Application Report
SWRA510–February 2017

# CC3x20 SimpleLink™ Wi-Fi® and Internet-of-Things Over the Air Update

*Ilan Biton*

**ABSTRACT**

Over-the-Air (OTA) update is wireless delivery of new software updates or configurations to embedded devices. Using the concept of an identity key (IRK) within the Internet-of-Things (IoT), OTA is an efficient way of distributing firmware updates or upgrades.

This document describes the OTA library for the SimpleLink™ Wi-Fi® CC3x20 family of devices from Texas Instruments™ and explains how to prepare a new cloud-ready update to be downloaded by the OTA library.

The SimpleLink MCU portfolio offers a single development environment that delivers flexible hardware, software, and tool options for customers developing wired and wireless applications. With 100 percent code reuse across host MCUs, Wi-Fi®, *Bluetooth*® low energy, Sub-1 GHz devices and more, choose the MCU or connectivity standard that fits your design. A one-time investment with the SimpleLink software development kit (SDK) allows you to reuse often, opening the door to create unlimited applications. For more information, visit www.ti.com/simplelink.

**Contents**

# 1 Introduction

The OTA library for the SimpleLink CC3x20 family of solutions simplifies the effort of MCU applications to access the cloud and download new upgrades (such as new firmware applications, service packs, and user files) in a secured and fail-safe manner, while keeping the integrity of the system.

The OTA library exposes a simple API set:

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

This API set is compatible for non-OS and OS-based platforms.

## 1.1 OTA System Block Diagram

Figure 1 shows the block diagram of the OTA system.

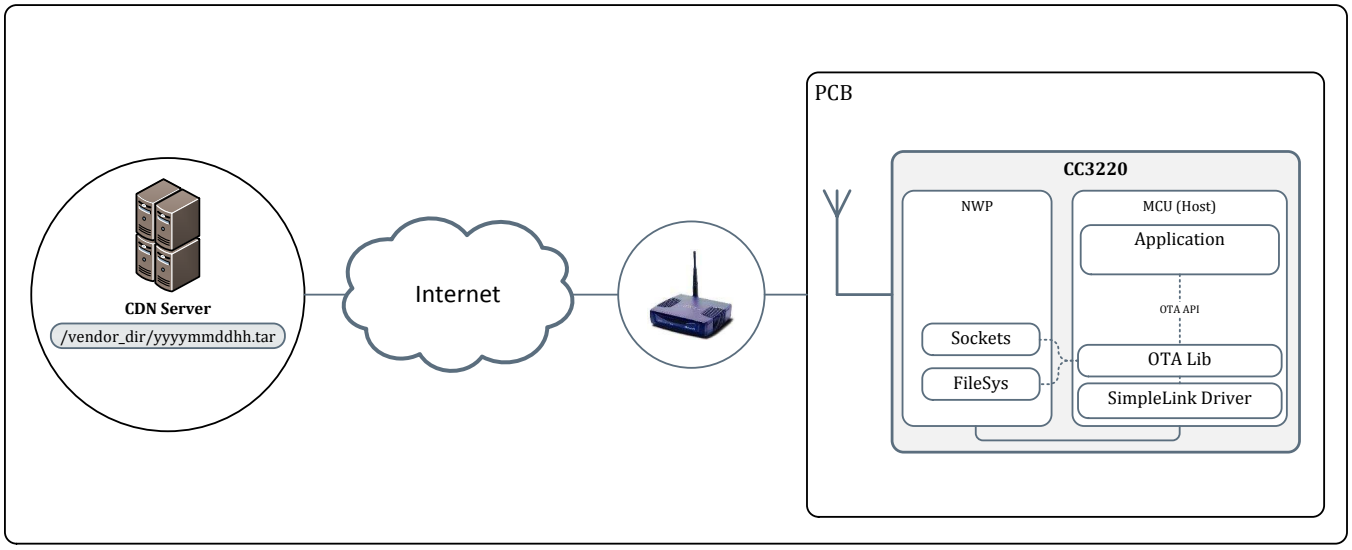


**Figure 1. OTA System Diagram**

The OTA library supports the following cloud CDN vendors:

- Dropbox™
- Github
- Custom (see Section 10)

The OTA library implements a simple HTTP client (TCP) to connect to the CDN server. This client can be configured by the host application as follows:

- Non-secured (Connect to CDN running HTTP server)
- Secured (Connect to CDN running HTTPS server)
  - Server authentication (required by default)
  - Domain name verifications (required by default)
  - No server authentication

The software upgrade application and user files should be put in an archive tar file. By default, all files are non-secured and fail-safe. The vendor can change the attributes of a file (such as secured, signature, certificate file name, and so forth) by adding entry to a command file (ota.cmd), which is in JSON format and included in the tar archive.

## 1.2 OTA Software Block Diagram

Figure 2 shows the block diagram of the OTA software.

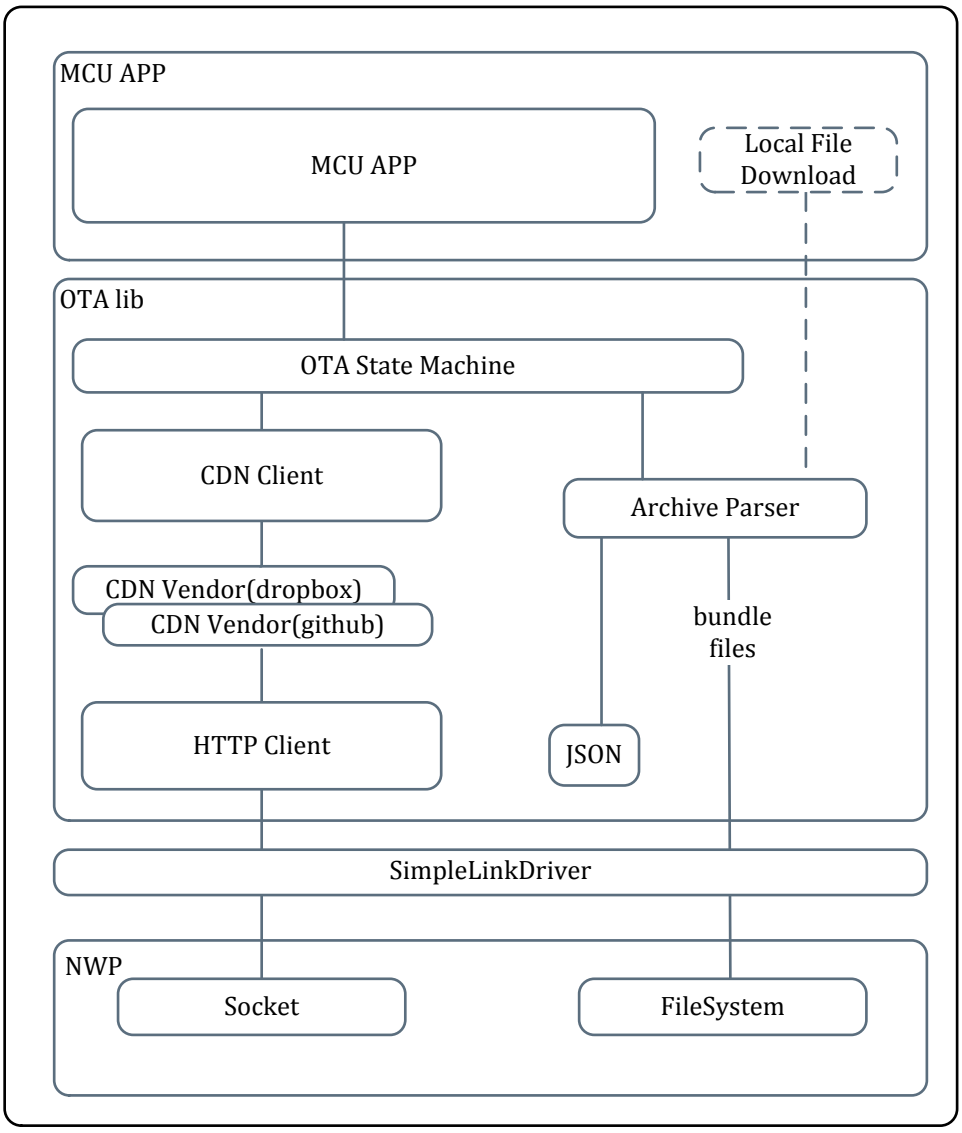


**Figure 2. OTA Software Block Diagram**

### 1.3 Terminology and Abbreviations

Table 1 lists terminology and abbreviations used in this documentation.

**Table 1. Terminology**

| Term | Description |
|---|---|
| OTA | Over-the-Air |
| CDN | Content delivery network. In the relation to the OTA lib, CDN relates to the servers that contain the new package. |
| ACM | Account manager |
| CSP | Cloud storage provider |
| HTTP | Hypertext transfer protocol |
| URL | Uniform resource locator |
| URI | Uniform resource identifier |
| REST | Representational state transfer |
| JSON | JavaScript object notation |

## 2 High-Level Description for Adding OTA Capability

Existing host applications can be expanded to have OTA capability by using the OTA library and the OTA sample application as a reference.

The OTA sample application works on the CC3220 and CC3120 devices with an MSP432™ microcontroller (MCU) as the host.

### 2.1 Software Installations and Downloads

Follow the SDK Getting Started Guide (CC3120 or CC3220) to prepare development environment as follows:

1. Download the newest version of Code Composer Studio™ (CCS).
2. Download the latest CC3220SDK or CC3120SDK.
3. Download the latest UniFlash tool, select CC3220 device, and start the Image Creator tool inside.
4. Download the latest CC3220 service pack.
5. Install the SimpleLink Starter Pro on your mobile phone.

### 2.2 OTA Process Brief

The following steps explain, at a high level, the steps to prepare and run the OTA application.

1. Create a developer account in the selected CDN server (Dropbox, Github, or custom).
2. Open the OTA sample application and change the selected CDN server info.
3. Compile the OTA sample application.
4. Create a TAR file with the host application and all other required files.
5. Upload the TAR file into the CDN server.
6. Open the UART terminal to view the application logs.
7. Using the Image Creator, prepare an image containing the OTA application, and program the device.
8. When the host application starts the OTA process, download and run the new host application.

**NOTE:** Replacing the TAR file in the CDN server with a newer one should reinvoke the OTA download.

# 3 Running the Default OTA Sample Application

The OTA sample application provides a basic application with Cloud OTA and provisioning capabilities. This application is described in more detail in Section 5 and can be used as a basis for developing a target application.

The Cloud OTA sample application supports:

- OS: FreeRTOS or TI-RTOS
- CC3220 variant: CC3220R, CC3220S, CC3220SF
- Development tools: CCS6.2 and IAR7.5

The following sections are for specific a CC3220 variant and OS; change the names for a different OS or CC3200 variant. For the CC3220R, use the name CC3200S.

**NOTE:** The OTA library produces compilation errors. Open a developer account on Dropbox or Github, and get a token to be used in <cc3220_sdk_install_dir>\source\ti\net\ota\otauser.h. For Dropbox cloud, see Section 8.1 and Section 4.1.

## *3.1 Building the Cloud OTA CCS Project for the CC3220SF*

Install the CC3220 SDK and import three CCS projects:

- OS project
- OTA library project
- OTA application project

**NOTE:** After installing CCS6.2 or later, also install the ccs_patch located in <cc3220_sdk_install_dir>\tools\ccs_patch.

