emDropbox

A client for Dropbox services

User Guide & Reference Manual

Document: UM15003
Software Version: 2.32
Revision: 0
Date: July 20, 2020

A product of SEGGER Microcontroller GmbH

www.segger.com

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel.	+49 2173-99312-0
Fax.	+49 2173-99312-28
E-mail:	support@segger.com[*]
Internet:	*www.segger.com*

[*]By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please report it to us and we will try to assist you as soon as possible.

Contact us for further information on topics or functions that are not yet documented.

Print date: July 20, 2020

Software	Revision	Date	By	Description
2.32	0	190510	PC	Chapter “API Reference” • Added `aModified` member to collected Dropbox metadata.
2.30	0	190301	PC	Update to latest software version.
2.20	0	180327	PC	Chapter “API Reference” • Added section “Preprocessor symbols”. • Added section “Data types”. • Added `IOT_DROPBOX_GetCopyrightText()`. • Added `IOT_DROPBOX_GetVersionText()`.
2.14	0	180214	PC	Aligned to public HTTP Client API.
2.12	0	171110	PC	Upgraded to use Dropbox API v2.
2.10	0	170208	PC	Initial release.
1.00	0	150725	PC	Internal release.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0--13--1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
`User Input`	Text entered at the keyboard by a user in a session transcript.
`Secret Input`	Text entered at the keyboard by a user, but not echoed (e.g. password entry), in a session transcript.
Reference	Reference to chapters, sections, tables and figures or other documents.
Emphasis	Very important sections.

Table of contents

Chapter 1

Introduction

This section presents an overview of emDropbox, its structure, and its capabilities.

1.1 What is emDropbox?

SEGGER emDropbox is a software library that enables you to create secure connections between a client and the Dropbox server and manage files in a Dropbox account.

SEGGER emDropbox is hardware independent and uses other components from emSSL and the SEGGER IoT Toolkit.

1.2 Design goals

SEGGER emDropbox is designed with the following goals in mind:

- Highly modular such that unused features are never linked into the application.
- Be completely runtime configurable, adding each modular feature as needed.
- Present a simple user-level API that is easy to use without extensive setup.
- Easy to maintain both by SEGGER and anybody with access to the sources.
- Conform to all necessary standards and current best practices.
- Be efficient both in terms of resource usage and execution speed.
- Target embedded processors with limited resources as well as workstations.
- Use the SEGGER Cryptographic Toolkit, the foundation of all SEGGER security products.

We believe all design goals are achieved by emDropbox.

1.3 Features

SEGGER emDropbox is written in ANSI C and can be used on virtually any CPU. Here is a list of emDropbox features:

- ISO/ANSI C source code.
- High performance.
- Small footprint.
- Runs “out-of-the-box”.
- Highly compact implementation runs effortlessly on single-chip MCUs.
- Easy-to-understand and simple-to-use API.
- Simple configuration.
- Secure communication over an open channel.
- Modular architecture links only what you need.
- Royalty-free.

1.4 Package content

SEGGER emDropbox is provided in source code and contains everything required. The following table shows the content of the emDropbox package:

Files	Description
`Config`	Configuration header files and I/O implementation.
`Doc`	emDropbox documentation.
`SEGGER`	SEGGER software component source code.
`IOT`	emDropbox implementation source code.
`Application`	emDropbox sample applications for embedded targets.
`Windows`	emDropbox sample applications for Windows.

Chapter 2

Exploring emDropbox

This section describes how to try out emDropbox on a PC and embedded hardware with minimal effort. We highly recommend that you try out a working version of emDropbox, shipped by SEGGER, with a known-good setup, preferably on an emPower board, before attempting to add it to your own application.

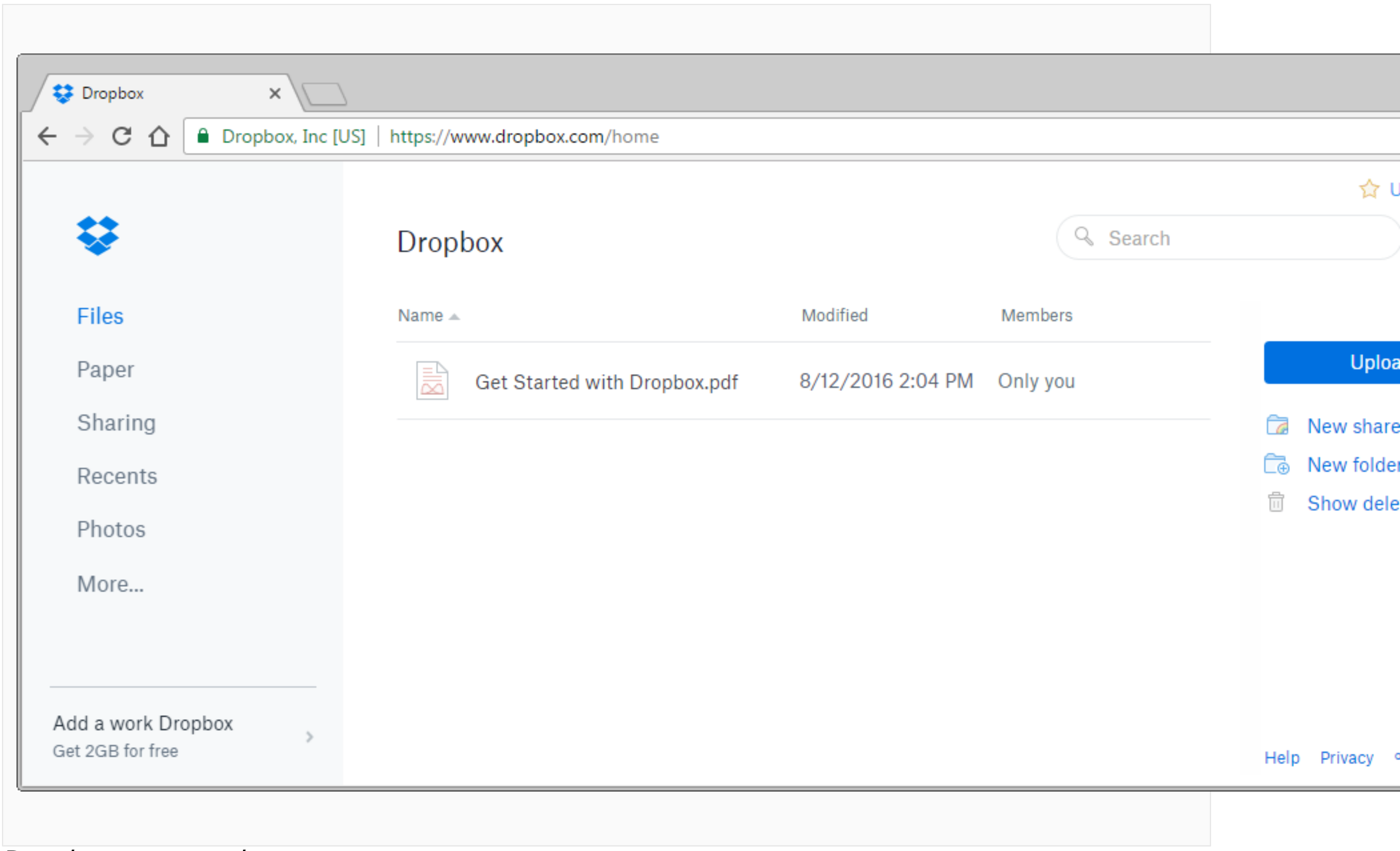
2.1 Setting up Dropbox

SEGGER emDropbox is shipped with a precompiled example that can interact with a user’s Dropbox. In order to do this, you must run through some setup. This setup will be used, by way of example, throughout the manual.

The following is a step-by-step guide showing you the preparation necessary to use a third-party Dropbox application with your Dropbox account. These steps are common for unofficial applications using the Dropbox API to access your account and are required to restrict access to your account to what you permit.

2.1.1 Get a Dropbox account

To evaluate emDropbox you need a Dropbox account. If you do not already have an account, you can register a free personal account at https://www.dropbox.com.

