdnCustom_ParseRespDir
#define CdnVendor_SendReqFileUrl        CdnCustom_SendReqFileUrl
#define CdnVendor_ParseRespFileUrl      CdnCustom_ParseRespFileUrl
#define CdnVendor_SendReqFileContent    CdnCustom_SendReqFileContent
```

修订历史记录

Changes from Revision A (January 2019) to Revision B (August 2020)	Page
• 更改了文档标题和整个文档以包括 CC3130 和 CC3230 器件	3

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265