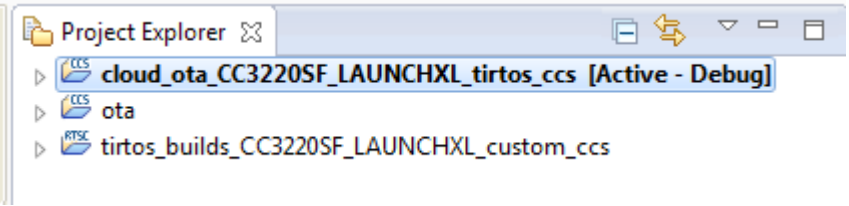


**Figure 3. Selected Cloud OTA Project Name**

1. Import the desired OS (freertos or tirtos) project from the following location, according to the connected CC3220 device variant (CC3220S or CC3220SF):
   - For FreeRTOS and CC3220SF devices:
     <cc3220_sdk_install_dir>\os\freertos\builds\CC3220SF_LAUNCHXL\custom
   - For TI-RTOS and CC3220SF devices:
     <cc3220_sdk_install_dir>\os\tirtos\builds\CC3220SF_LAUNCHXL\custom
2. Import the OTA library project using CCS IDE from the following location:
   <cc3220_sdk_install_dir>\source\ti\net\ota

   **NOTE:** This project should not be copied to the wCCS workspace.
3. Import the Cloud OTA project using CCS IDE from the following location, according to the connected CC3220 flavor and the desired OS (CC3220S or CC3220SF):
   - For FreeRTOS and CC3220SF devices:
     <cc3220_sdk_install_dir>\examples\os\CC3220S_LAUNCHXL\demos\cloud_ota\freertos

- For TI-RTOS and CC3220SF devices:

<cc3220_sdk_install_dir>\examples\os\CC3220SF_LAUNCHXL\demos\cloud_ota\tirtos

4. Configure cloud server.
   (a) Open a developer account on Dropbox or Github. For instructions, see Section 8.1 and Section 4.1.
   (b) Copy the given cloud server token into the OTA lib user configuration file:
   <cc3220_sdk_install_dir>\source\ti\net\ota\otauser.h in

```
#define OTA_SERVER_TYPE    OTA_SERVER_DROPBOX
#define OTA_VENDOR_TOKEN   "<Dropbox server access token>"
#define OTA_VENDOR_DIR     "OTA_R2_MCU_FLASH" /* for CC3220SF device */
```

5. Configure OTA special compilation flags.
   - OTA lib – Define SL_ENABLE_OTA_DEBUG_TRACES in otauser.h for detailed OTA library debug prints
   - OTA project – Define DISABLE_OTA_SWITCH_TRIGGER in cloud_ota.c to start the OTA process after 5 pings, and not to wait for an external trigger (CC3220LP button). Define OTA_LOOP_TESTING to continue running OTA attempts when the return value is NO_UPDATE, OLDER_VERSION.
6. Complete other CCS configurations.
   (a) To produce a cloud_ota bin file for the Image Creator, open CCS project properties->Builds->Steps->Post-Build steps, and add the following command:

```
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin" "${BuildArtifactFileName}"
"${BuildArtifactFileBaseName}.bin" "${CG_TOOL_ROOT}/bin/armofd"
"${CG_TOOL_ROOT}/bin/armhex" "${CCE_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"
```

   (b) For debug mode, open the CCS view tab and select Target Configurations. Configure the Connection to Texas Instruments XDS110 USB Debug Probe.
7. Compile the project and produce a cloud_ota.bin file to be programmed by the UniFlash.
   (a) Build OS project – tirtos_builds_CC3220SF_LAUNCHXL_custom_ccs.
   (b) Build OTA library project – ota
   (c) Build OTA sample application project – cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs
   (d) Output bin file is in the projects workspace directory <cc3220_sdk_workspace_dir>\cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs\Debug\cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs.bin.

### 3.2 Program by UniFlash Tool With Image Creator Inside

The following instructions describe how to program the device.

1. Open UniFlash and select CC3220 device, as shown in Figure 4.

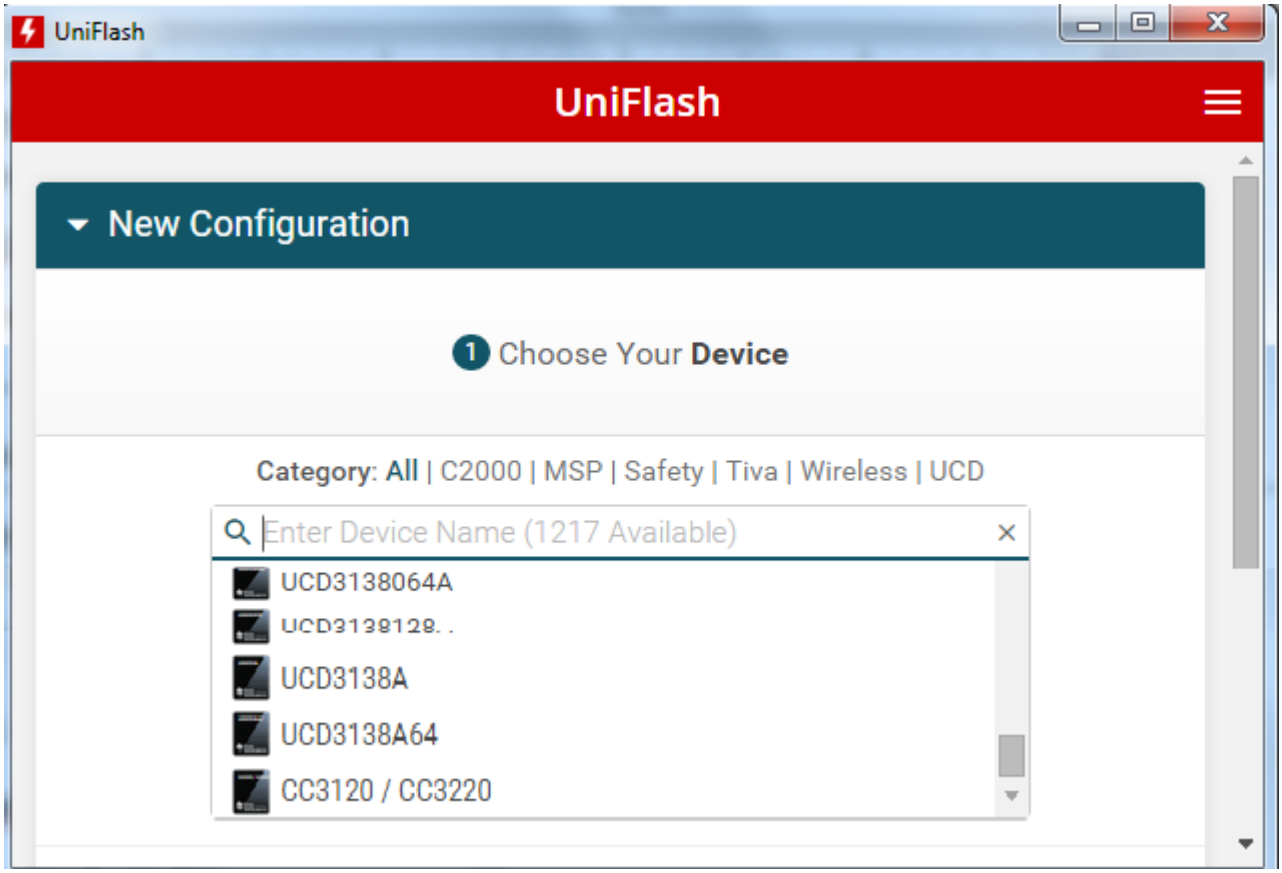


**Figure 4. Choose Your Device**

2. Start Image Creator and select New Project, fill the project name, and click create project (for debug mode, the device must be opened first in development mode).

3. Enter Files→Trusted Root-Certificate Catalog, and select <cc3220_sdk_install_dir>\tools\certificate-playground\certcatalogPlayGroung20160911.lst, as shown in Figure 5 (do not use the default).

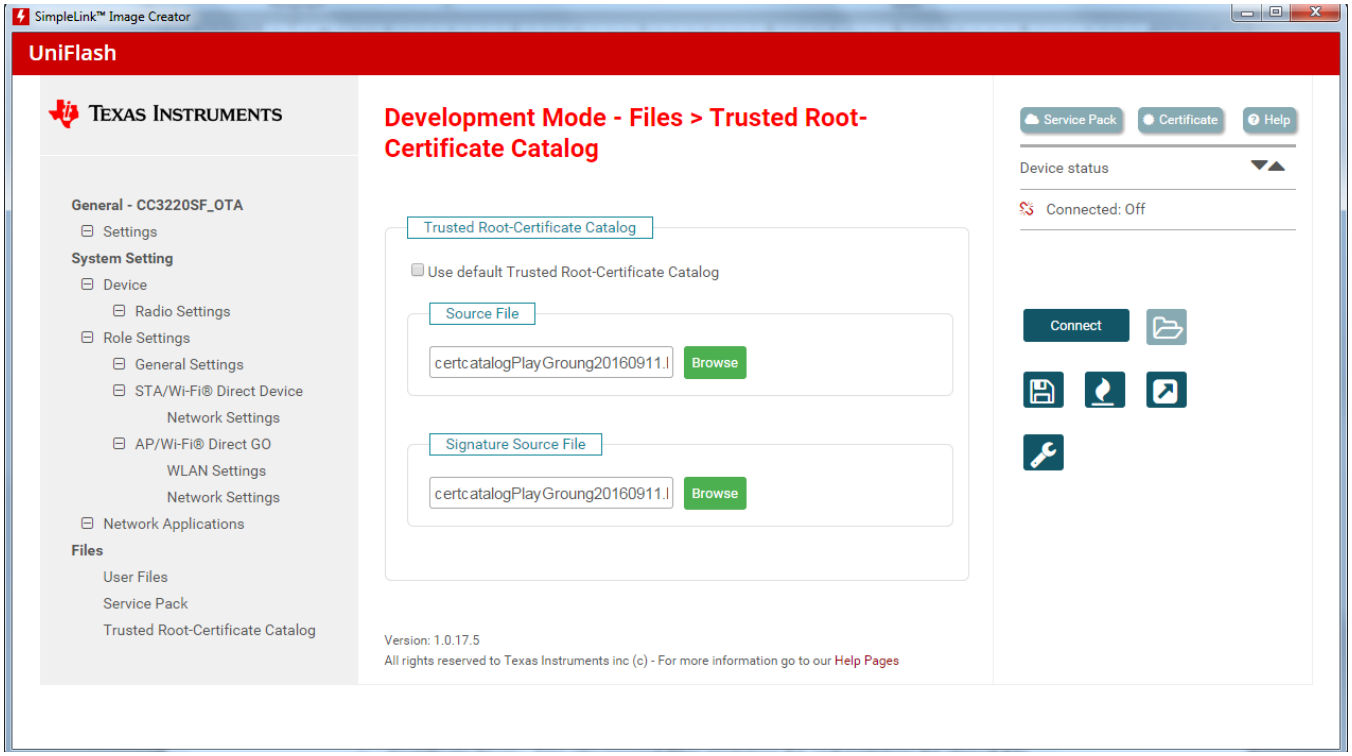


**Figure 5. Trusted Root-Certificate Catalog**

4. Enter Files→Service Pack, and select the latest networking subsystem service pack, as shown in Figure 6.


**Figure 6. Service Pack File Name**

5. Enter Files→User Files, and select the following, as shown in Figure 7.

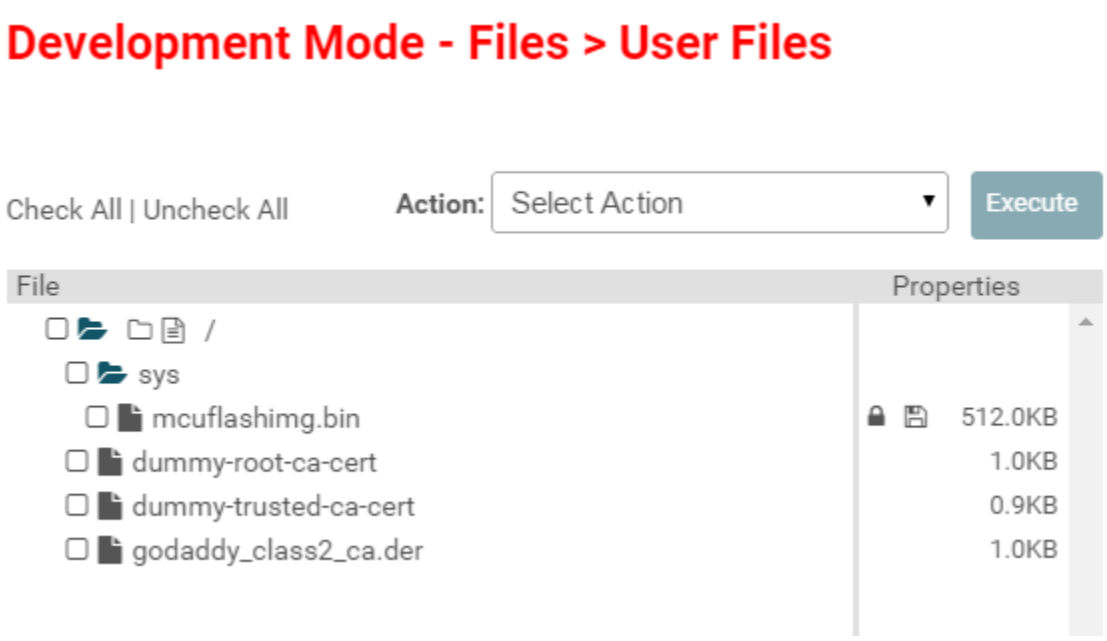


**Figure 7. User Files**

- Certificates files:
  - <cc3220_sdk_install_dir>\tools\certificate-playground\dummy-root-ca-cert
  - <cc3220_sdk_install_dir>\tools\certificate-playground\dummy-trusted-ca-cert
- Certificates for Dropbox and Github:
  - Download the Dropbox and Github Root CA certificates (that is, ”DigCert_High_Assurance_CA.der” and “GoDaddy_class2_CA.der”).
- User files MCU image (see Figure 8):
  - <cc3220_sdk_workspace_dir>\cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs\Debug\cloud_ota_CC3220SF_LAUNCHXL_tirtos_ccs.bin