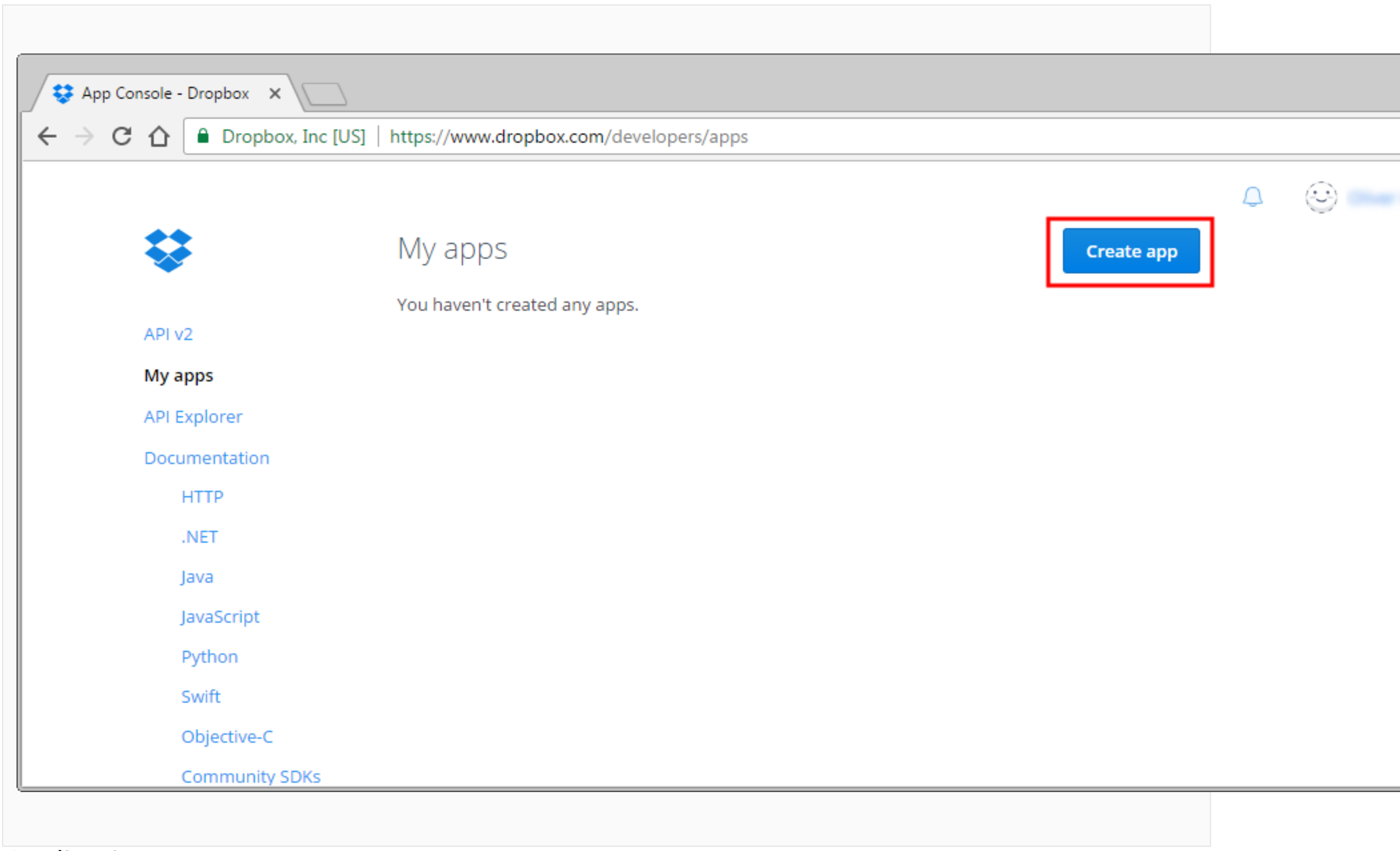


Name	Modified	Members
Get Started with Dropbox.pdf	8/12/2016 2:04 PM	Only you

Dropbox account home page

2.1.2 Register an application

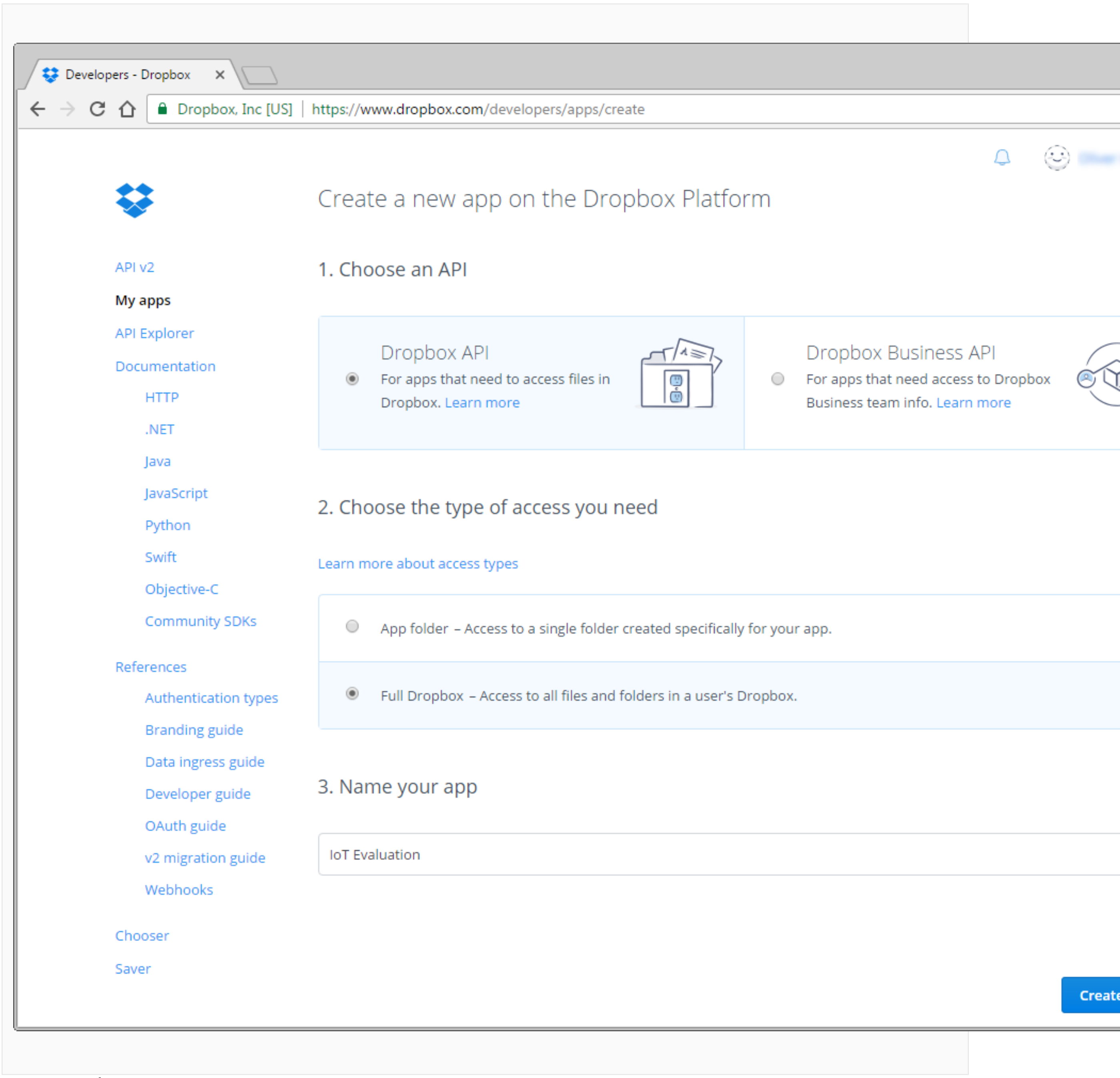
To grant the sample application access to your Dropbox account you need to supply it with an access token. For this, log in to your Dropbox account in your browser and visit the Dropbox developers page, https://www.dropbox.com/developers/apps. Click the **Create app** button to register a new application.



Application page

You will be asked three questions about the application:

- *Choose an API*: Select **Dropbox API**.
- *Choose the type of access you need*: You can grant the application full access to your Dropbox or access to a folder with the applications name only. emDropbox works with both.
- *Name your app*: Use whatever name you would like. Note, there are some restrictions such as not having “Dropbox” in the name.


Developers - Dropbox

Dropbox, Inc [US] https://www.dropbox.com/developers/apps/create

Create a new app on the Dropbox Platform

1. Choose an API

[✓ Dropbox API] For apps that need to access files in Dropbox. Learn more

Dropbox Business API For apps that need access to Dropbox Business team info. Learn more

2. Choose the type of access you need

Learn more about access types

App folder – Access to a single folder created specifically for your app.

[✓ Full Dropbox] – Access to all files and folders in a user's Dropbox.

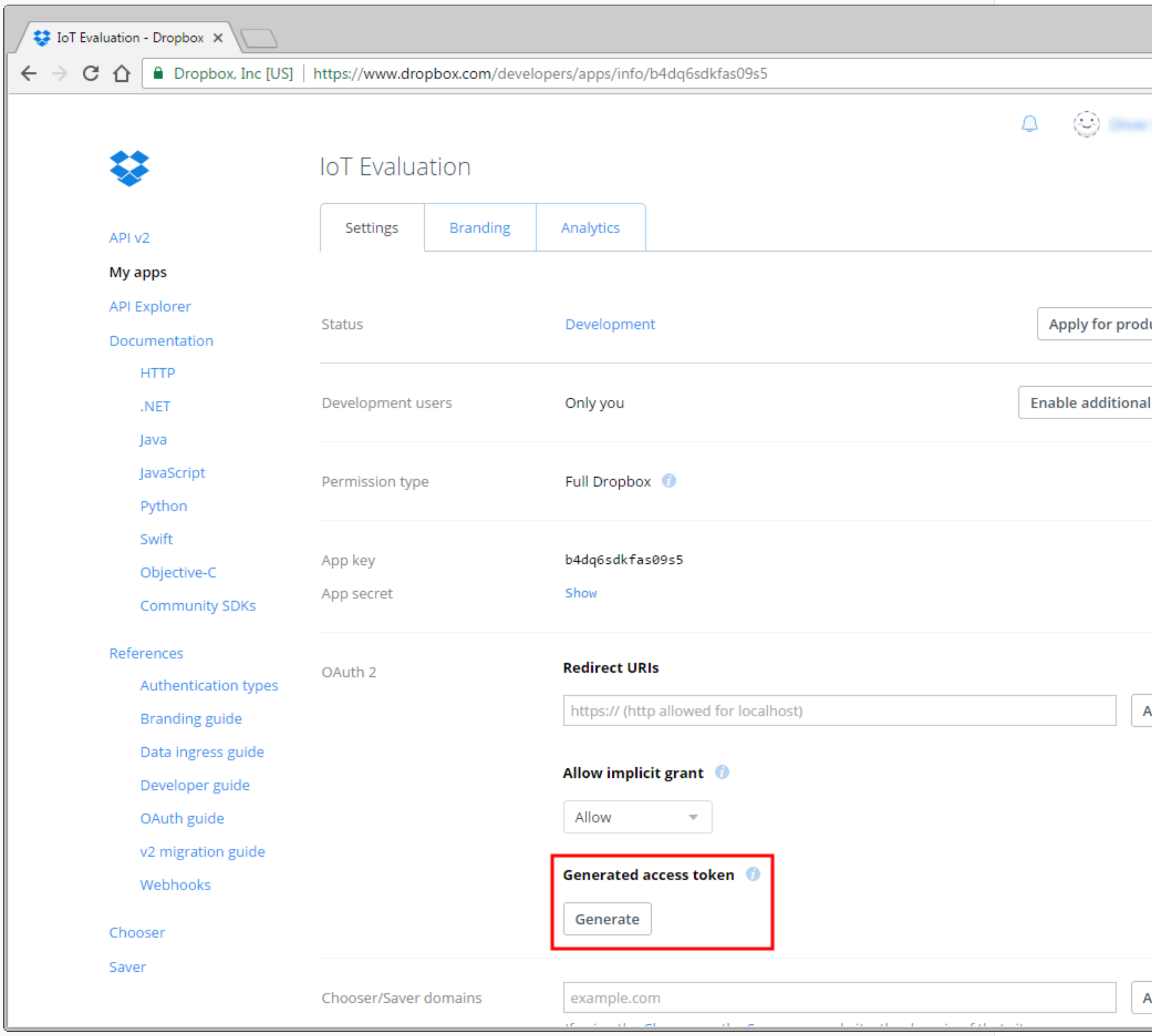
3. Name your app

IoT Evaluation

New application page

2.1.3 Generate an access token

On the page after clicking the **Create app** button, you will be able to generate an “OAuth 2” access token. Please click the **Generate** button to create the token.