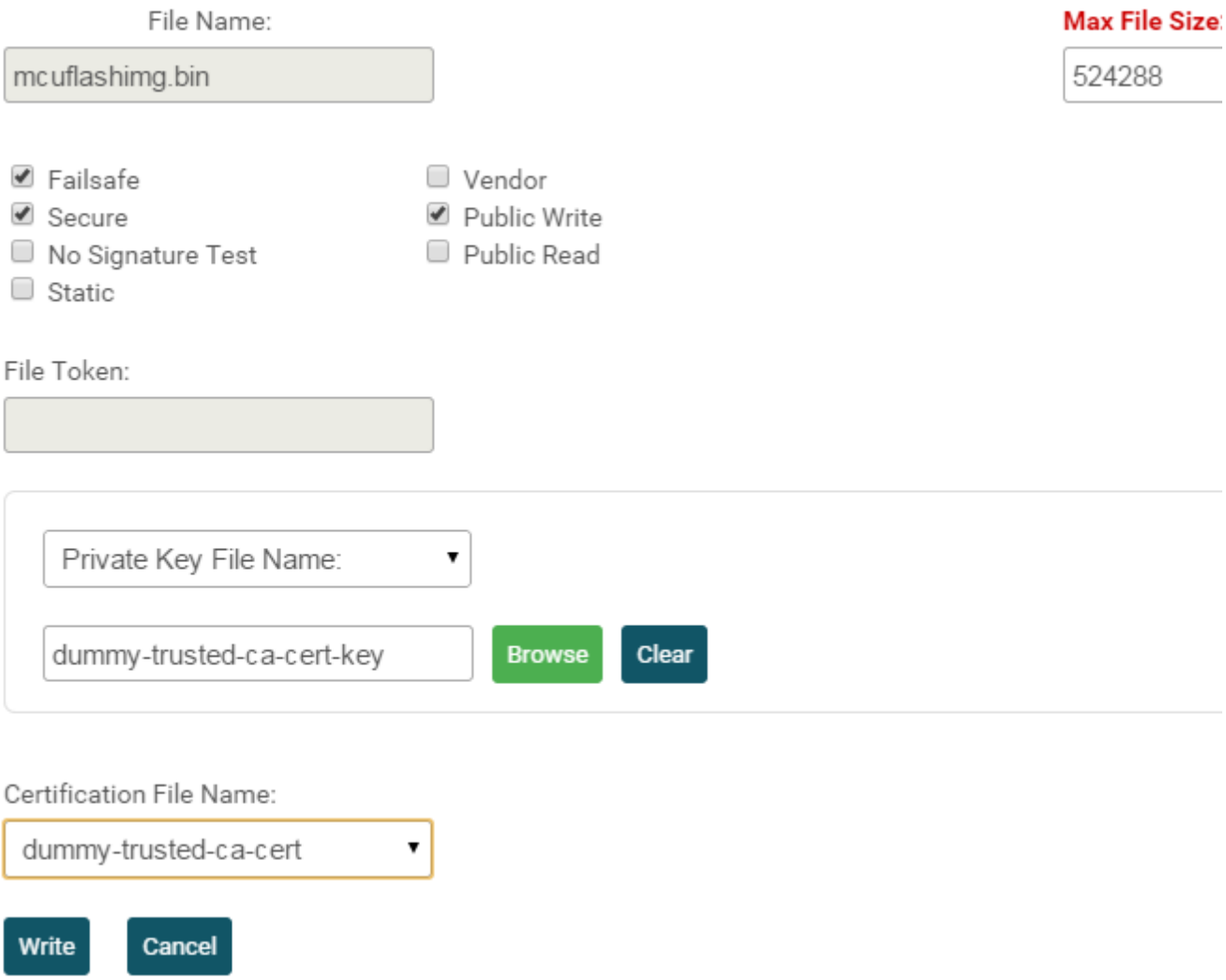


**Figure 8. MCU Image**

6. Configure the device to STA role.
7. Click Create OTA to produce a tar file to be put in the cloud server. The output file 2016091917_CC3220SF_CC3220SF_OTA.tar should be put in the cloud directory, as shown in Figure 9.

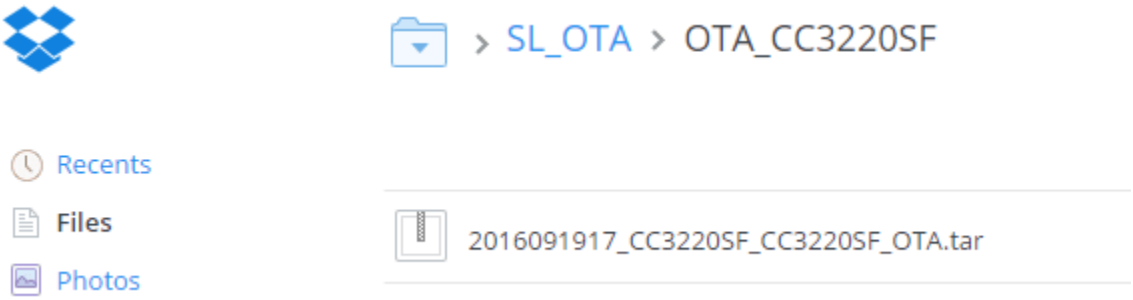


**Figure 9. Cloud Directory**

8. Configure in otauser.h:

```
#define OTA_VENDOR_DIR     "OTA_R2_MCU_FLASH" /* for CC3220SF device */
```

9. Click on Program Image (Create and Program), as shown in Figure 10. The image is now programmed to the device.

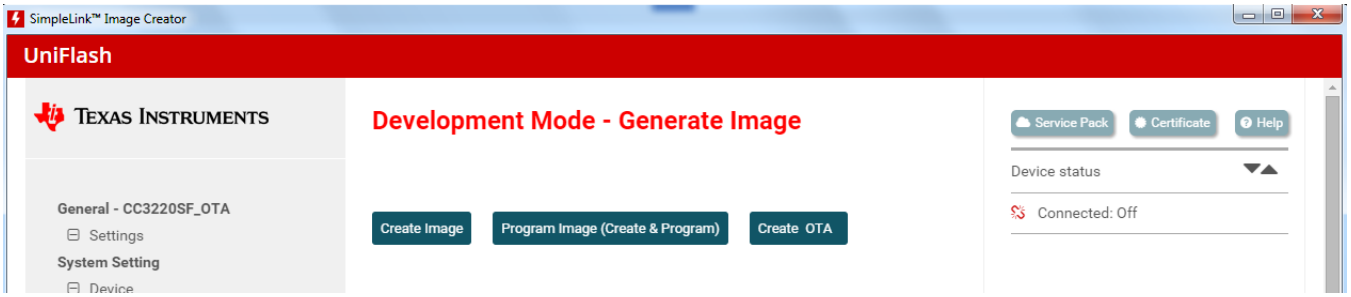


**Figure 10. Generate Image**

## 3.3 Running a Sample Application

After programming the OTA sample application in the device and putting the OTA tar file in the cloud, press reset to start the OTA process. The major steps that the sample application executes are as follows:

1. Connect to AP through an existing profile, or by the provisioning process.
2. Send pings and wait for the CC3220LP button click to start the OTA.
3. Configure the OTA library with the CDN server details.
4. Run the OTA process, initiated from external trigger (button on the CC3220LP board)
   (a) Connect to the CDN HTTPS server.
   (b) Request for directory content, and download the TAR archive file.
   (c) Extract TAR archive files, including the file mcuflashimg.bin, into the serial flash.
5. After the download is finished, set IMAGE_TESTING mode and reset the MCU.
6. On successful system tests, set IMAGE_COMMIT and continue running the new MCU image.

**NOTE:** A new OTA process can be initiated again by the user. The download process is only reinvoked if a new TAR file is uploaded to the CDN or when the user configures it to ignore the newer version check by #define OTA_LOOP_TESTING in the cloud_ota.c file.

## 3.4 Building the Cloud OTA IAR Project for CC3220SF

After installing IAR7.5 or later, also install the iar_patch located in <cc3220_sdk_install_dir>\tools\iar_patch.

1. Open cmd window in <cc3220_sdk_install_dir>\os\tirtos\builds\CC3220SF_LAUNCHXL\custom\iar, and run c:\ti\xdctools_3_32_01_22_core\gmake.exe.
2. Import an IPCF file into IAR Workbench:
   (a) Enable project connections: (Options->Project-Enable project connections).
   (b) Configure Custom Argument Variables: (Tools->Configure Custom Argument Variables, at the Global tab choose Import. Navigate to: SDK_installation_Dir->examples->CC32XX_SDK.custom_argvars, and open).
   (c) Verify that the paths imported are correct for your machine.
   (d) Create a new empty project. Go into Project->Create New Project..... and choose an empty project, choose a location and name to the project workspace, and Save.
   (e) Right-click the empty project, and select Add->Add Files.
3. Navigate to and select the desired IPCF file from <cc3220_sdk_install_dir>\examples\os\CC3220SF_LAUNCHXL\demos\cloud_ota\tirtos\iar\cloud_ota.ipcf. The user may need to change the Windows Explorer filtering on the bottom right to “All Files”.

4. There is no IAR project for the OTA library. To compile the IAR project, do the following:
   (a) Edit the otauser.h file, following the instructions in Section 4.1.
   (b) cd <cc3220_sdk_install_dir>/source/ti/net/ota
   (c) Edit Makefile.defs, and replace BUILDDIR=iar with BUILDDIR=ewarm.
   (d) gmake.exe IAR-ARMCOMPILER="C:/Program Files (x86)/IAR Systems/Embedded Workbench 6.5/arm" –f Makefile.defs OS_CONFIG=tirtos BUILD_TOOL=iar BUILD_CONFIG=Release -f Makefile.defs.
   (e) Rebuild the cloud_ota project in the IAR environment to link with the new ota.a library.
5. IAR debugger
   - Project->General Options->Target->Device select TexasInstaruments CC3220SF
   - Project->Debugger->Setup->Driver TI XDS
   - Project->Debugger->Download->Use flash loader

# 4 Adding OTA Capability Into an Embedded Software Application

This application note focuses on showcasing the ability of the CC3220 device to receive firmware updates and any related files over the Internet-enabled Wi-Fi interface.

## *4.1 Updating OTA Definitions*

The OTA library has support for the following CDN vendors:

- Dropbox
- Github

This section describes how to use one of these CDN vendors. To add support for another CDN vendor, refer to Section 10.

To run the OTA lib, a vendor must have an account in one of the supported CDNs to distribute the software updates. For more information how to open an account, refer to Section 8.

The CDN setting, including the credentials, are updated in the otauser.h file with the selected directory and token.

### 4.1.1 otauser.h

This file is found here: C:\ti\simplelink_cc32xx_sdk_1_02_02_00\source\ti\net\ota.

Select the OTA server vendor by defining OTA_SERVER_TYPE:

```
#define OTA_SERVER_TYPE    OTA_SERVER_DROPBOX
```

Define the vendor server information section:

```
#define OTA_VENDOR_DIR  "OTA_R2_MCU_FLASH"    /* for CC3220SF device      */

/* Custom server info */
#define OTA_SERVER_NAME                "api.dropbox.com"
#define OTA_SERVER_IP_ADDRESS          0x00000000
#define OTA_SERVER_SECURED             1

/* Custom vendor info */
#define OTA_VENDOR_TOKEN               "<Dropbox server access token>"
#define OTA_SERVER_ROOT_CA_CERT        "GoDaddy_class2_CA.der"
#define OTA_SERVER_AUTH_IGNORE_DATA_TIME_ERROR
#define OTA_SERVER_AUTH_DISABLE_CERT_STORE
```

The host application can set the server IP address in OTA_SERVER_IP_ADDRESS instead of server name.