IoT Evaluation - Dropbox

Dropbox, Inc [US] | https://www.dropbox.com/developers/apps/info/b4dq6sdkfas09s5

API v2
My apps
API Explorer
Documentation
HTTP
.NET
Java
JavaScript
Python
Swift
Objective-C
Community SDKs
References
Authentication types
Branding guide
Data ingress guide
Developer guide
OAuth guide
v2 migration guide
Webhooks
Chooser
Saver

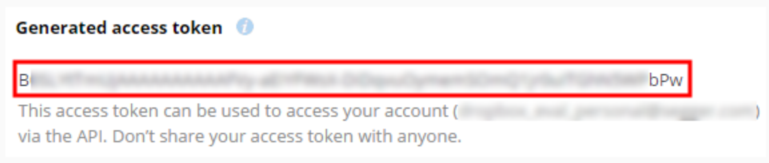
IoT Evaluation

Settings | Branding | Analytics

Status	Development	Apply for prod
Development users	Only you	Enable additional
Permission type	Full Dropbox	
App key	b4dq6sdkfas09s5	
App secret	Show	
OAuth 2	Redirect URIs: https:// (http allowed for localhost)	A
	Allow implicit grant: Allow	
	Generated access token: Generate	
Chooser/Saver domains	example.com	A

Generating an access token

The token is based on your current login credentials and becomes invalid if you change your credentials. This means that you must to regenerate the access token if you change your password.


Generated access token

B[redacted]bPw

This access token can be used to access your account ([redacted])
via the API. Don't share your access token with anyone.

Access token

2.2 Setting up the example Dropbox Commander

Once you have an access token you are ready to use the Dropbox Commander application that provides command-line access to your Dropbox account.

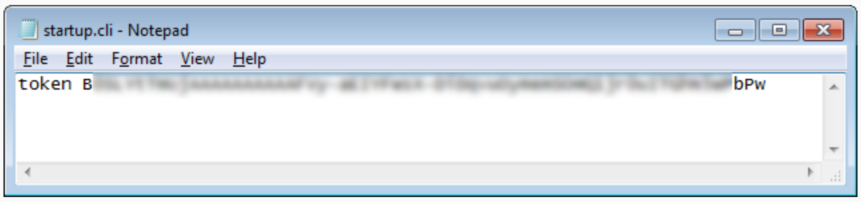
Setting up the access token for use with Dropbox Commander can be done in two ways.

2.2.1 Using the "token" command

Start Dropbox Command and use the command "`token` *x*" where "*x*" is to be replaced by your access token.

2.2.2 Using the "startup.cli" file

You can enter commands to be executed during startup of Dropbox Commander into the file `startup.cli` that resides in the same folder as Dropbox Commander. You can open the file with your preferred text editor and exchange the *x* for your access token, resulting in the `token` command being executed automatically on each start of Dropbox Commander.



```
token B[redacted]bPw
```

Startup file

2.3 Using Dropbox Commander

Dropbox Commander comes with several commands such as listing directory content, file management and file upload and download that allow you to operate on your Dropbox account.

You should be able to confirm access to your Dropbox by using the `dir` command to list the contents of your Dropbox.

```
C:> Dropbox.exe

(c) 2014-2018 SEGGER Microcontroller GmbH   www.segger.com
SEGGER Dropbox Commander V2.20 compiled Feb 23 2018 23:49:34

THIS UTILITY IS FREE.

This utility demonstrates that you can use SEGGER
software to access Dropbox content securely with emSSL
and any TCP/IP stack (such as embOS/IP) that supports
sockets.

For more information, contact info@segger.com.

Type "?" for a list of commands.

> ls
  9459751  Get Started With Dropbox.pdf
> put monologue.txt
Wrote 232 bytes
> ls
  9459751  Get Started With Dropbox.pdf
      232  monologue.txt
> cp monologue.txt batty.txt
> ls
      232  batty.txt
  9459751  Get Started With Dropbox.pdf
      232  monologue.txt
> get "Get Started With Dropbox.pdf"
Wrote 9459751 bytes
> _
```

A complete listing of Dropbox Commander is presented in *Dropbox Commander complete listing* on page 72.

Chapter 3

Using emDropbox

This section describes how to configure emDropbox for use and set up a shell connection using a sample project. The sample project can be customized and integrated into your application.

In this section we assume that you have a fully-functioning embOS/IP project, with emSSL, and that is able to connect to the network and all that is required is to add emDropbox to the project.

3.1 Sample applications

SEGGER emDropbox ships with a number of sample applications that demonstrate how to integrate Dropbox capability into your application. Each sample application shows a different aspect of emDropbox.

The sample applications are:

Application	Description
`IOT_DROPBOX_PutFile.c`	Upload file to Dropbox.
`IOT_DROPBOX_GetFile.c`	Download file from Dropbox.
`IOT_DROPBOX_Commander.c`	Full-featured Dropbox client.

3.1.1 A note on the samples

Each sample that we present in this section is written in a style that makes it easy to describe and that fits comfortably within the margins of printed paper. Therefore, it may well be that you would rewrite the sample to have a slightly different structure that fits better, but please keep in mind that these examples are written with clarity as the prime objective, and to that end we sacrifice some brevity and efficiency.

3.1.2 Where to find the sample code

All samples are included in the `Application` directory of the emDropbox distribution.

3.2 Uploading a file

To upload a file to Dropbox is straightforward. This example uploads content to the file `monologue.txt` in the account's root directory.

For a complete listing of this application, see *`IOT_DROPBOX_PutFile` complete listing* on page 28.

3.2.1 Application entry

The main application task is responsible for setting up the environment ready for emDropbox. This is simply boilerplate code that has no configuration:

```
void MainTask(void) {
  int Status;
  //
  SEGGER_SYS_Init();   ❶
  SEGGER_SYS_IP_Init();
  SSL_Init();   ❷
```

❶ Initialize system components

The calls to `SEGGER_SYS_Init()` and `SEGGER_SYS_IP_Init()` use the SEGGER system abstraction layer to initialize services to the application.

❷ Initialize SSL component

This initializes the SSL component. All Dropbox communication must be protected using TLS (also known as SSL). SEGGER emDropbox requires emSSL to provide the TLS connection, so please refer to the *emSSL User Guide & Reference Manual* for details on emSSL. The cipher suites and elliptic curves that are common to emSSL and the Dropbox server is covered in *Cipher suites and elliptic curves* on page 68.

3.2.2 Set up emDropbox

Once the system components are initialized, it's time to set up emDropbox.

```
IOT_DROPBOX_Init     (&_DbxCtx, _aJSONBuf, sizeof(_aJSONBuf));  ❶
IOT_DROPBOX_SetIO    (&_DbxCtx, &_IOAPI, &_SSLCtx);    ❷
IOT_DROPBOX_SetAPIKey(&_DbxCtx, "<your access token>");  ❸
```

❶ Initialize the client context

The call to `IOT_DROPBOX_Init()` sets up the emDropbox context and provides some working store. The working store is required to parse JSON responses sent by the Dropbox server and its size depends upon the maximum length of filenames that are stored in your Dropbox. In this case we have defined the buffer with a maximum of 260 characters:

```
static char _aJSONBuf[260];
```

This means that emDropbox will handle incoming names of up to 260 octets. Windows operating systems only allow file names and folder names of 260 characters or less, so this is an acceptable configuration value.

> **Note**
>
> As a filename can contain characters outside of the US ASCII 7-bit code, Dropbox encodes those characters as Unicode escapes in the JSON response. The underlying JSON parser converts those escapes into UTF-8 encoding, where a single character in the filename can expand to between two and five octets in the buffer.

❷ Initialize I/O interface

The call to `IOT_DROPBOX_SetIO()` sets up the connection and I/O API required for Dropbox connections. The interface is described in the section *I/O over SSL* on page 25, and we defer implementation details until later.

❸ Set secret API key (access token)

The call to `IOT_DROPBOX_SetAPIKey()` sets up the access token associated with the Dropbox account. Copy and paste the access token you generated from *Generate an access token* on page 17.

3.2.3 Upload file content

Once emDropbox context is set up for I/O and has the access token, everything is ready to work with your Dropbox. In this example we will upload some static content to the file `monologue.txt`. Here is the content to upload:

```
static const char _aContent[] =
  "I've seen things you people wouldn't believe.\n"
  "Attack ships on fire off the shoulder of Orion.\n"
  "I watched C-beams glitter in the dark near the Tannhauser Gate.\n"
  "All those moments will be lost in time, like tears in rain.\n"
  "Time to die.\n";
```

Uploading the content is as follows:

```
IOT_DROPBOX_SetFlag(&_DbxCtx, IOT_DROPBOX_FLAG_OVERWRITE);    ❶
Status = IOT_DROPBOX_PutBegin(&_DbxCtx, "monologue.txt", sizeof(_aContent));    ❷
if (Status >= 0) {
  Status = IOT_DROPBOX_PutContent(&_DbxCtx, _aContent, sizeof(_aContent));    ❸
  if (Status >= 0) {
    IOT_DROPBOX_PutEnd(&_DbxCtx);    ❹
  }
}
```

❶ Set operation flags

Dropbox provides a number of options when uploading a file that already exists:

- Consider it an error, return an appropriate error code and do not overwrite.
- Automatically create a new name for the file using a numeric suffix (e.g. `monologue (1).txt`) and upload to that.
- Overwrite the file.

In this instance the flag `IOT_DROPBOX_FLAG_OVERWRITE` is set on the emDropbox request which requsts that an existing be overwritten. The other flag is `IOT_DROPBOX_FLAG_AUTORENAME` which automatically renames any existing file of the same name. After initialization, no flags are set in the emDropbox context and it's considered an error to overwrite an existing file.

❷ Start the upload

Commencing the upload requires that not only is the file name provided, but also the payload size. The payload size is the uploaded file's size, which must be less than 150 MB. Unfortunately, the Dropbox API does not provide a way to stream an unbounded stream where the size is not known in advance, so the size of the file is *always* required.

❸ Upload data

If the upload is started successfully, the file content can be uploaded using `IOT_DROPBOX_PutContent()`. In this example we use a single call to upload the entire file content in one go.

Although Dropbox does not support unbounded content, it is possible to call `IOT_DROPBOX_PutContent()` multiple times to upload the bounded content in “chunks.” So, for instance, the above single-call upload can be rewritten to upload one byte at a time:

```
for (i = 0; Status >= 0 && i < sizeof(_aContext); ++i) {
  Status = IOT_DROPBOX_PutContent(&_DbxCtx, &_aContent[i], 1);
}
```

❹ Finish upload

If all preceding steps completed successfully, the call to `IOT_DROPBOX_PutEnd()` finishes off the upload and processes the response from the Dropbox server.

3.2.4 Close connection

Irrespective of whether an upload completes successfully or not, the connection to the Dropbox server must be closed to release resources:

```
IOT_DROPBOX_Exit(&_DbxCtx);
```

That’s all there is to uploading a file, so now it’s time to show how I/O is handled.