By default, the lib requires server authentication and domain name verification. From a security perspective, the previous requirements are essential, and TI recommends using this mode. If the custom server is not secured, or server authentication and domain name verification are not required, OTA_SERVER_ROOT_CA_CERT can be undefined.

## 4.2 Using OTA lib in the Main Application

Figure 11 shows a basic possible flow of a host application running the OTA.

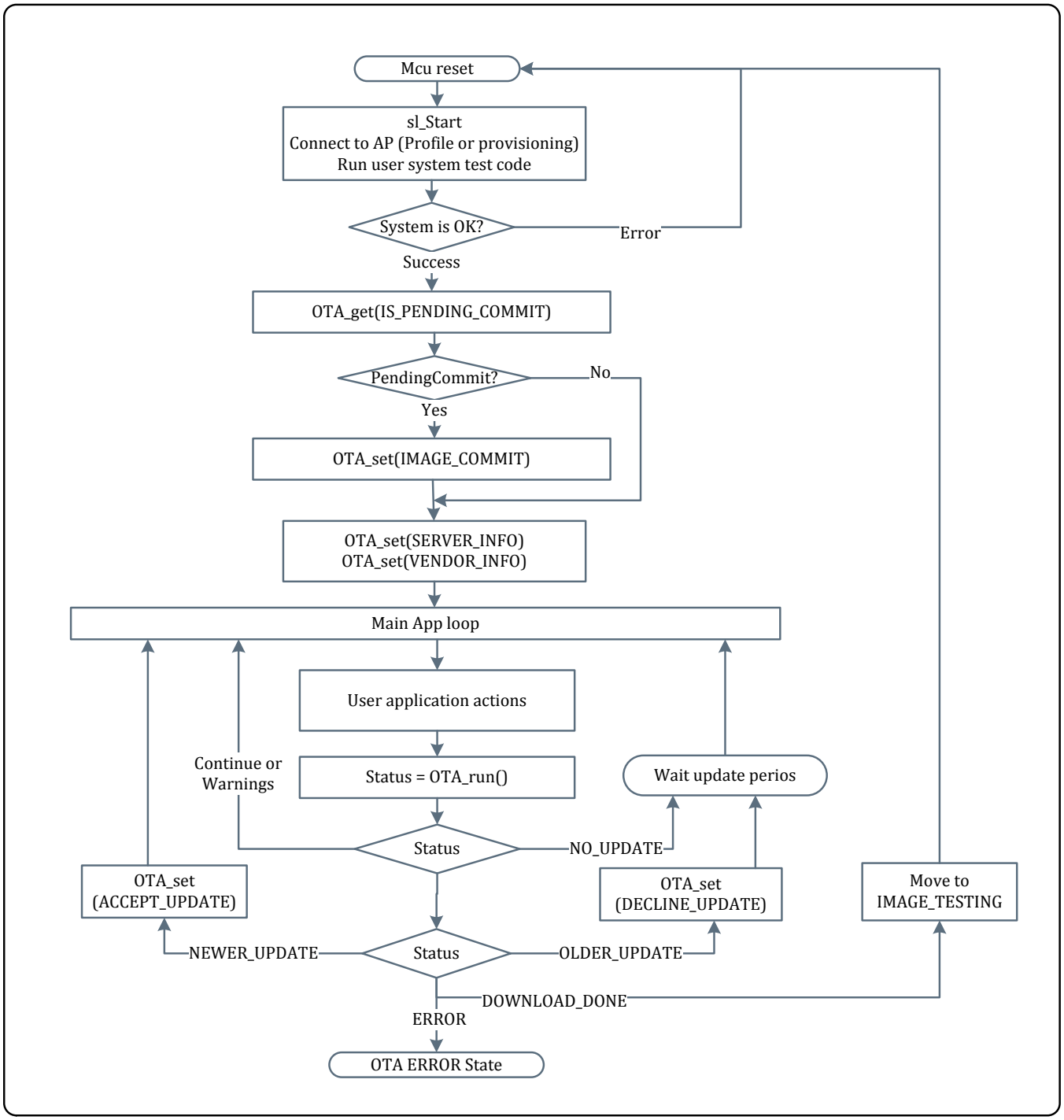


**Figure 11. Basic OTA Application Flow Chart**

### 4.2.1 Initiate the OTA Library

The host application must define a global buffer for the use of the OTA lib, and provide this buffer while it initiates the OTA:

```
#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 */
}
```

Set OTA server information:

```
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 */
}
```

Set OTA vendor 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 Running the OTA Process

The OTA process runs in steps side-by-side with the main application. The user should continue to call the OTA process as long as the return value is OTA_RUN_STATUS_CONTINUE.

The following example shows a simple main loop of a non-OS application:

```
_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 */
}
```

The possible OTA process return values are:

- OTA_RUN_STATUS_CONTINUE – Continue calling OTA_run.
- RUN_STAT_DOWNLOAD_DONE – Current OTA update session is completed. Host should reset the MCU and test the image. On MCU init, check if the system is connected, then test if the bundle is in pending commit state and set commit command. On MCU init error, rollback the new image bundle.
- OTA_RUN_STATUS_NO_UPDATES – There is no image to be downloaded; retry on next OTA period or external trigger.
- OTA_RUN_STATUS_CHECK_NEW_VERSION – OTA found a newer update version, the user can accept the new version by calling OTA_set with option _ ACCEPT_UPDATE, or continue running, or do another check before continuing.
- OTA_RUN_STATUS_CHECK_OLDEST_VERSION – OTA found an older update version, the user can stop the OTA by calling OTA_set with option _DECLINE_UPDATE, or continue running, or do another check before continuing.
- OTA_RUN_STATUS_CONTINUE_WARNING_FAILED__ <reason> – A group of OTA WARNINGS; errors, but the OTA will retry the process five times, so continue to run the OTA.
- OTA_RUN_ERROR_CONSECUTIVE_OTA_ERRORS - OTA process failed 5 consecutive times; reset the MCU to retry or to choose to stop the OTA.
- OTA_RUN_ERROR_<reason> – (Negative values) A group of OTA errors; stop the OTA process.
- OTA_RUN_ERROR_SECURITY_ALERT - Security alert from file system; stop downloading the current OTA update. The file system errors that cause this error are: 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, and 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 Commit Process

Upon a successful completion of the OTA process (Status RUN_STAT_DOWNLOAD_DONE), all software upgrade package files are stored in the serial flash in bundle mode, but are still not in use.

The host application should move the bundle into IMAGE_TESTING mode by calling sl_Stop() with a timeout different than 0, and then resetting the MCU.

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

After the reset, the bundle files are used and the application should check if the new image works correctly (for example, connect to a network, run some network tests, and so forth).

On a successful system test, the application should check if the image is in WAIT_FOR_COMMIT mode, and call the OTA lib to perform the commit. After the commit, the files are used and a rollback is not allowed.