3.2.5 I/O over SSL

SEGGER emDropbox requires a set of methods that control how a TLS connection is set up and torn down, and also a set of methods that implement I/O over that secure connection after it is established.

The API is embodied in the `IOT_IO_API` type:

```
typedef struct {
  int (*pfConnect)   (void *pSession, const char *sHost, unsigned Port);
  int (*pfDisconnect)(void *pSession);
  int (*pfSend)      (void *pSession, const void *pData, unsigned DataLen);
  int (*pfRecv)      (void *pSession,       void *pData, unsigned DataLen);
} IOT_IO_API;
```

In this example the methods are implemented by functions local to the application:

```
static const IOT_IO_API _IOAPI = {
  _Connect,
  _Disconnect,
  _Send,
  _Recv
};
```

Here is the initialization of the emDropbox I/O again:

```
IOT_DROPBOX_SetIO(&_DbxCtx, &_IOAPI, &_SSLCtx);
```

Notice that the third argument passed into `IOT_DROPBOX_SetIO()` is the address of an SSL context: this context is passed to all `IOT_IO_API` methods as their first parameter, `pSession`.

3.2.5.1 Connection

Connection requires that a secure socket, protected by TLS, is opened to the Dropbox server.

```
static int _Connect(void *pVoid, const char *sHost, unsigned Port) {    ❶
  SSL_SESSION * pSession;
  int           Socket;
  int            Status;
  //
  pSession = pVoid;
  Status = SEGGER_SYS_IP_Open(sHost, Port);    ❷
  if (Status >= 0) {
    Socket = Status;
    SSL_SESSION_Prepare(pSession, Socket, &_IP_Transport);    ❸
    Status = SSL_SESSION_Connect(pSession, sHost);    ❹
    if (Status < 0) {
      SEGGER_SYS_IP_Close(Socket);
    }
  }
  return Status;
}
```

❶ **Receive connection parameters**

The implementation of the TLS setup requires that a TCP socket is opened to the given host and port specified by `sHost` (a domain name that must be resolved) and `Port` which is in host byte order.

The first parameter, `pSession`, is a pointer to an SSL session context, of type `SSL_SESSION`.

❷ **Open socket connection**

The incoming parameters are used to set up a TCP socket to the provided host and port.

❸ **Prepare to upgrade to secure**

The SSL session context is initialized and the underlying socket I/O is set up. Please refer to the *emSSL User Guide & Reference Manual* for further details relating to the SSL setup.

❹ **Upgrade socket to secure**

Once the session is initialized, emSSL is called to create a secure connection over the socket. If the handshake fails, the TCP socket is closed and the connection fails.

3.2.5.2 Disconnection

Disconnecting an established connection is handled by forwarding the incoming void pointer to `SSL_SESSION_Disconnect()` as an SSL session conext. Disconnection always succeeds and returns a success code.

```
static int _Disconnect(void *pVoid) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  SSL_SESSION_Disconnect(pSession);
  //
  return 0;
}
```

3.2.5.3 Send data

Sending data to an established connection is handled by forwarding the incoming void pointer to `SSL_SESSION_Send()` as an SSL session conext, along with the data to send.

```
static int _Send(void *pVoid, const void *pData, unsigned DataLen) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  return SSL_SESSION_Send(pSession, pData, DataLen);
}
```

3.2.5.4 Receive data

Receiving data from an established connection is handled by forwarding the incoming void pointer to `SSL_SESSION_Receive()` as an SSL session conext, along with the object to receive into.

```
static int _Recv(void *pVoid, void *pData, unsigned DataLen) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  return SSL_SESSION_Receive(pSession, pData, DataLen);
}
```

3.2.6 IOT_DROPBOX_PutFile complete listing

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : IOT_DROPBOX_PutFile.c
Purpose     : Demonstration of the SEGGER Dropbox API.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "IOT_Dropbox.h"
#include "SSL.h"
#include "SEGGER_SYS.h"

/*********************************************************************
*
*       Prototypes
*
**********************************************************************
*/

static int _Connect   (void *pVoid, const char *sHost, unsigned Port);
static int _Disconnect(void *pVoid);
static int _Send      (void *pVoid, const void *pData, unsigned DataLen);
static int _Recv      (void *pVoid, void *pData, unsigned DataLen);

/*********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const SSL_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
};

static const IOT_IO_API _IOAPI = {
  _Connect,
  _Disconnect,
  _Send,
  _Recv
};

static const char _aContent[] =
  "I've seen things you people wouldn't believe.\n"
  "Attack ships on fire off the shoulder of Orion.\n"
  "I watched C-beams glitter in the dark near the Tannhauser Gate.\n"
  "All those moments will be lost in time, like tears in rain.\n"
  "Time to die.\n";

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static SSL_SESSION          _SSLCtx;
static IOT_DROPBOX_CONTEXT _DbxCtx;
static char                 _aJSONBuf[260];
```

```
/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _Connect()
*
*  Function description
*    Connect to host using secure sockets.
*
*  Parameters
*    pVoid - Pointer to SSL session context.
*    sHost - Name of server we wish to connect to.
*    Port  - Port number in host byte order.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Connect(void *pVoid, const char *sHost, unsigned Port) {
  SSL_SESSION * pSession;
  int           Socket;
  int           Status;
  //
  pSession = pVoid;
  Status = SEGGER_SYS_IP_Open(sHost, Port);
  if (Status >= 0) {
    Socket = Status;
    SSL_SESSION_Prepare(pSession, Socket, &_IP_Transport);
    Status = SSL_SESSION_Connect(pSession, sHost);
    if (Status < 0) {
      SEGGER_SYS_IP_Close(Socket);
    }
  }
  return Status;
}

/*********************************************************************
*
*       _Disconnect()
*
*  Function description
*    Disconnect from host.
*
*  Parameters
*    pVoid - Pointer to SSL session context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Disconnect(void *pVoid) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  SSL_SESSION_Disconnect(pSession);
  //
  return 0;
}

/*********************************************************************
*
*       _Send()
*
*  Function description
*    Send data to host.
*
*  Parameters
*    pVoid   - Pointer to SSL session context.
*    pData   - Pointer to octet string to send over SSL.
*    DataLen - Octet length of the octet string to send.
*
```

```
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Send(void *pVoid, const void *pData, unsigned DataLen) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  return SSL_SESSION_Send(pSession, pData, DataLen);
}

/*********************************************************************
*
*       _Recv()
*
*  Function description
*    Receive data from host.
*
*  Parameters
*    pVoid   - Pointer to SSL session context.
*    pData   - Pointer to object that receives the data.
*    DataLen - Octet length of receiving object.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Recv(void *pVoid, void *pData, unsigned DataLen) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  return SSL_SESSION_Receive(pSession, pData, DataLen);
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void) {
  int Status;
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSL_Init();
  //
  IOT_DROPBOX_Init     (&_DbxCtx, _aJSONBuf, sizeof(_aJSONBuf));
  IOT_DROPBOX_SetIO    (&_DbxCtx, &_IOAPI, &_SSLCtx);
  IOT_DROPBOX_SetAPIKey(&_DbxCtx, "<your access token>");
  //
  IOT_DROPBOX_SetFlag(&_DbxCtx, IOT_DROPBOX_FLAG_OVERWRITE);
  Status = IOT_DROPBOX_PutBegin(&_DbxCtx, "monologue.txt", sizeof(_aContent));
  if (Status >= 0) {
    Status = IOT_DROPBOX_PutContent(&_DbxCtx, _aContent, sizeof(_aContent));
    if (Status >= 0) {
      IOT_DROPBOX_PutEnd(&_DbxCtx);
    }
  }
  //
  IOT_DROPBOX_Exit(&_DbxCtx);
  SSL_Exit();
  SEGGER_SYS_IP_Exit();
  SEGGER_SYS_Exit();
  //
  SEGGER_SYS_OS_Halt(Status);
}
```

```
/*************************** End of file ****************************/
```

3.3 Downloading a file

To download a file from Dropbox is straightforward. This example downloads the file `monologue.txt`, uploaded from the previous example, from the account's root directory.

For a complete listing of this application, see *`IOT_DROPBOX_GetFile` complete listing* on page 33.

Much of the code is identical to the previous example, so this section concentrates on the download.

3.3.1 Download file content

Downloading a file from Dropbox mirrors upload: there are steps that start, continue, and complete downloading.

```
Status = IOT_DROPBOX_GetBegin(&_DbxCtx, "Monologue.txt");    ❶
if (Status >= 0) {
  for (;;) {
    N = IOT_DROPBOX_GetContent(&_DbxCtx, aBuf, sizeof(aBuf)-1);    ❷
    if (N <= 0) {
      break;
    }
    aBuf[N] = 0;
    SEGGER_SYS_IO_Print(aBuf);
  }
  IOT_DROPBOX_GetEnd(&_DbxCtx);    ❸
}
```

❶ **Start the download**

The call to `IOT_DROPBOX_GetBegin()` starts downloading the file `monologue.txt` from the user's root directory. If the file is not found or cannot be opened for any reason, the return value is negative and can be decoded as an error from emDropbox or, alternatively, as an error from the SSL connection.

❷ **Download data**

Once the download has been started, successive calls to `IOT_DROPBOX_GetContent()` deliver the file content. Any error during download (for example a network error) is returned as a negative status; when all content has been delivered successfully, `IOT_DROPBOX_GetContent()` returns zero.

❸ **Finish upload**

If all preceding steps completed successfully, the call to `IOT_DROPBOX_GetEnd()` finishes off management of the download.

3.3.2 Close connection

Irrespective of whether a download completes successfully or not, the connection to the Dropbox server must be closed to release resources:

```
IOT_DROPBOX_Exit(&_DbxCtx);
```

That's all there is to downloading a file.

3.3.3 IOT_DROPBOX_GetFile complete listing

```
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : IOT_DROPBOX_GetFile.c
Purpose     : Demonstration of the SEGGER Dropbox API.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "IOT_Dropbox.h"
#include "SSL.h"
#include "SEGGER_SYS.h"

/*********************************************************************
*
*       Prototypes
*
**********************************************************************
*/

static int _Connect   (void *pVoid, const char *sHost, unsigned Port);
static int _Disconnect(void *pVoid);
static int _Send      (void *pVoid, const void *pData, unsigned DataLen);
static int _Recv      (void *pVoid, void *pData, unsigned DataLen);

/*********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const SSL_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  NULL
};

static const IOT_IO_API _IOAPI = {
  _Connect,
  _Disconnect,
  _Send,
  _Recv
};

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static SSL_SESSION          _SSLCtx;
static IOT_DROPBOX_CONTEXT _DbxCtx;
static char                 _aJSONBuf[260];

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/
```

```
/*********************************************************************
*
*       _Connect()
*
*  Function description
*    Connect to host using secure sockets.
*
*  Parameters
*    pVoid - Pointer to SSL session context.
*    sHost - Name of server we wish to connect to.
*    Port  - Port number in host byte order.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Connect(void *pVoid, const char *sHost, unsigned Port) {
  SSL_SESSION * pSession;
  int           Socket;
  int           Status;
  //
  pSession = pVoid;
  Status = SEGGER_SYS_IP_Open(sHost, Port);
  if (Status >= 0) {
    Socket = Status;
    SSL_SESSION_Prepare(pSession, Socket, &_IP_Transport);
    Status = SSL_SESSION_Connect(pSession, sHost);
    if (Status < 0) {
      SEGGER_SYS_IP_Close(Socket);
    }
  }
  return Status;
}

/*********************************************************************
*
*       _Disconnect()
*
*  Function description
*    Disconnect from host.
*
*  Parameters
*    pVoid - Pointer to SSL session context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Disconnect(void *pVoid) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  SSL_SESSION_Disconnect(pSession);
  //
  return 0;
}

/*********************************************************************
*
*       _Send()
*
*  Function description
*    Send data to host.
*
*  Parameters
*    pVoid   - Pointer to SSL session context.
*    pData   - Pointer to octet string to send over SSL.
*    DataLen - Octet length of the octet string to send.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Send(void *pVoid, const void *pData, unsigned DataLen) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  return SSL_SESSION_Send(pSession, pData, DataLen);
}

/*********************************************************************
*
*       _Recv()
*
*  Function description
*    Receive data from host.
*
*  Parameters
*    pVoid   - Pointer to SSL session context.
*    pData   - Pointer to object that receives the data.
*    DataLen - Octet length of receiving object.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Recv(void *pVoid, void *pData, unsigned DataLen) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  return SSL_SESSION_Receive(pSession, pData, DataLen);
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void);
void MainTask(void) {
  char aBuf[128];
  int  Status;
  int  N;
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSL_Init();
  //
  IOT_DROPBOX_Init     (&_DbxCtx, _aJSONBuf, sizeof(_aJSONBuf));
  IOT_DROPBOX_SetIO    (&_DbxCtx, &_IOAPI, &_SSLCtx);
  IOT_DROPBOX_SetAPIKey(&_DbxCtx, "<your access token>");
  //
  Status = IOT_DROPBOX_GetBegin(&_DbxCtx, "monologue.txt");
  if (Status >= 0) {
    for (;;) {
      N = IOT_DROPBOX_GetContent(&_DbxCtx, aBuf, sizeof(aBuf)-1);
      if (N <= 0) {
        break;
      }
      aBuf[N] = 0;
      SEGGER_SYS_IO_Print(aBuf);
    }
    IOT_DROPBOX_GetEnd(&_DbxCtx);
  }
  //
  IOT_DROPBOX_Exit(&_DbxCtx);
  SSL_Exit();
  SEGGER_SYS_IP_Exit();
  SEGGER_SYS_Exit();
  //
  SEGGER_SYS_OS_Halt(Status);
}
```

```
/*************************** End of file ****************************/
```

3.4 Adding emDropbox to your project

In this section we assume that you have a fully-functioning embOS/IP project, with emSSL, and that is able to connect to the network and all that is required is to add emDropbox to the project. You can use the sample “start” projects as a reference when setting up your own application.

❶ Set up include directories

You should make sure that the include path contains the following directories (the order of inclusion is of no importance):

- `Config`
- `CRYPTO`
- `IOT`
- `SSL`
- `SEGGER`

The contents of `CRYPTO` and `SSL` are part of the emSSL distribution and are not included in the emDropbox add-on.

Note

Always make sure that you have only one version of each file!

It is frequently a major problem when updating to a new version of emDropbox if you have old files included and therefore mix different versions. If you keep Dopbox Client in the directories as suggested (and only in these), this type of problem cannot occur. When updating to a newer version, you should be able to keep your configuration files and leave them unchanged. For safety reasons, we recommend backing up (or at least renaming) the existing directories before updating.

❷ Add source files

Add the source code files that you find in the shipment to your project. The `SEGGER` and `CRYPTO` folders are shared components.

❸ Initialize emSSL

You initialize emSSL using `SSL_Init()`. You must call `SSL_Init()` before using any other emSSL API function and before using emDropbox.

With these three steps, emDropbox is installed and ready to run.

Chapter 4

API reference

This section summarizes the API functions of emDropbox. The Dropbox Client API is kept as simple as possible to provide a straightforward way for integration into a product.

4.1 Preprocessor symbols

4.1.1 Dropbox client errors

Description

Errors that the Dropbox Client can generate.

Definition

```
#define IOT_DROPBOX_STATUS_BAD_INPUT_PARAMETER          -900
#define IOT_DROPBOX_STATUS_BAD_OR_EXPIRED_TOKEN         -901
#define IOT_DROPBOX_STATUS_BAD_OAUTH_REQUEST            -902
#define IOT_DROPBOX_STATUS_FILE_OR_FOLDER_NOT_FOUND     -903
#define IOT_DROPBOX_STATUS_REQUEST_METHOD_NOT_EXPECTED  -904
#define IOT_DROPBOX_STATUS_NOT_ACCEPTABLE               -905
#define IOT_DROPBOX_STATUS_RATE_LIMITED                 -906
#define IOT_DROPBOX_STATUS_STORAGE_QUOTA_EXCEEDED       -907
#define IOT_DROPBOX_STATUS_SERVER_ERROR                 -908
#define IOT_DROPBOX_STATUS_BAD_IMAGE                    -909
#define IOT_DROPBOX_STATUS_CHUNKING_UNSUPPORTED         -910
#define IOT_DROPBOX_STATUS_OTHER_ERROR                  -911
#define IOT_DROPBOX_STATUS_NOT_FILE                     -912
#define IOT_DROPBOX_STATUS_NOT_FOLDER                   -913
#define IOT_DROPBOX_STATUS_RESTRICTED_CONTENT           -914
```

Symbols

Definition	Description
IOT_DROPBOX_STATUS_BAD_INPUT_PARAMETER	Dropbox response: 400
IOT_DROPBOX_STATUS_BAD_OR_EXPIRED_TOKEN	Dropbox response: 401
IOT_DROPBOX_STATUS_BAD_OAUTH_REQUEST	Dropbox response: 403
IOT_DROPBOX_STATUS_FILE_OR_FOLDER_NOT_FOUND	Dropbox response: 404 or path/not_found
IOT_DROPBOX_STATUS_REQUEST_METHOD_NOT_EXPECTED	Dropbox response: 405
IOT_DROPBOX_STATUS_NOT_ACCEPTABLE	Dropbox response: 406, 409, 429
IOT_DROPBOX_STATUS_RATE_LIMITED	Dropbox response: 503
IOT_DROPBOX_STATUS_STORAGE_QUOTA_EXCEEDED	Dropbox response: 507
IOT_DROPBOX_STATUS_SERVER_ERROR	Dropbox response: 5xx not otherwise covered
IOT_DROPBOX_STATUS_BAD_IMAGE	Dropbox response: 415
IOT_DROPBOX_STATUS_CHUNKING_UNSUPPORTED	Dropbox response: 411
IOT_DROPBOX_STATUS_OTHER_ERROR	Dropbox response: any response not otherwise covered
IOT_DROPBOX_STATUS_NOT_FILE	path/not_file
IOT_DROPBOX_STATUS_NOT_FOLDER	path/not_folder
IOT_DROPBOX_STATUS_RESTRICTED_CONTENT	path/restricted_content

Additional information

Note that lower-level errors originating from the underlying socket or HTTP Client (which the Dropbox Client uses as a service) are also possible as errors returned by the API.

The Dropbox REST API responses are converted to Dropbox Client errors and the table indicates the Dropbox response that corresponds to the Dropbox Client error.

4.1.2 Dropbox request flags

Description

Flags that a Dropbox request can honor.

Definition

```
#define IOT_DROPBOX_FLAG_MUTE           0
#define IOT_DROPBOX_FLAG_OVERWRITE      1
#define IOT_DROPBOX_FLAG_AUTORENAME     2
```

Symbols

Definition	Description
IOT_DROPBOX_FLAG_MUTE	Unused, deprecated by Dropbox REST API v2.
IOT_DROPBOX_FLAG_OVERWRITE	Overwrite file on upload if it exists.
IOT_DROPBOX_FLAG_AUTORENAME	Auto-rename file on conflict.

See also

`IOT_DROPBOX_SetFlag()`

4.2 Data types

4.2.1 IOT_DROPBOX_METADATA

Description

Metadata extracted from a metadata query.

Type definition

```
typedef struct {
  char   aTag[];
  char   aPath[];
  U64    Size;
  U8     IsFolder;
  U8     IsDeleted;
  U8     aHash[];
  U64    Version;
  char   aModified[];
  int    Valid;
} IOT_DROPBOX_METADATA;
```

Structure members

Member	Description
`aTag`	Tag, field="`.tag`"
`aPath`	Canonical file path, field="name"
`Size`	File size in bytes, field="size"
`IsFolder`	Nonzero for folders, derived, ".tag = folder"
`IsDeleted`	Nonzero for deleted files, field="`is_deleted`"
`aHash`	File digest, binary, field="hash"
`Version`	File version, field="ver"
`aModified`	Timestamp of last modification, field="`client_modified`" [note: Dropbox provides this only for files]
`Valid`	Nonzero when members contain valid data

See also

`IOT_DROPBOX_GetMetadata()`

4.2.2 IOT_DROPBOX_METADATA_ENUM_FUNC

Description

Callback invoked during metadata enumeration.

Type definition

```
typedef void (IOT_DROPBOX_METADATA_ENUM_FUNC)(const IOT_DROPBOX_METADATA * pMeta);
```

Parameters

Parameter	Description
pMeta	Pointer to parsed metadata.

See also

`IOT_DROPBOX_GetMetadata()`

4.3 Information functions

The table below lists the functions that return emDropbox information.

Function	Description
`IOT_DROPBOX_GetVersionText()`	Get Dropbox Client version as printable string.
`IOT_DROPBOX_GetCopyrightText()`	Get Dropbox Client copyright as printable string.

4.3.1 IOT_DROPBOX_GetVersionText()

Description

Get Dropbox Client version as printable string.

Prototype

```
char *IOT_DROPBOX_GetVersionText(void);
```

Return value

Zero-terminated version string.

4.3.2 IOT_DROPBOX_GetCopyrightText()

Description

Get Dropbox Client copyright as printable string.

Prototype

```
char *IOT_DROPBOX_GetCopyrightText(void);
```

Return value

Zero-terminated copyright string.

4.4 Configuration functions

The table below lists the functions provided by the emDropbox API. Detailed description of each function is found in the sections that follow.

Function	Description
`IOT_DROPBOX_Init()`	Initialize a Dropbox client context.
`IOT_DROPBOX_Exit()`	Disconnect Dropbox client from server.
`IOT_DROPBOX_SetIO()`	Set HTTP transport I/O.
`IOT_DROPBOX_SetAPIKey()`	Set API key for user's Dropbox.
`IOT_DROPBOX_SetFlag()`	Set client context flag.
`IOT_DROPBOX_ClrFlag()`	Clear client context flag.

4.4.1 IOT_DROPBOX_ClrFlag()

Description

Clear client context flag.

Prototype

```
void IOT_DROPBOX_ClrFlag(IOT_DROPBOX_CONTEXT * pSelf,
                         unsigned              Flag);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
Flag	Flag to clear.