On system test failure, the application must reset the MCU to perform a rollback of all files to the pervious image:

```
_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 Sample OTA Application

The sample OTA application is implemented by a state machine (SM). The SM is started on response to sl_Start, and driven by NWP events and application events, as shown in Table 2 and Table 3.

**Table 2. OTA App States**

| OTA App States | Description |
|---|---|
| APP_STATE_STARTING | Waiting to INIT_COMPLETE event from the NWP (after calling sl_Start) |
| APP_STATE_WAIT_FOR_CONNECTION | Waiting to WLAN_CONNECTED event from the NWP - Connection runs if a profile is saved on the device. |
| APP_STATE_WAIT_FOR_IP | Waiting to IPV4_ACQUIRED event from the NWP |
| APP_STATE_PROVISIONING_IN_PROGRESS | Waiting for PROVISIONING_SUCCESS event. Provisioning starts if no profile, or connection failed |
| APP_STATE_PROVISIONING_WAIT_COMPLETE | Waiting for PROVISIONING_STOP event, provisioning stop indicate complete loop close of provisioning connection. |
| APP_STATE_PINGING_GW | This is the main application state. In this state, the application sends ping requests to the gateway.<br>Waiting to PING_COMPLETE event from the NWP and initiate another ping request (sl_NetAppPing) |
| APP_STATE_OTA_RUN | Keep Calling OtaRunStep and check return status<br>On download completion, reset the MCU to test the new image. |
| APP_STATE_ERROR | Fatal error state – halt |

**Table 3. NWP Events**

| NWP Event | Convert to OTA APP events |
|---|---|
| GeneralEvent | *SL_ERROR_LOADING_CERTIFICATE_STORE – Ignored(*)<br>Otherwise: SignalEvent(APP_EVENT_ERROR) |
| WlanEvent | SL_WLAN_EVENT_CONNECT: SignalEvent(APP_EVENT_CONNECTED)<br>SL_WLAN_EVENT_DISCONNECT: SignalEvent(APP_EVENT_DISCONNECTED)<br>Otherwise: SignalEvent(APP_EVENT_ERROR)<br>SL_WLAN_EVENT_PROVISIONING_STATUS:<br>According to status:<br>• SignalEvent(APP_EVENT_PROVISIONING_STARTED)<br>• SignalEvent(APP_EVENT_PROVISIONING_SUCCESS)<br>• SignalEvent(APP_EVENT_PROVISIONING_STOPPED);<br>• SignalEvent(APP_EVENT_ERROR) |
| FatalErrorEvent | SignalEvent(APP_EVENT_ERROR) |
| NetAppEvent | SL_NETAPP_EVENT_IPV4_ACQUIRED: SignalEvent(APP_EVENT_IP_ACQUIRED)<br>*SL_NETAPP_EVENT_IPV4_LOST: SignalEvent(APP_EVENT_DISCONNECT)<br>*SL_NETAPP_EVENT_DHCP_IPV4_ACQUIRE_TIMEOUT: SignalEvent(APP_EVENT_DISCONNECT)<br>Otherwise: SignalEvent(APP_EVENT_ERROR) |
| HttpServerEvent | SignalEvent(APP_EVENT_ERROR) |
| SockEvent | SL_SOCKET_TX_FAILED_EVENT: SignalEvent(APP_EVENT_RESTART)<br>Otherwise: SignalEvent(APP_EVENT_ERROR) |

## 5.1 Application State Machine

Figure 12 shows the sample application state machine.

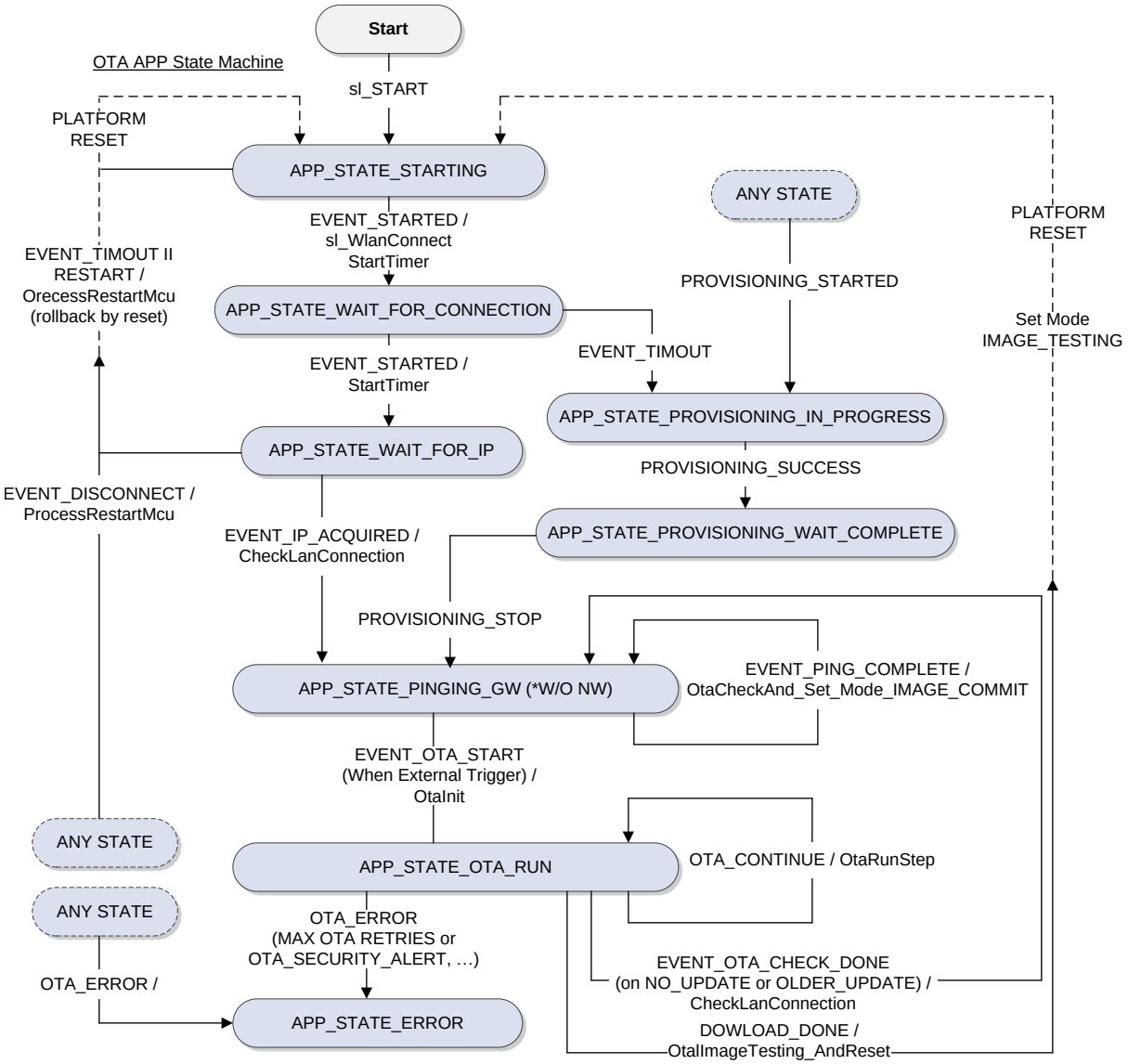


**Figure 12. Sample Application State Machine**

## 5.2 Provisioning Connection

The host application uses an existing profile to connect to an AP. If no profile exists, or it is unable to connect to any of the existing profile, the host application has the option to add a new profile using the TI provisioning process.

The provisioning process automatically starts if no connection is available. The user should use the Mobile Provisioning application to add a new profile.

More about provisioning can be found in the Provisioning Application Report.

More about the mobile application can be found in the Provisioning User Guide.

### 5.3 OTA External Trigger

After establishing a connection, the host application continues to check the connection to the AP endlessly (PING).

To start the OTA process, initiate an external trigger. The external trigger is defined according to the different platform used:

- CC3220 – Switch 2
- MSP432 – Switch P1.1

After the external trigger is initiated, the host application starts the OTA process.

## 6 Creating OTA Software Upgrade Package

The software upgrade package is an archive tar file (not compressed) that contains all relevant files. The files should be placed in hierarchy directories, as shown in Figure 13.

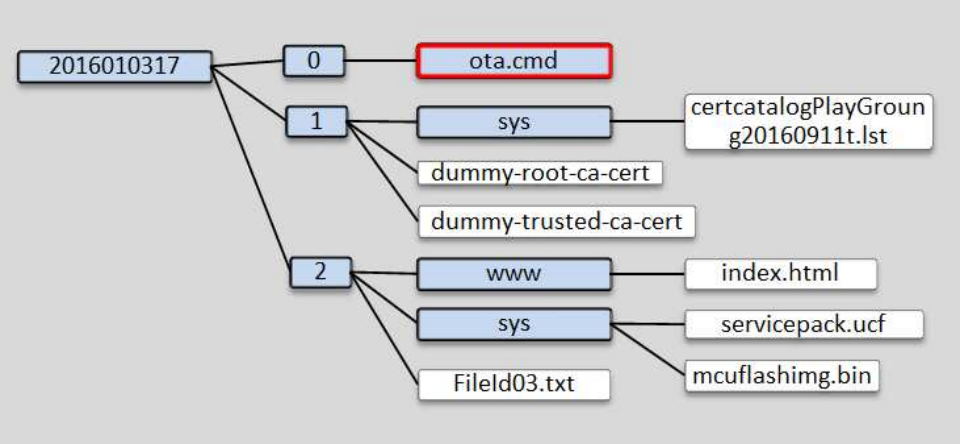


**Figure 13. Directory Hierarchy**

The name of the root directory should contain the package version number. This name should be constructed of numerical digits that can be compared to the existing package version number, and based on that to determine if an upgrade is required. The example above uses the date/time numeric number with the format: YYYYMMDDHH.

The directories 0, 1, and 2 are used to set the package files download order:

- Directory 0 – Contains package files of metadata named ota.cmd; each file in the secured package must have an entry in the metadata file with an optional signature and certificate file name. Files opened with default attributes (non-secured, fail-safe, bundle mode) should not have an entry in the metadata file.
- Directory 1 – Contains certificates that must be downloaded to the files that are using it.
- Directory 2 – Files using metadata from directory 0, certificates from directory 1, or other files.

The files and directory names should have the same name as in the serial flash. For files in the root directory, the leading slash is removed (certificate filename limitation).

The base tar file should be produced using the UniFlash tool and clicking Create OTA, as shown in Figure 14. This base tar file only has the MCU image and the service pack.

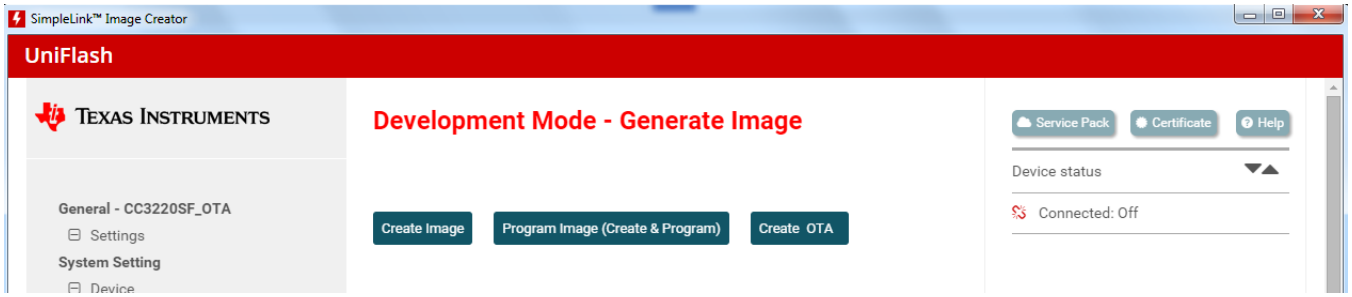


**Figure 14. Create OTA**

The user can add user files to the existing tar file by un-archiving the files into local directory, adding the user files, and archiving the directory into the tar file.

Limitations:

- Supports only TAR file types: 5-dir, 0-file
- The filename includes the full path (with subdirectory), and is limited to 100 bytes
- Certificate store (certstore.lst): MaxRequestSize = 7000, secured, fail safe, public write
- Service pack : For ROM(patches) : MaxRequestSize = 131072, fail safe
- Service pack : For TDFLASH : MaxRequestSize = 262144, no fail safe, secured, public write

# 7 Preparing ota.cmd Metadata File

The ota.cmd file is the first file in the OTA archive files, and it is in JSON format. Each JSON object represents one file in the archive file which is not used, and one file with the default attributes (fail-safe, non-secured, and bundle mode). Archived files with the default attributes do not need to have a JSON object in the ota.cmd. For non-bundle files (such as a large file with no place for mirroring), system integrity is not ensured.

The metadata supports the following JSON tokens:

- filename – The name of the file, the same name in both the TAR file and in the target SFLASH
- signature_base64 – The file signature in Base64 format, with a size of 345 bytes
- certificate – The certificate file name
- secured – 1 if the file is secured (default non-secured)
- maxsize – The maximum file size, relevant only for the creation of a new file (default is the actual file size)
- bundle – 1 if the file is part of the bundle (default bundle 1)

The following code example contains object for a secured file, FileId03.txt, with a signature and certificate. The second object is a service pack file with a signature.

```
[
    {
        "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 Distributing Software Upgrades Through a Cloud Service

## 8.1 Getting Started With the Dropbox™ Cloud Delivery Network (CDN)

The purpose of the following steps is to describe working with the Dropbox server to run the cloud OTA application. Before following the next instructions, you must create a Dropbox account.