4.4.2 IOT_DROPBOX_Exit()

Description

Disconnect Dropbox client from server.

Prototype

```
void IOT_DROPBOX_Exit(IOT_DROPBOX_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.

4.4.3 IOT_DROPBOX_Init()

Description

Initialize a Dropbox client context.

Prototype

```
void IOT_DROPBOX_Init(IOT_DROPBOX_CONTEXT * pSelf,
                      char                * pJSONBuf,
                      unsigned              JSONBufLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
pJSONBuf	Pointer to JSON parse buffer.
JSONBufLen	Octet length of the JSON parse buffer.

Additional information

The JSON parse buffer must be large enough to accumulate JSON “atoms” such as strings and numbers. If the JSON buffer is not large enough when processing a JSON atom, the JSON parse fails and the error is propagated through the Dropbox client to the caller.

4.4.4 IOT_DROPBOX_SetAPIKey()

Description

Set API key for user’s Dropbox.

Prototype

```
void IOT_DROPBOX_SetAPIKey(      IOT_DROPBOX_CONTEXT * pSelf,
                           const char                * pAPIKey);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
pAPIKey	Pointer to API key.

Additional information

The Dropbox client takes possession of the API key and the data that it points to must remain within scope during all API calls until the Dropbox context is released by `IOT_DROPBOX_Exit()`.

4.4.5 IOT_DROPBOX_SetFlag()

Description

Set client context flag.

Prototype

```
void IOT_DROPBOX_SetFlag(IOT_DROPBOX_CONTEXT * pSelf,
                         unsigned              Flag);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
Flag	Flag to set.

4.4.6 IOT_DROPBOX_SetIO()

Description

Set HTTP transport I/O.

Prototype

```
void IOT_DROPBOX_SetIO(      IOT_DROPBOX_CONTEXT * pSelf,
                       const IOT_IO_API          * pAPI,
                             void                * pContext);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
pAPI	Pointer to I/O API.
pContext	Pointer to context passed to I/O API.

4.5 Management functions

The table below lists the functions provided by the emDropbox API. Detailed description of each function is found in the sections that follow.

Function	Description
`IOT_DROPBOX_Copy()`	Copy file or folder.
`IOT_DROPBOX_Remove()`	Delete file or folder.
`IOT_DROPBOX_Move()`	Move file or folder.
`IOT_DROPBOX_CreateFolder()`	Create folder.

4.5.1 IOT_DROPBOX_Copy()

Description

Copy file or folder.

Prototype

```
int IOT_DROPBOX_Copy(      IOT_DROPBOX_CONTEXT * pSelf,
                     const char                * sFromPath,
                     const char                * sToPath);
```

Parameters

Parameter	Description
`pSelf`	Pointer to Dropbox client context.
`sFromPath`	Existing path to user file or folder within Dropbox.
`sToPath`	New path for user file or folder within Dropbox.

Return value

≥ 0	Success.
< 0	Failure.

4.5.2 IOT_DROPBOX_Remove()

Description

Delete file or folder.

Prototype

```
int IOT_DROPBOX_Remove(      IOT_DROPBOX_CONTEXT * pSelf,
                       const char                * sPath);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
sPath	Existing path to user file or folder within Dropbox.

Return value

≥ 0	Success.
< 0	Failure.

4.5.3 IOT_DROPBOX_Move()

Description

Move file or folder.

Prototype

```
int IOT_DROPBOX_Move(      IOT_DROPBOX_CONTEXT * pSelf,
                     const char                * sFromPath,
                     const char                * sToPath);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
sFromPath	Existing path to user file or folder within Dropbox.
sToPath	New path for user file or folder within Dropbox.

Return value

≥ 0	Success.
< 0	Failure.

4.5.4 IOT_DROPBOX_CreateFolder()

Description

Create folder.

Prototype

```
int IOT_DROPBOX_CreateFolder(      IOT_DROPBOX_CONTEXT * pSelf,
                             const char                * sPath);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
sPath	Path to new folder within Dropbox.

Return value

≥ 0	Success.
< 0	Failure.

4.6 Upload and download functions

The table below lists the functions provided by the emDropbox API. Detailed description of each function is found in the sections that follow.

Function	Description
`IOT_DROPBOX_GetBegin()`	Download file from Dropbox.
`IOT_DROPBOX_GetContent()`	Continue download from Dropbox.
`IOT_DROPBOX_GetEnd()`	Finish download from Dropbox.
`IOT_DROPBOX_GetMetadata()`	Acquire and process Dropbox metadata.
`IOT_DROPBOX_PutBegin()`	Upload file to Dropbox.
`IOT_DROPBOX_PutContent()`	Continue upload to Dropbox.
`IOT_DROPBOX_PutEnd()`	Finish upload to Dropbox.

4.6.1 IOT_DROPBOX_GetBegin()

Description

Download file from Dropbox.

Prototype

```
int IOT_DROPBOX_GetBegin(      IOT_DROPBOX_CONTEXT * pSelf,
                         const char                * sPath);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
sPath	Path to existing file within Dropbox.

Return value

≥ 0	Success.
< 0	Failure.

4.6.2 IOT_DROPBOX_GetContent()

Description

Continue download from Dropbox.

Prototype

```
int IOT_DROPBOX_GetContent(IOT_DROPBOX_CONTEXT * pSelf,
                           void                * pData,
                           unsigned              DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
pData	Pointer to object that receives the data.
DataLen	Maximum amount of data to receive.

Return value

≥ 0	Success.
< 0	Failure.

4.6.3 IOT_DROPBOX_GetEnd()

Description

Finish download from Dropbox.

Prototype

```
void IOT_DROPBOX_GetEnd(IOT_DROPBOX_CONTEXT * pSelf);
```

Parameters

Parameter	Description
`pSelf`	Pointer to Dropbox client context.

4.6.4 IOT_DROPBOX_GetMetadata()

Description

Acquire and process Dropbox metadata.

Prototype

```
int IOT_DROPBOX_GetMetadata(      IOT_DROPBOX_CONTEXT            * pSelf,
                            const char                           * sPath,
                                  IOT_DROPBOX_METADATA_ENUM_FUNC * pfEnum);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
sPath	Path to user folder within Dropbox.
pfEnum	Pointer to metadata enumeration function.

Return value

≥ 0	Success.
< 0	Failure.

4.6.5 IOT_DROPBOX_PutBegin()

Description

Upload file to Dropbox.

Prototype

```
int IOT_DROPBOX_PutBegin(      IOT_DROPBOX_CONTEXT * pSelf,
                         const char                * sPath,
                               unsigned              ContentLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
sPath	Path to new file within Dropbox.
ContentLen	Octet length of the payload (file size) to be uploaded.

Return value

≥ 0	Success.
< 0	Failure.

4.6.6 IOT_DROPBOX_PutContent()

Description

Continue upload to Dropbox.

Prototype

```
int IOT_DROPBOX_PutContent(      IOT_DROPBOX_CONTEXT * pSelf,
                           const void                * pData,
                                 unsigned              DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.
pData	Pointer to payload data to transmit to Dropbox.
DataLen	Octet length of the payload data to transmit to Dropbox.

Return value

≥ 0	Success.
< 0	Failure.

4.6.7 IOT_DROPBOX_PutEnd()

Description

Finish upload to Dropbox.

Prototype

```
int IOT_DROPBOX_PutEnd(IOT_DROPBOX_CONTEXT * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to Dropbox client context.

Return value

≥ 0	Success.
< 0	Failure.

Chapter 5

Configuration

5.1 Configuring emSSL for Dropbox

5.1.1 Cipher suites and elliptic curves

Bescause RAM and flash is limited on embedded systems, it’s important to know what capabilities are required to connect to a server securely. With emSSL it is possible to scan a server and derive a set of cipher suites that are common to client and server.

The Dropbox API uses `api.dropboxapi.com` for general administration and `content.dropboxapi.com` for file upload and download. Using emSSL’s scan capability provides the set of common cipher suites:

```
C:> ssl_scan -c api.dropboxapi.com

(c) 2014-2018 SEGGER Microcontroller GmbH    www.segger.com
emSSL TLS Scan V2.52 compiled Feb  2 2018 16:07:16

Scanning cipher suites for content.dropboxapi.com:443...

009D RSA_WITH_AES_256_GCM_SHA384              TLS 1.2  RSA     171 ms
003D RSA_WITH_AES_256_CBC_SHA256              TLS 1.2  RSA     195 ms
0035 RSA_WITH_AES_256_CBC_SHA                 TLS 1.2  RSA     183 ms
009C RSA_WITH_AES_128_GCM_SHA256              TLS 1.2  RSA     181 ms
003C RSA_WITH_AES_128_CBC_SHA256              TLS 1.2  RSA     151 ms
002F RSA_WITH_AES_128_CBC_SHA                 TLS 1.2  RSA     170 ms
000A RSA_WITH_3DES_EDE_CBC_SHA                TLS 1.2  RSA     175 ms
C030 ECDHE_RSA_WITH_AES_256_GCM_SHA384        TLS 1.2  RSA      77 ms
      | secp256r1
C028 ECDHE_RSA_WITH_AES_256_CBC_SHA384        TLS 1.2  RSA      64 ms
      | secp256r1
C014 ECDHE_RSA_WITH_AES_256_CBC_SHA           TLS 1.2  RSA      94 ms
      | secp256r1
C02F ECDHE_RSA_WITH_AES_128_GCM_SHA256        TLS 1.2  RSA      94 ms
      | secp256r1
C027 ECDHE_RSA_WITH_AES_128_CBC_SHA256        TLS 1.2  RSA     101 ms
      | secp256r1
C013 ECDHE_RSA_WITH_AES_128_CBC_SHA           TLS 1.2  RSA      94 ms
      | secp256r1
009F DHE_RSA_WITH_AES_256_GCM_SHA384          TLS 1.2  RSA     253 ms
006B DHE_RSA_WITH_AES_256_CBC_SHA256          TLS 1.2  RSA     259 ms
0039 DHE_RSA_WITH_AES_256_CBC_SHA             TLS 1.2  RSA     264 ms
009E DHE_RSA_WITH_AES_128_GCM_SHA256          TLS 1.2  RSA     262 ms
0067 DHE_RSA_WITH_AES_128_CBC_SHA256          TLS 1.2  RSA     267 ms
0033 DHE_RSA_WITH_AES_128_CBC_SHA             TLS 1.2  RSA     274 ms

19 common cipher suites out of 78 tested

C:> _
```

The only elliptic curve supported by the Dropbox server is P-256 and this will be the only curve that needs to be installed into emSSL when acting as a Dropbox client.