1. On the main page, open the Developers tab, as shown in Figure 15.

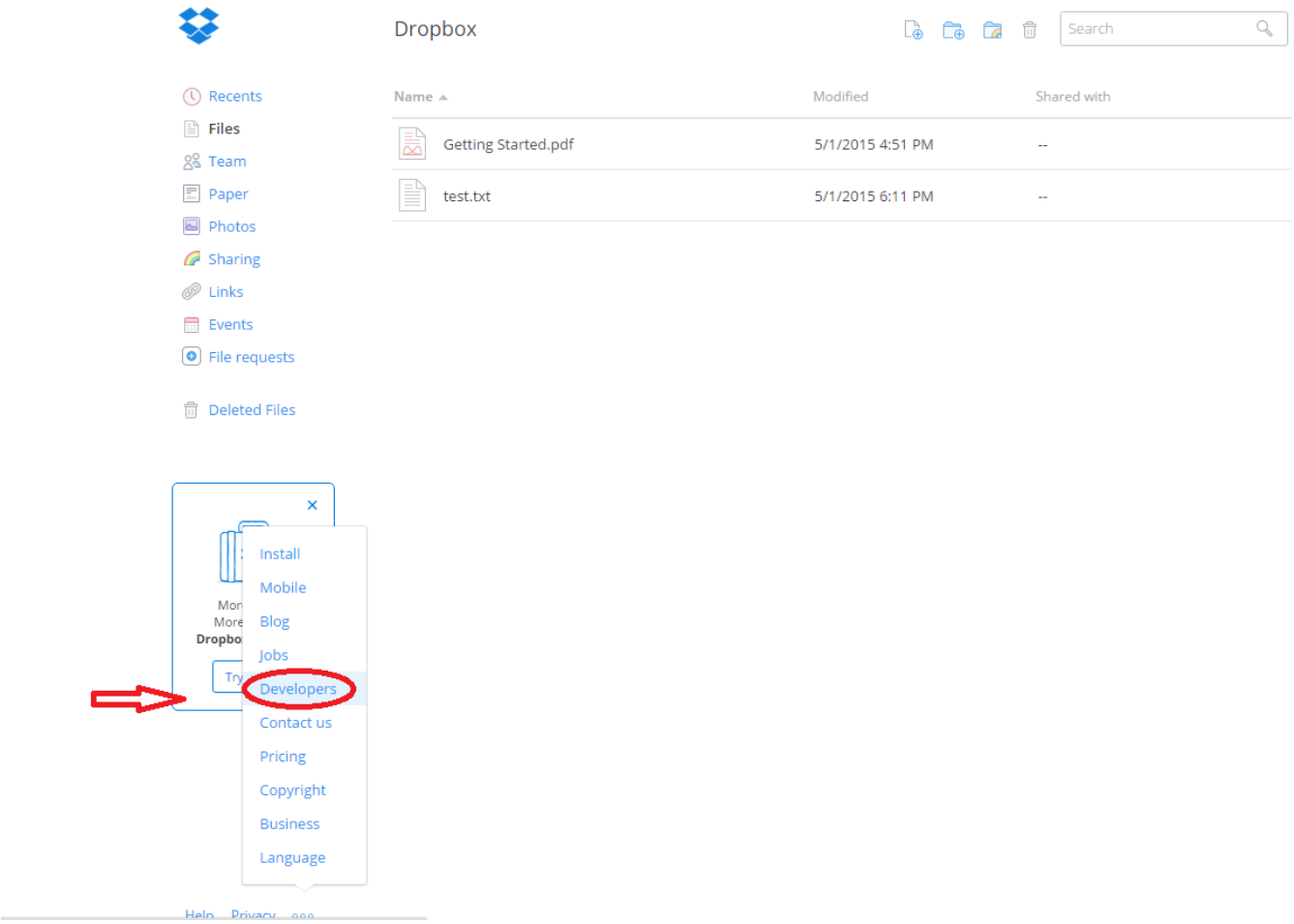


**Figure 15. Developers Tab**

2. Choose Create your app, as shown in Figure 16.

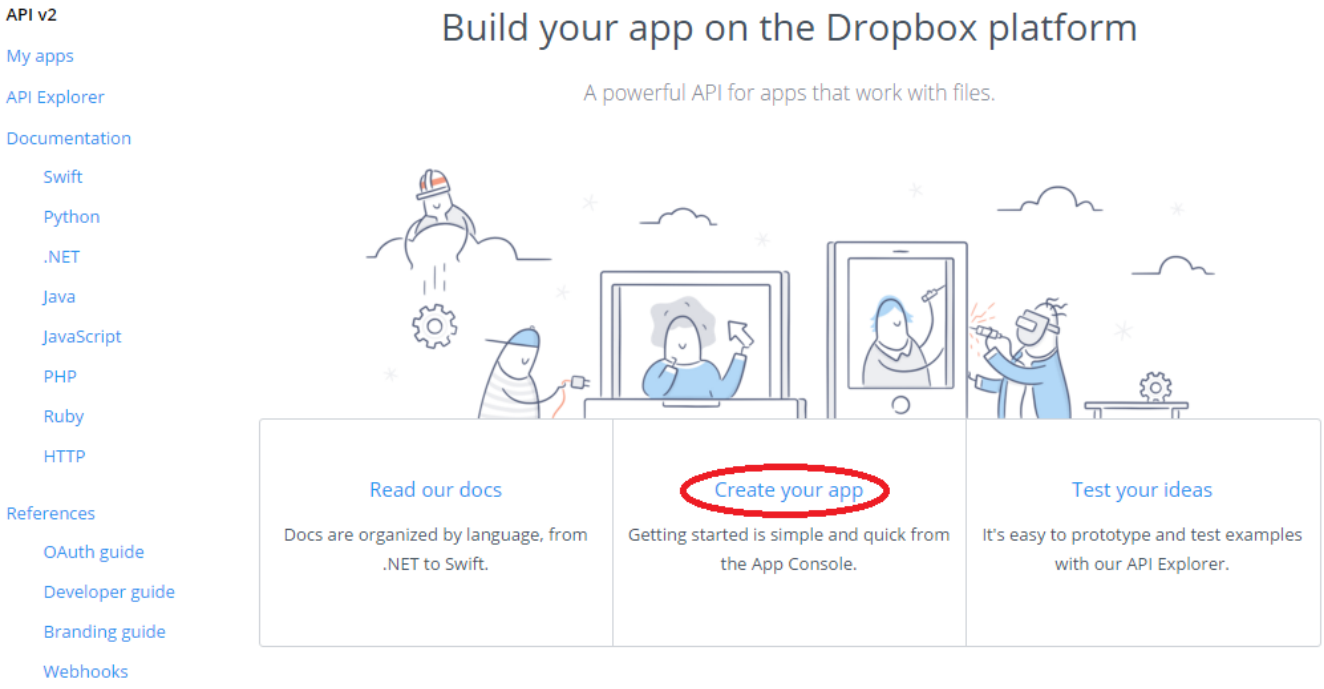


**Figure 16. Create Your App**

3. Choose Dropbox API. For access type, choose App folder, as shown in Figure 17, and name the application (for example, "OTA_R2"), then press Create app.

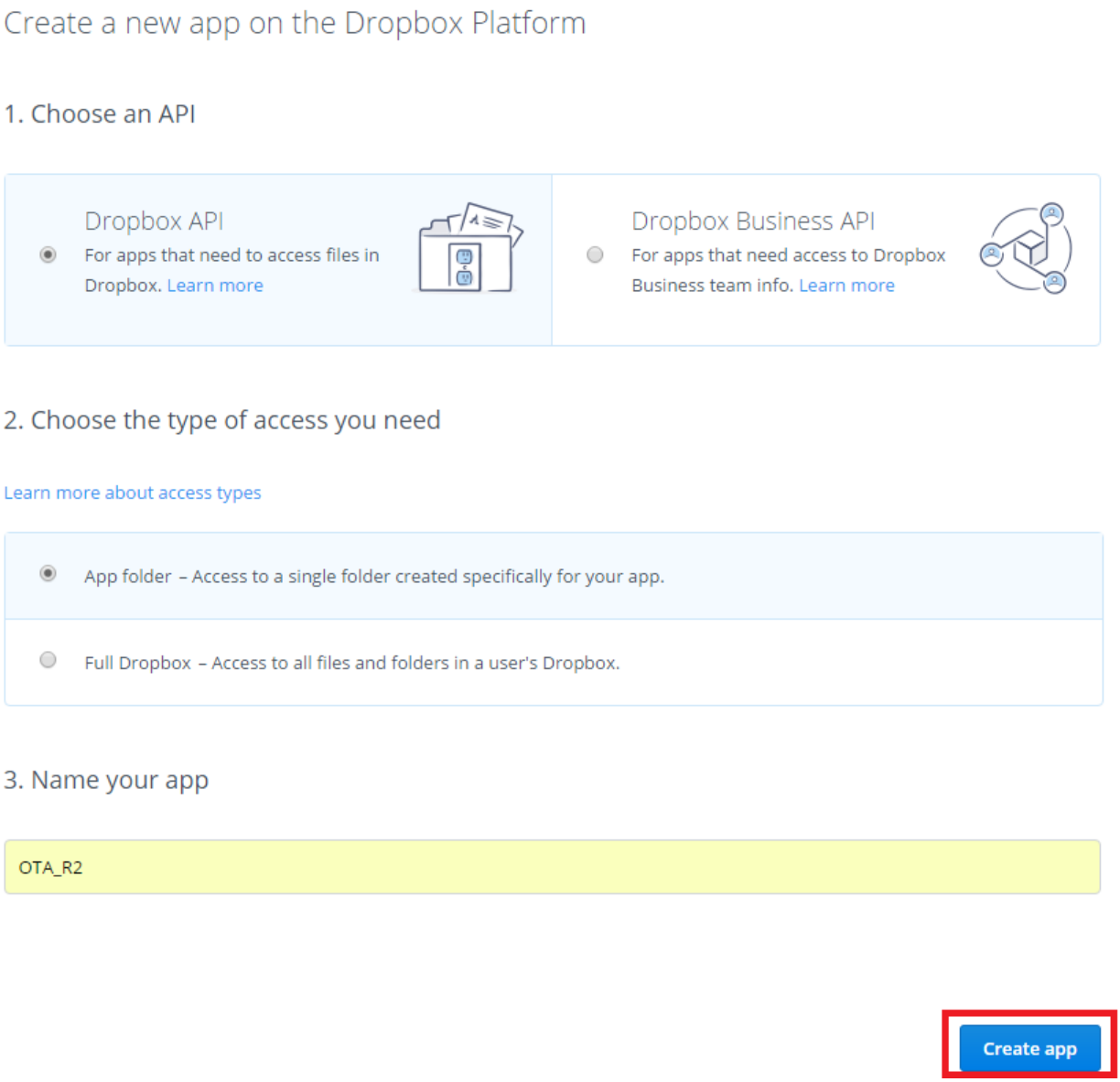


**Figure 17. Create App**

4. Select the App key and App secret, as shown in Figure 18.

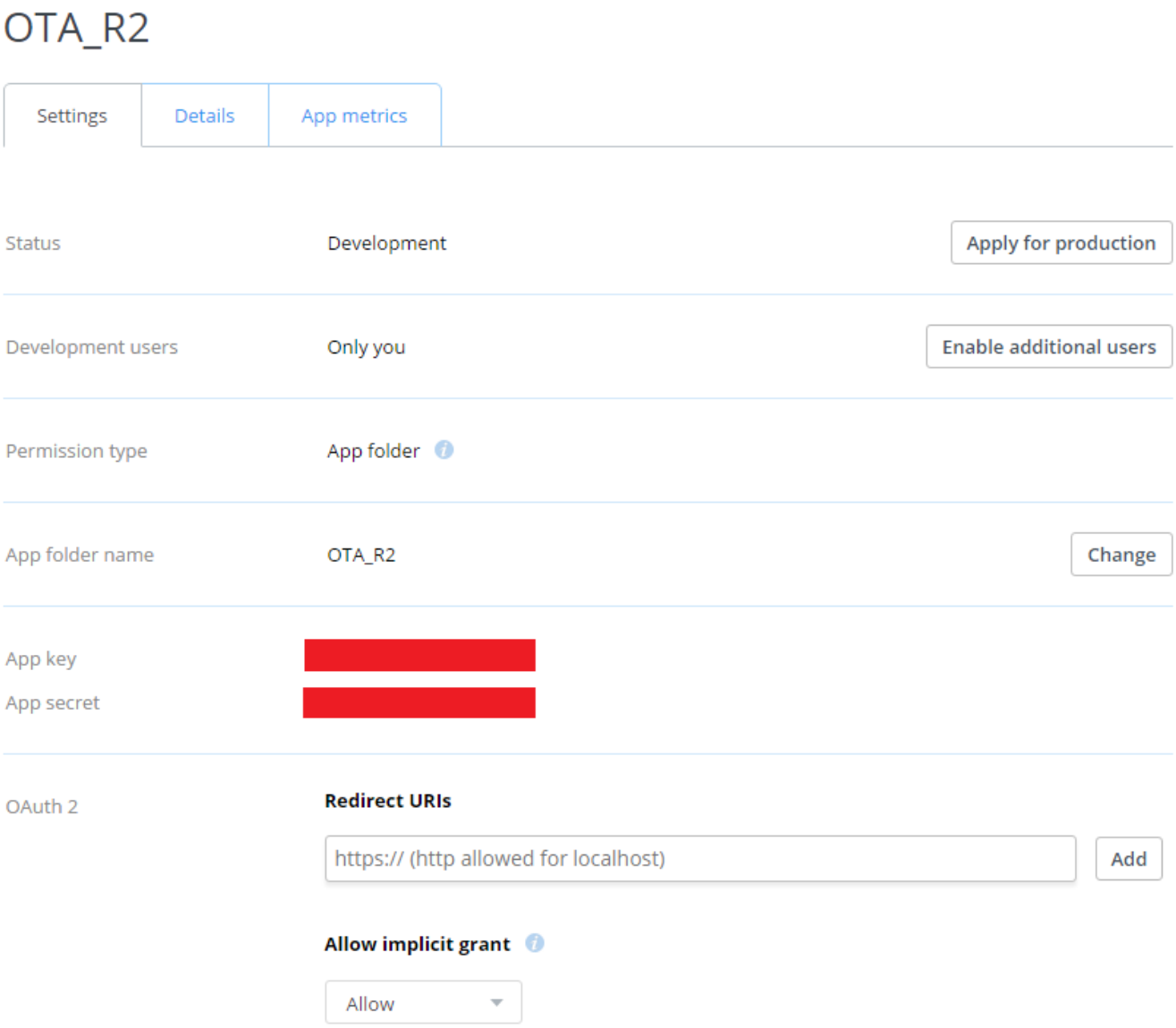


**Figure 18. App Key**

5. Generate the access token, as shown in Figure 19.

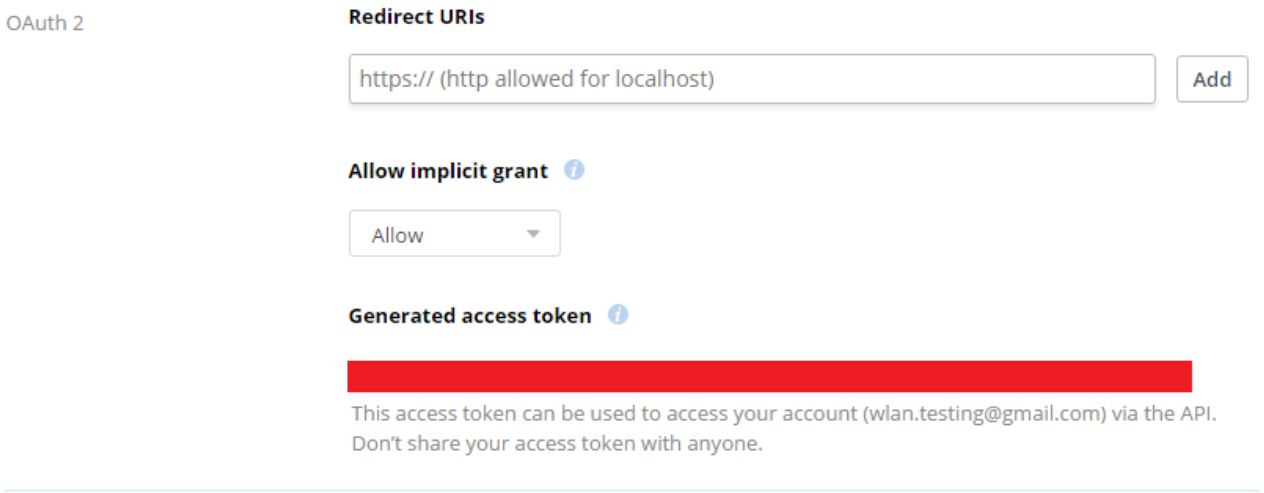


**Figure 19. Generate Access Token**

**NOTE:** This step-by-step guide shows how to use the Dropbox server based on developer rest client API-v1. For more information, read: https://www.dropbox.com/developers-v1/core/docs.

6. Authorize the app with the previously generated token, app key, and secret.
7. After the app is authorized, save the authorization bearer Token Key. This is the OTA Vendor Token which should be placed in the otauser.h file.
8. An Apps folder appears at the main page, as shown in Figure 20.

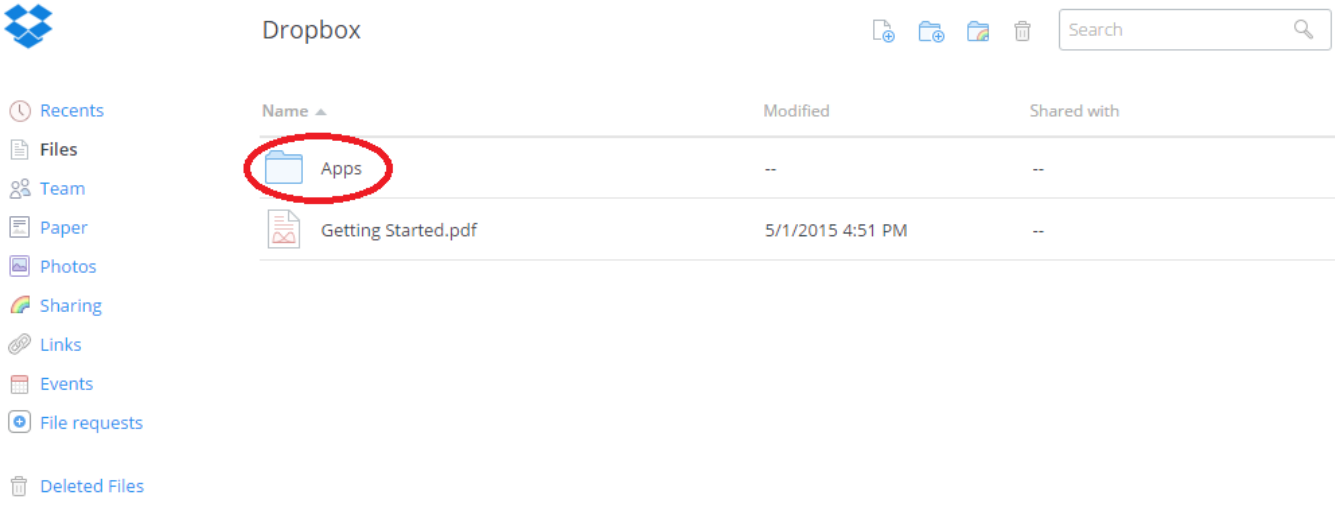


**Figure 20. Apps Folder**

9. Open Apps. The previously created folder should appear. Open the folder, as shown in Figure 21.

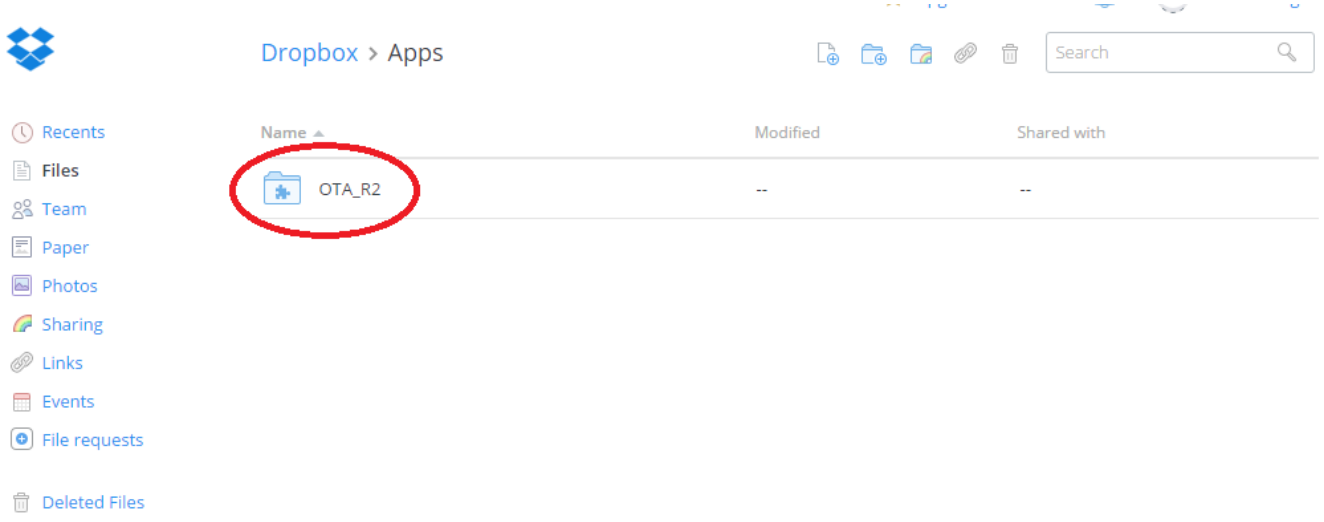


**Figure 21. OTA Vendor Directory**

10. Create new folder and name it with the same name which appears in the otauser.h file under the define OTA_VENDOR_DIR. Put the TAR file here. For information on how to create a TAR file, refer to Section 6.

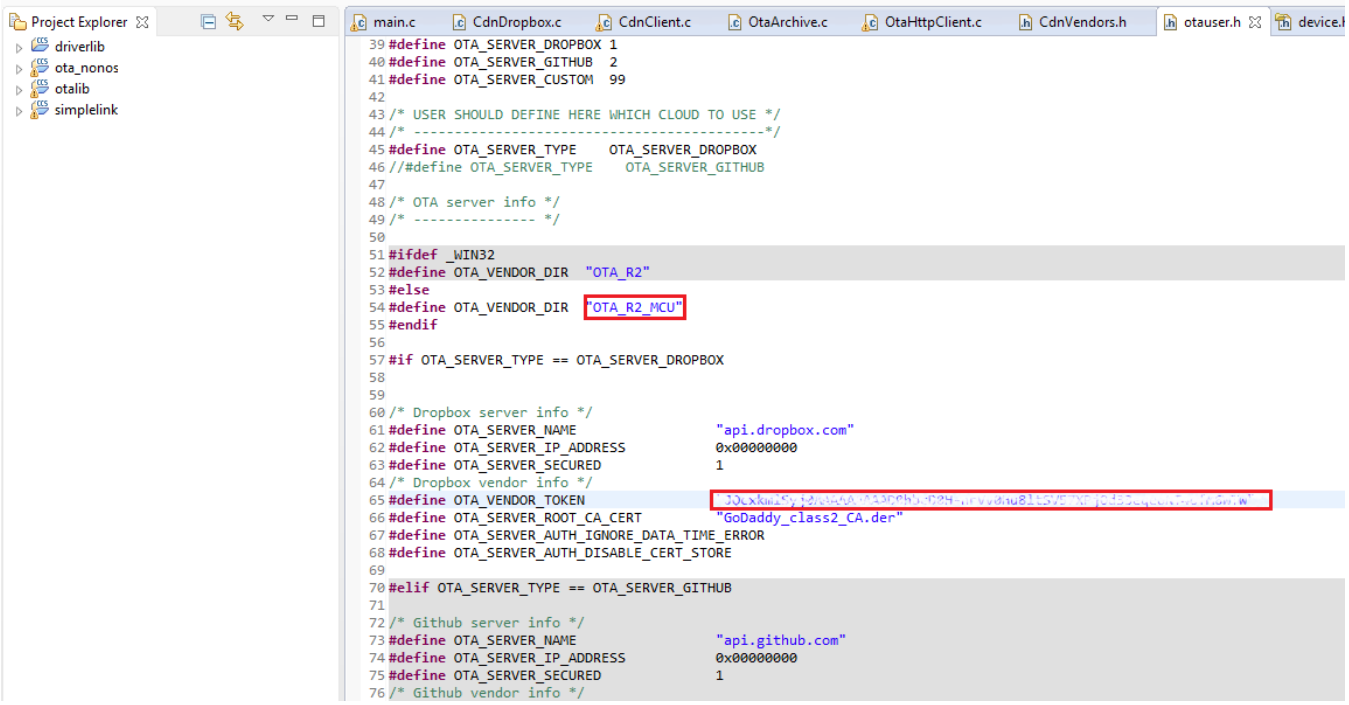


**Figure 22. Edit Marked Fields**

11. Build the cloud OTA project, and run.

## 8.2 GitHub CDN Getting Started

The following steps describe how to use GitHub CDN.