Chapter 6

Resource usage

This chapter covers the resource usage of emDropbox. It contains information about the memory requirements in typical systems, which can be used to obtain sufficient estimates for most target systems.

6.1 Memory footprint

SEGGER emDropbox uses shared components such as the HTTP client, JSON parser, and emSSL for secure connections. In addition it will require a TCP/IP stack such as embOS/IP. The numbers presented for emDropbox exclude the emSSL and TCP/IP stack requirements because these can be configured separately and resource requirements are documented in their respective manuals.

6.1.1 Target system configuration

The following table shows the hardware and the toolchain details of a typical target system:

Detail	Description
CPU	Cortex-M4
Tool chain	SEGGER Embedded Studio with Clang version 3.7
Model	Thumb-2 instructions
Compiler options	Highest size optimization

6.1.2 ROM use

The following table shows the ROM requirement for the emDropbox components:

Component	Size (approximate)
emDropbox (all capabilities)	2.2 KB
HTTP client	2.3 KB
JSON Parser	2.0 KB
Total	**6.5 KB**

6.1.3 RAM use

SEGGER emDropbox has no static RAM requirement.

Chapter 7

Appendix

7.1 Dropbox Commander complete listing

```c
/*********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File        : IOT_DROPBOX_Commander.c
Purpose     : Demonstration of the SEGGER Dropbox API.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "IOT_Dropbox.h"
#include "SSL.h"
#include "SEGGER_SYS.h"
#include "SEGGER_SHELL.h"
#include "SEGGER_MEM.h"
#include <stdio.h>

/*********************************************************************
*
*       Prototypes
*
**********************************************************************
*/

static int _ExecToken      (SEGGER_SHELL_CONTEXT *pShell);
static int _ExecMkdir      (SEGGER_SHELL_CONTEXT *pShell);
static int _ExecCat        (SEGGER_SHELL_CONTEXT *pShell);
static int _ExecGet        (SEGGER_SHELL_CONTEXT *pShell);
static int _ExecPut        (SEGGER_SHELL_CONTEXT *pShell);
static int _ExecLs         (SEGGER_SHELL_CONTEXT *pShell);
static int _ExecRm         (SEGGER_SHELL_CONTEXT *pShell);
static int _ExecMv         (SEGGER_SHELL_CONTEXT *pShell);
static int _ExecCp         (SEGGER_SHELL_CONTEXT *pShell);

/*********************************************************************
*
*       Local data types
*
**********************************************************************
*/

typedef struct {
  SSL_SESSION          SSLContext;
  IOT_DROPBOX_CONTEXT DropboxContext;
} DROPBOX_SESSION;


/*********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const SEGGER_SHELL_CONSOLE_API _ConsoleAPI = {
  SEGGER_SYS_IO_Printvf,
  SEGGER_SYS_IO_Gets
};

static const SSL_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
};
```

```
static const SEGGER_SHELL_COMMAND_API _aCommands[] = {
  { "token",  "Set the access token.",    "<token>",            NULL, _ExecToken },
  { "mkdir",  "Create a folder.",         "<name>",             NULL, _ExecMkdir },
  { "type",   "Display file content.",    "<name>",             NULL, _ExecCat   },
  { "cat",    "Display file content.",    "<name>",             NULL, _ExecCat   },
  { "get",    "Retrieve a file.",         "<name>",             NULL, _ExecGet   },
  { "put",    "Store a file.",            "<name>",             NULL, _ExecPut   },
  { "dir",    "List directory contents.", "[<name>]",           NULL, _ExecLs    },
  { "ls",     "List directory contents.", "[<name>]",           NULL, _ExecLs    },
  { "del",    "Remove a file or folder.", "<name>",             NULL, _ExecRm    },
  { "rm",     "Remove a file or folder.", "<name>",             NULL, _ExecRm    },
  { "cp",     "Copy a file or folder.",   "<fromame> <toname>",  NULL, _ExecCp    },
  { "copy",   "Copy a file or folder.",   "<fromame> <toname>",  NULL, _ExecCp    },
  { "mv",     "Rename a file or folder.", "<oldname> <newname>", NULL, _ExecMv    },
  { "rename", "Rename a file or folder.", "<oldname> <newname>", NULL, _ExecMv    }
};

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static SEGGER_MEM_CONTEXT   _MemContext;
static DROPBOX_SESSION      _API;
static char                 _aToken[128];
static char                 _aJSONBuf[260];

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _PrintBanner()
*
*  Function description
*    Displays the application's sign-on banner.
*
*  Parameters
*    pShell - Pointer to shell context.
*/
static void _PrintBanner(SEGGER_SHELL_CONTEXT *pShell) {
  SEGGER_SHELL_Printf(pShell, "\n");
  SEGGER_SHELL_Printf(pShell, "%s   www.segger.com\n",
                      IOT_DROPBOX_GetCopyrightText());
  SEGGER_SHELL_Printf(pShell, "SEGGER Dropbox Commander V%s ",
                      IOT_DROPBOX_GetVersionText());
  SEGGER_SHELL_Printf(pShell, "compiled " __DATE__ " " __TIME__ "\n\n");
}

/*********************************************************************
*
*       _PrintWarranty()
*
*  Function description
*    Displays the application's warranty information.
*
*  Parameters
*    pShell - Pointer to shell context.
*/
static void _PrintWarranty(SEGGER_SHELL_CONTEXT *pShell) {
  SEGGER_SHELL_Printf(pShell, "THIS UTILITY IS FREE.\n\n");
  SEGGER_SHELL_Printf(pShell, "This utility demonstrates that you can use SEGGER\n");
  SEGGER_SHELL_Printf(pShell, "software to access Dropbox content securely with emSSL\n");
  SEGGER_SHELL_Printf(pShell, "and any TCP/IP stack (such as embOS/IP) that supports\n");
  SEGGER_SHELL_Printf(pShell, "sockets.\n\n");
  SEGGER_SHELL_Printf(pShell, "For more information, contact info@segger.com.\n\n");
  SEGGER_SHELL_Printf(pShell, "Type \"?\" for a list of commands.\n\n");
}
```

```
/*********************************************************************
*
*       _Connect()
*
*  Function description
*    Connect to host using secure sockets.
*
*  Parameters
*    pVoid - Pointer to SSL session context.
*    sHost - Name of server we wish to connect to.
*    Port  - Port number in host byte order.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Connect(void *pVoid, const char *sHost, unsigned Port) {
  SSL_SESSION * pSession;
  int           Socket;
  int           Status;
  //
  pSession = pVoid;
  Status = SEGGER_SYS_IP_Open(sHost, Port);
  if (Status >= 0) {
    Socket = Status;
    SSL_SESSION_Prepare(pSession, Socket, &_IP_Transport);
    Status = SSL_SESSION_Connect(pSession, sHost);
    if (Status < 0) {
      SEGGER_SYS_IP_Close(Socket);
    }
  }
  return Status;
}

/*********************************************************************
*
*       _Disconnect()
*
*  Function description
*    Disconnect from host.
*
*  Parameters
*    pVoid - Pointer to SSL session context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Disconnect(void *pVoid) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  SSL_SESSION_Disconnect(pSession);
  //
  return 0;
}

/*********************************************************************
*
*       _Send()
*
*  Function description
*    Send data to host.
*
*  Parameters
*    pVoid   - Pointer to SSL session context.
*    pData   - Pointer to octet string to send over SSL.
*    DataLen - Octet length of the octet string to send.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Send(void *pVoid, const void *pData, unsigned DataLen) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  return SSL_SESSION_Send(pSession, pData, DataLen);
}

/*********************************************************************
*
*       _Recv()
*
*  Function description
*    Receive data from host.
*
*  Parameters
*    pVoid   - Pointer to SSL session context.
*    pData   - Pointer to object that receives the data.
*    DataLen - Octet length of receiving object.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _Recv(void *pVoid, void *pData, unsigned DataLen) {
  SSL_SESSION * pSession;
  //
  pSession = pVoid;
  return SSL_SESSION_Receive(pSession, pData, DataLen);
}

/*********************************************************************
*
*       _PrintListing()
*
*  Function description
*    Display Dropbox metadata.
*
*  Parameters
*    pMetadata - Pointer to Dropbox metadata.
*/
static void _PrintListing(const IOT_DROPBOX_METADATA *pMetadata) {
  if (pMetadata->IsFolder) {
    SEGGER_SYS_IO_Printf("    <DIR>  %s\n", pMetadata->aPath);
  } else {
    SEGGER_SYS_IO_Printf("%9lld  %s\n", pMetadata->Size, pMetadata->aPath);
  }
}

/*********************************************************************
*
*       _ExecMkdir()
*
*  Function description
*    Create a folder.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecMkdir(SEGGER_SHELL_CONTEXT *pShell) {
  char * sPath;
  int    Status;
  //
  Status = SEGGER_SHELL_ReadNextArg(pShell, &sPath);
  if (Status >= 0) {
    Status = IOT_DROPBOX_CreateFolder(&_API.DropboxContext, sPath);
    IOT_DROPBOX_Exit(&_API.DropboxContext);
  }
  return Status;
}

/*********************************************************************
*
*       _ExecCat()
*
*  Function description
*    List a file.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecCat(SEGGER_SHELL_CONTEXT *pShell) {
  char   aBuf[128];
  char * sPath;
  int    Status;
  //
  Status = SEGGER_SHELL_ReadNextArg(pShell, &sPath);
  if (Status >= 0) {