1. Create an account at gitHub.
2. Create new repository by pressing on "Create new..." (see Figure 23).

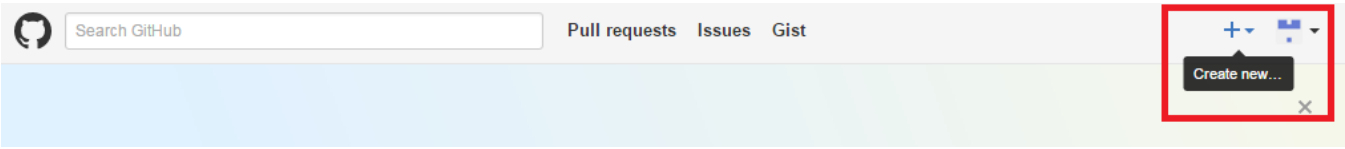


**Figure 23. Create New**

3. Choose New Repository (see Figure 24).

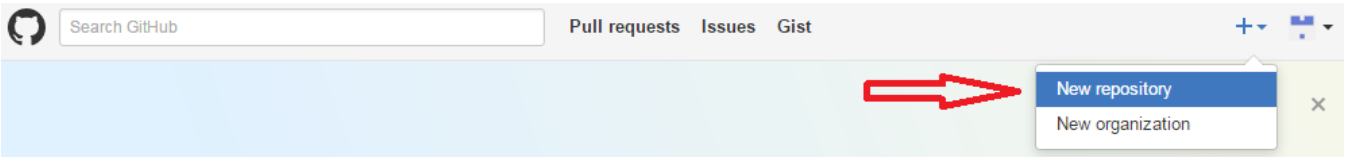


**Figure 24. New Repository**

4. Name the repository, select public or private access types, and press Create repository, as shown in Figure 25.

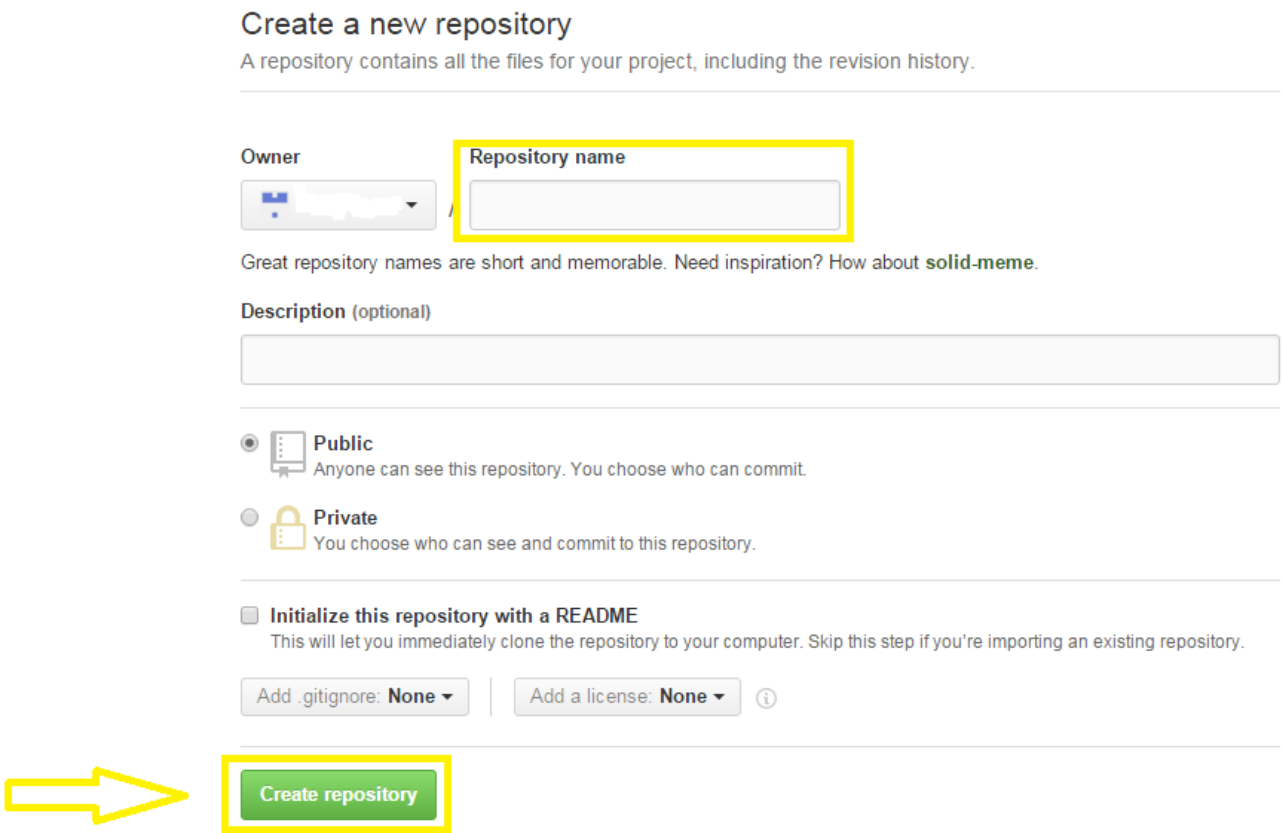


**Figure 25. Create New Repository**

5. Save the repository URL, as shown in Figure 26. An example would be: https://github.com/<username>/<repository name>.

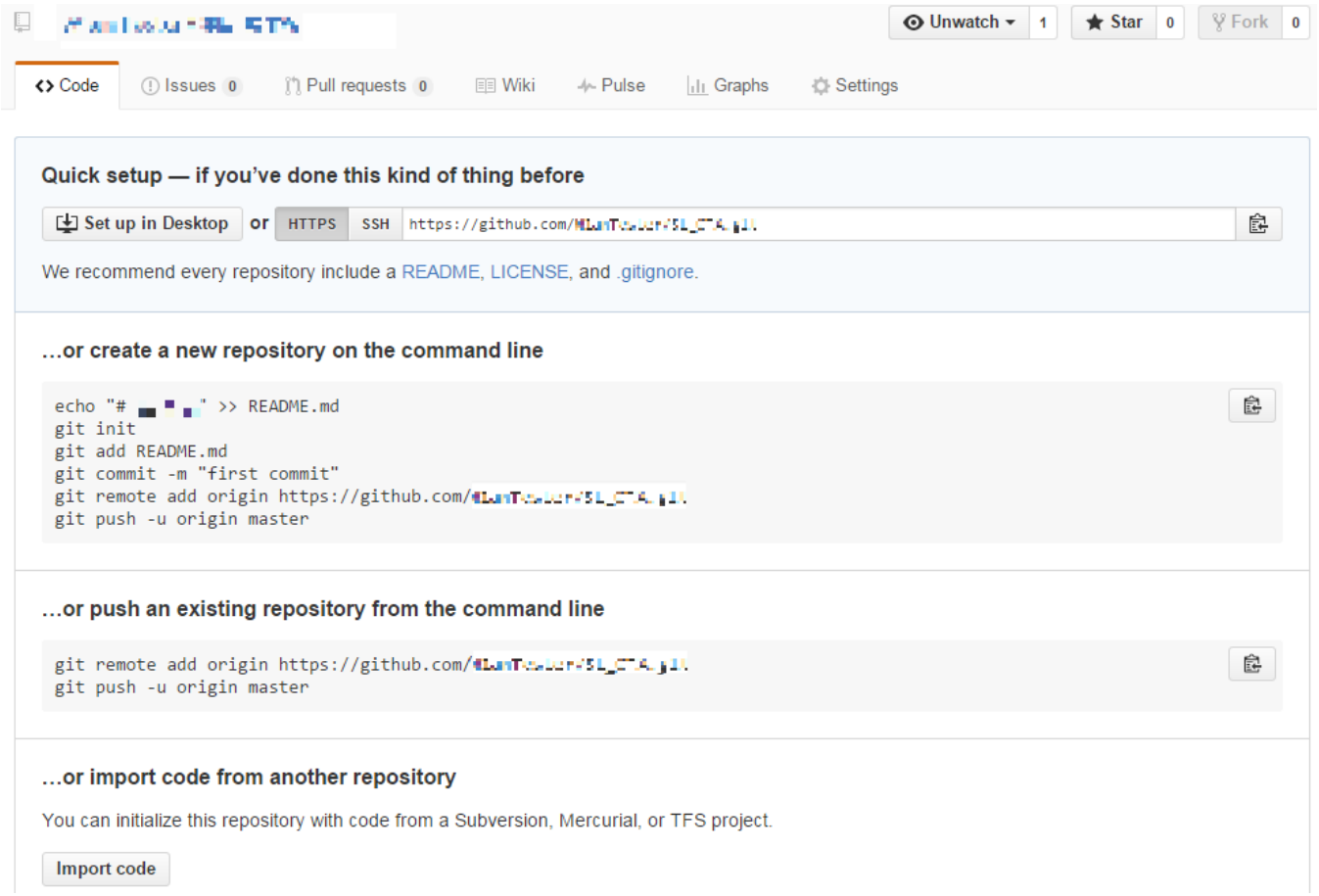


**Figure 26. Save Repository**

Following instructions shows how to activate a GitHub repository using Git GUI on the user's PC. Download the Git GUI from here: https://git-for-windows.github.io/).

6. Create a directory on the PC.

7. Right-click to open Git options, and choose Git Bash Here, as shown in Figure 27.

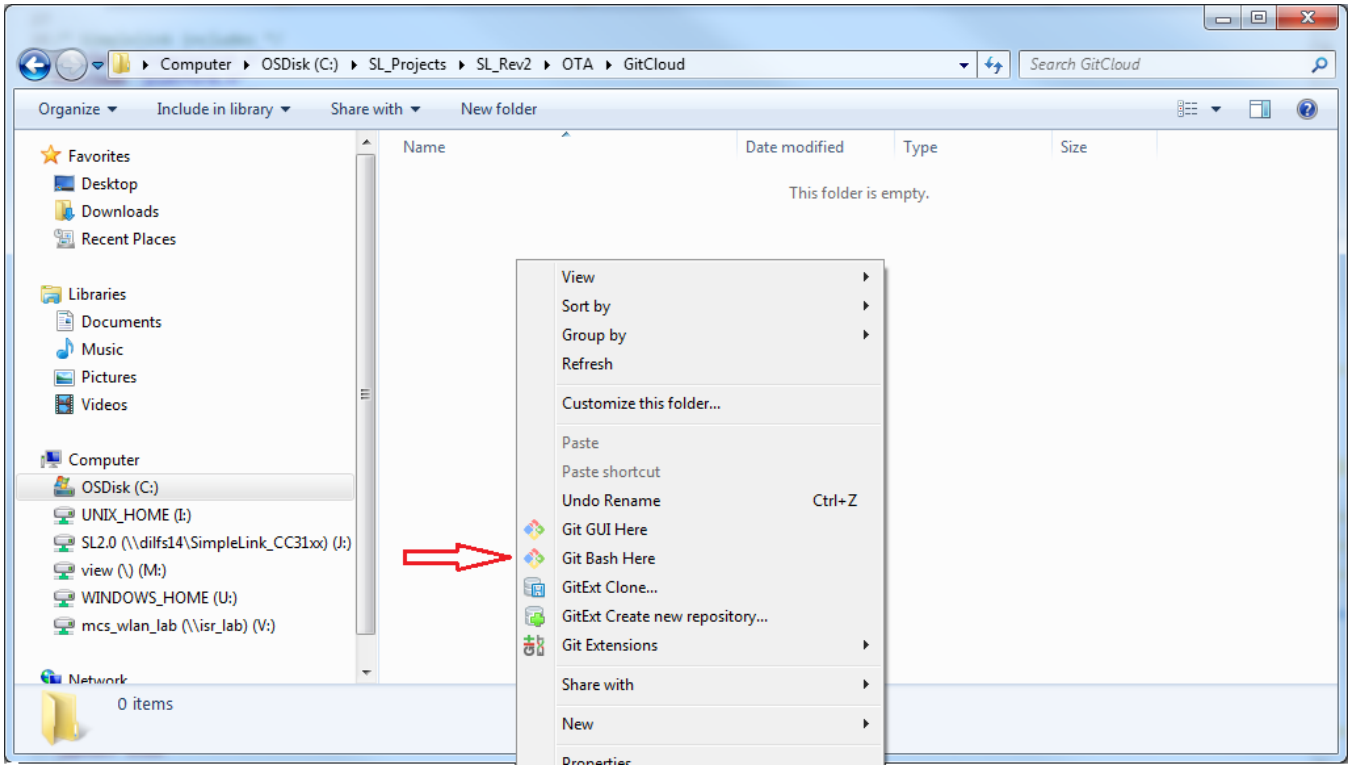


**Figure 27. Git Bash Here**

8. Type the following at the Git bash window:

```
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. Enter the username and password. Now the SP TAR file can be seen in the github.
10. After the GitHub CDN repository is activated, open a cloud OTA example application and edit the otauser.h file, as shown in Figure 28.
    - Uncomment 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
```

**Figure 28. Set Server Name in otauser.h**

    - Edit the fields shown in Figure 29.

```
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
```

**Figure 29. Set Directory in otauser.h**

11. Build the relevant cloud OTA application, and run.

## 9 Local Link Support

OTA on a local link is done from a local mobile device, not from the cloud. Part of the OTA library can be used to support OTA on a local link network. The application gets the archive file chunks using a NetApp API from a local mobile device, and uses only the OtaArchive and OtaJson modules to handle the archive files chunks, processes them, and stores them on the requested files in the NWP file system.

```
#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();
    }
```

After rebooting, check to commit the new image.

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

# 10 Support New CDN Vendor

The OTA lib supports two CDN vendors: Dropbox and Github. A vendor can use another CDN server. This chapter describes how to define this custom CDN.

## 10.1 otauser.h

Add the CDN to the list and define it as the selected server:

```
#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
```

Add custom vendors defines section:

```
#define OTA_VENDOR_DIR  "OTA_R2_MCU_FLASH"    /* for CC3220SF device      */

/* 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
```

The host application can skip the GetHostByName and define the server IP address in OTA_SERVER_IP_ADDRESS. If the custom server does not support server authentication and domain name verification, OTA_SERVER_ROOT_CA_CERT should be undefined.

## 10.2 ota/source/CdnVendors/Custom.c

Add the file and implement the following functions (see the example in Dropbox.c and in Github.c):

1. CdnCustom_SendReqDir – Send a directory tree request using the custom server name, the custom vendor directory, and the custom vendor token.
2. CdnCustom_ParseReqDir – Parse the directory tree reply from the custom server, and find for each file two tokens:
   - custom_file_name – The file name in the list
   - custom_file_size – The file size

   **NOTE:** The OTA lib uses only the first four files to search the TAR file.
3. CdnCustom_SendReqFileUrl – Send a file URL request to the custom server using the server name, the requested file name, and the custom server token.
4. CdnCustom_ParseRespFileUrl – Parse the file URL response from the custom server, and find the URL token:

   custom_file_url – The file URL

   **NOTE:** The OTA lib uses only the first four files.
5. CdnCustom_SendReqFileContent – Send a file content request from the custom file server.

### 10.3 ota/source/CdnVendors/CdnVendors.h

Add custom vendor macros:

```
#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
```

## Revision History

| Date | Revision | Notes |
|---|---|---|
| February 2017 | SWRA510* | Initial release |

## IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265