    Status = IOT_DROPBOX_GetBegin(&_API.DropboxContext, sPath);
  }
  if (Status >= 0) {
    for (;;) {
      int N;
      //
      N = IOT_DROPBOX_GetContent(&_API.DropboxContext, aBuf, sizeof(aBuf)-1);
      if (N <= 0) {
        break;
      }
      aBuf[N] = 0;
      SEGGER_SHELL_Printf(pShell, "%s", aBuf);
    }
  }
  IOT_DROPBOX_Exit(&_API.DropboxContext);
  //
  return Status;
}

/*********************************************************************
*
*       _ExecLs()
*
*  Function description
*    List a directory.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecLs(SEGGER_SHELL_CONTEXT *pShell) {
  char * sPath;
  int    Status;
  //
  if (SEGGER_SHELL_HasUnreadArgs(pShell)) {
    Status = SEGGER_SHELL_ReadNextArg(pShell, &sPath);
  } else {
    Status = 0;
    sPath = "/";
  }
  if (Status >= 0) {
    Status = IOT_DROPBOX_GetMetadata(&_API.DropboxContext, sPath, _PrintListing);
    IOT_DROPBOX_Exit(&_API.DropboxContext);
  }
  return Status;
}

/*********************************************************************
*
*       _ExecRm()
*
*  Function description
*    Remove a file or folder.
*
*  Parameters
*    pShell - Pointer to shell context.
*
```

```
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecRm(SEGGER_SHELL_CONTEXT *pShell) {
  char * sPath;
  int    Status;
  //
  Status = SEGGER_SHELL_ReadNextArg(pShell, &sPath);
  if (Status >= 0) {
    Status = IOT_DROPBOX_Remove(&_API.DropboxContext, sPath);
    IOT_DROPBOX_Exit(&_API.DropboxContext);
  }
  return Status;
}

/*********************************************************************
*
*       _ExecMv()
*
*  Function description
*    Rename a file or folder.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecMv(SEGGER_SHELL_CONTEXT *pShell) {
  char * sFromPath;
  char * sToPath;
  int    Status;
  //
  Status = SEGGER_SHELL_ReadNextArg(pShell, &sFromPath);
  if (Status >= 0) {
    Status = SEGGER_SHELL_ReadNextArg(pShell, &sToPath);
    if (Status >= 0) {
      Status = IOT_DROPBOX_Move(&_API.DropboxContext, sFromPath, sToPath);
      IOT_DROPBOX_Exit(&_API.DropboxContext);
    }
  }
  return Status;
}

/*********************************************************************
*
*       _ExecCp()
*
*  Function description
*    Copy a file or folder.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecCp(SEGGER_SHELL_CONTEXT *pShell) {
  char * sFromPath;
  char * sToPath;
  int    Status;
  //
  Status = SEGGER_SHELL_ReadNextArg(pShell, &sFromPath);
  if (Status >= 0) {
    Status = SEGGER_SHELL_ReadNextArg(pShell, &sToPath);
    if (Status >= 0) {
      Status = IOT_DROPBOX_Copy(&_API.DropboxContext, sFromPath, sToPath);
      IOT_DROPBOX_Exit(&_API.DropboxContext);
    }
  }
  return Status;
}
```

```
/*********************************************************************
*
*       _ExecToken()
*
*  Function description
*    Set authentication token.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecToken(SEGGER_SHELL_CONTEXT *pShell) {
  char     *sToken;
  int       Status;
  //
  Status = SEGGER_SHELL_ReadNextArg(pShell, &sToken);
  if (Status >= 0) {
    if (strlen(sToken) < sizeof(_aToken)) {
      strcpy(&_aToken[0], sToken);
      IOT_DROPBOX_SetAPIKey(&_API.DropboxContext, &_aToken[0]);
    } else {
      Status = -1;
    }
  } else {
    SEGGER_SHELL_Printf(pShell, "Token: %s\n", _API.DropboxContext.sToken);
    Status = 0;
  }
  return Status;
}

/*********************************************************************
*
*       _ExecGet()
*
*  Function description
*    Download file.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecGet(SEGGER_SHELL_CONTEXT *pShell) {
  char*    sPath;
  char*    sName;
  void*    hFile;
  unsigned ContentLen;
  int      Status;
  char     aBuf[128];
  int      N;
  //
  //
  Status = SEGGER_SHELL_ReadNextArg(pShell, &sPath);
  if (Status >= 0) {
    Status = IOT_DROPBOX_GetBegin(&_API.DropboxContext, sPath);
    if (Status >= 0) {
      sName = strrchr(sPath, '/');
      if (sName == 0) {
        sName = sPath;
      } else {
        ++sName;
      }
      hFile = fopen(sName, "wb");
      if (hFile == NULL) {
        Status = -1;
      } else {
        ContentLen = 0;
        for (;;) {
          N = IOT_DROPBOX_GetContent(&_API.DropboxContext, aBuf, sizeof(aBuf));
          if (N <= 0) {
            break;
          }
          fwrite(aBuf, 1, N, hFile);
          ContentLen += N;
        }
        fclose(hFile);
      }
      IOT_DROPBOX_GetEnd(&_API.DropboxContext);
    }
  }
  IOT_DROPBOX_Exit(&_API.DropboxContext);
  //
  if (Status >= 0) {
    SEGGER_SHELL_Printf(pShell, "Wrote %u bytes.\n", ContentLen);
  }
  //
  return Status;
}

/*********************************************************************
*
*       _ExecPut()
*
*  Function description
*    Upload file.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Processing error.
*/
static int _ExecPut(SEGGER_SHELL_CONTEXT *pShell) {
  char     aBuf[128];
  char*    sPath;
  void*    hFile;
  unsigned ContentLen;
  unsigned FilePos;
  unsigned FragLen;
  int      Status;
  //
  Status = SEGGER_SHELL_ReadNextArg(pShell, &sPath);
  hFile = fopen(sPath, "rb");
  if (hFile == NULL) {
    Status = -1;
  } else {
    fseek(hFile, 0, SEEK_END);
    ContentLen = ftell(hFile);
    fseek(hFile, 0, SEEK_SET);
    IOT_DROPBOX_SetFlag(&_API.DropboxContext, IOT_DROPBOX_FLAG_OVERWRITE);
    Status = IOT_DROPBOX_PutBegin(&_API.DropboxContext, sPath, ContentLen);
    if (Status >= 0) {
      FilePos = 0;
      do {
        FragLen  = SEGGER_MIN(sizeof(aBuf), ContentLen-FilePos);
        fread(aBuf, 1, FragLen, hFile);
        FilePos += FragLen;
        Status = IOT_DROPBOX_PutContent(&_API.DropboxContext, aBuf, FragLen);
      } while ((FilePos < ContentLen) && (Status >= 0));
    }
    if (Status >= 0) {
      Status = IOT_DROPBOX_PutEnd(&_API.DropboxContext);
    }
    fclose(hFile);
  }
  //
  if (Status >= 0) {
    SEGGER_SHELL_Printf(pShell, "Wrote %u bytes.\n", ContentLen);
  }
  //
  return Status;
}

/*********************************************************************
*
*       _ExecInitScript()
```

```
*
*  Function description
*    Execute initialization script.
*
*  Parameters
*    pShell    - Pointer to shell context.
*    sFileName - File name of initialization script.
*/
static void _ExecInitScript(SEGGER_SHELL_CONTEXT *pShell, const char *sFileName) {
  FILE * pFile;
  char * pNewline;
  char   acBuf[256];
  //
  pFile = fopen(sFileName, "r");
  if (pFile == 0) {
    return;
  }
  //
  while (!feof(pFile)) {
    fgets(acBuf, sizeof(acBuf)-1, pFile);
    pNewline = strchr(acBuf, '\n');
    if (pNewline) {
      *pNewline = 0;
    }
    if (SEGGER_SHELL_ParseInput(pShell, acBuf) >= 0) {
      SEGGER_SHELL_Process(pShell);
    }
  }
  //
  fclose(pFile);
}

/*********************************************************************
*
*       _InitDropboxContext()
*
*  Function description
*    Initialize Dropbox client.
*
*  Parameters
*    pShell - Pointer to shell context.
*/
static void _InitDropboxContext(SEGGER_SHELL_CONTEXT *pShell) {
  unsigned i;
  //
  // Initialize our Dropbox context with an appropriate
  // bearer token.
  //
  static const IOT_IO_API _IOAPI = {
    _Connect,
    _Disconnect,
    _Send,
    _Recv
  };
  //
  IOT_DROPBOX_Init (&_API.DropboxContext, _aJSONBuf, sizeof(_aJSONBuf));
  IOT_DROPBOX_SetIO(&_API.DropboxContext, &_IOAPI, &_API.SSLContext);
  //
  for (i = 0; i < SEGGER_COUNTOF(_aCommands); ++i) {
    SEGGER_SHELL_AddCommandAPI(pShell, &_aCommands[i]);
  }
  SEGGER_SHELL_AddCommandAPI(pShell, &SEGGER_SHELL_QuitAPI);
  SEGGER_SHELL_AddCommandAPI(pShell, &SEGGER_SHELL_QuestionMarkAPI);
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       main()
*
```

```
*  Function description
*    Application entry point.
*
*  Parameters
*    argc - Argument count.
*    argv - Argument vector.
*
*  Return value
*    Application exit status.
*/
int main(int argc, char *argv[]) {
  char                   * sToken;
  SEGGER_SHELL_CONTEXT     Shell;
  int                      Status;
  //
  // Initialize subsystems.
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SEGGER_MEM_SYSTEM_HEAP_Init(&_MemContext);
  SSL_Init();
  //
  // Inherit CLI passed from the OS.
  //
  SEGGER_SHELL_Init(&Shell, &_ConsoleAPI, &_MemContext);
  SEGGER_SHELL_InheritExternal(&Shell, 0, argc, argv);
  //
  // Initialize Dropbox context and commands.
  //
  _InitDropboxContext(&Shell);
  //
  // Show startup text.
  //
  _PrintBanner(&Shell);
  _PrintWarranty(&Shell);
  //
  // Process the startup script.
  //
  IOT_DROPBOX_SetAPIKey(&_API.DropboxContext, "...access token is not set!...");
  _ExecInitScript(&Shell, "startup.cli");
  //
  // Parse Dropbox access token to use.
  //
  if (SEGGER_SHELL_ReadNextArg(&Shell, &sToken) >= 0) {    // Command name
    if (SEGGER_SHELL_ReadNextArg(&Shell, &sToken) >= 0) {  // First argument
      IOT_DROPBOX_SetAPIKey(&_API.DropboxContext, sToken);
    }
  }
  //
  // Process shell.
  //
  Status = SEGGER_SHELL_Enter(&Shell);
  //
  IOT_DROPBOX_Exit(&_API.DropboxContext);
  SSL_Exit();
  SEGGER_SYS_IP_Exit();
  SEGGER_SYS_Exit();
  //
  SEGGER_SYS_OS_Halt(Status);
  return Status;
}

/*************************** End of file ****************************/
```

Chapter 8

Indexes

8.1 Data type index

8.2 Function index