EBook Gratuito

APPRENDIMENTO

Dropbox API

Free unaffiliated eBook created from
Stack Overflow contributors.

#dropbox-api

Sommario

Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: dropbox-api

It is an unofficial and free Dropbox API ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Dropbox API.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Introduzione a Dropbox API

Osservazioni

L' API Dropbox consente agli sviluppatori di creare funzionalità Dropbox direttamente nelle loro app.

L'API consente l'accesso a funzionalità come il caricamento di file, il download, la condivisione, la ricerca e il ripristino. L'API può essere utilizzata su piattaforme come Windows, Mac, Linux, iOS, Android o qualsiasi altro che possa effettuare connessioni HTTPS.

Ulteriori informazioni, tra cui la documentazione completa per l'API Dropbox, le linee guida sull'utilizzo e gli strumenti per sviluppatori come gli SDK ufficiali sono disponibili sul sito Web dell'API Dropbox .

Il primo passo è registrare un'app API con Dropbox .

Versioni

Versione	Data di rilascio
2	2015/11/04

Examples

Ottenere un token di accesso OAuth 2 per l'API Dropbox tramite la concessione del codice utilizzando curl

Abbreviato da https://blogs.dropbox.com/developers/2013/07/using-oauth-2-0-with-the-core-api/ :

Passaggio 1: iniziare l'autorizzazione

Invia l'utente a questa pagina Web, con i tuoi valori compilati:

```
https://www.dropbox.com/oauth2/authorize?client_id=<app
key>&response_type=code&redirect_uri=<redirect URI>&state=<CSRF token>
```

Il codice di autorizzazione verrà incluso come parametro di `code` sull'URI di reindirizzamento.

Passaggio 2: ottenere un token di accesso

```
curl https://api.dropbox.com/oauth2/token -d code=<authorization code> -d
grant_type=authorization_code -d redirect_uri=<redirect URI> -u <app key>:<app secret>
```

Passaggio 3: chiama l'API

Nella tua chiamata API, imposta l'intestazione:

```
Authorization: Bearer <access token>
```

Consulta il post del blog per ulteriori dettagli, tra cui un'importante nota di sicurezza sull'utilizzo `state` per proteggerti dagli attacchi CSRF.

Leggi Introduzione a Dropbox API online: https://riptutorial.com/it/dropbox-api/topic/359/introduzione-a-dropbox-api

Capitolo 2: Caricamento di un file

Examples

Caricamento di un file tramite curl in PHP

```
<?php

$path = 'test_php_upload.txt';
$fp = fopen($path, 'rb');
$size = filesize($path);

$cheaders = array('Authorization: Bearer <ACCESS_TOKEN>',
                  'Content-Type: application/octet-stream',
                  'Dropbox-API-Arg: {"path":"/test/'.$path.'", "mode":"add"}');

$ch = curl_init('https://content.dropboxapi.com/2/files/upload');
curl_setopt($ch, CURLOPT_HTTPHEADER, $cheaders);
curl_setopt($ch, CURLOPT_PUT, true);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST');
curl_setopt($ch, CURLOPT_INFILE, $fp);
curl_setopt($ch, CURLOPT_INFILESIZE, $size);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);

echo $response;
curl_close($ch);
fclose($fp);

?>
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Caricamento di un file tramite curl in C ++

```
#include <stdio.h>
#include <curl/curl.h>

int main (int argc, char *argv[])
{
      CURL *curl;
      CURLcode res;

      /* In windows, this will init the winsock stuff */
      curl_global_init(CURL_GLOBAL_ALL);

      /* get a curl handle */
      curl = curl_easy_init();
      if(curl) {

            printf ("Running curl test.\n");

            struct curl_slist *headers=NULL; /* init to NULL is important */
            headers = curl_slist_append(headers, "Authorization: Bearer <ACCESS_TOKEN>");
```

```
            headers = curl_slist_append(headers, "Content-Type: application/octet-stream");
            headers = curl_slist_append(headers, "Dropbox-API-Arg:
{\"path\":\"/test_c++_upload_test.txt\"}");
            curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);

            curl_easy_setopt(curl, CURLOPT_URL,
"https://content.dropboxapi.com/2/files/upload");
            curl_easy_setopt(curl, CURLOPT_POSTFIELDS, "test data for upload");

            /* Perform the request, res will get the return code */
            res = curl_easy_perform(curl);
            /* Check for errors */
            if(res != CURLE_OK)
                  fprintf(stderr, "curl_easy_perform() failed: %s\n",
                    curl_easy_strerror(res));

            /* always cleanup */
              curl_easy_cleanup(curl);

              printf ("\nFinished curl test.\n");

      }
      curl_global_cleanup();

      printf ("Done!\n");
      return 0;

}
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Caricamento di un file tramite arricciatura

Questo carica un file dal percorso locale `matrices.txt` nella cartella corrente in `/Homework/math/Matrices.txt` nell'account Dropbox e restituisce i metadati per il file caricato:

```
echo "some content here" > matrices.txt

curl -X POST https://content.dropboxapi.com/2/files/upload \
    --header "Authorization: Bearer <ACCESS_TOKEN>" \
    --header "Dropbox-API-Arg: {\"path\": \"/Homework/math/Matrices.txt\"}" \
    --header "Content-Type: application/octet-stream" \
    --data-binary @matrices.txt
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Caricamento di un file da NSData con ogni caso di errore gestito utilizzando la libreria SwiftyDropbox

Questo utilizza la libreria SwiftyDropbox per caricare un file da un `NSData` all'account Dropbox, utilizzando sessioni di upload per file più grandi, gestendo ogni caso di errore:

```
import UIKit
import SwiftyDropbox
```

```
class ViewController: UIViewController {

    // replace this made up data with the real data
    let data = String(count: 20 * 1024 * 1024, repeatedValue:
Character("A")).dataUsingEncoding(NSUTF8StringEncoding)!


    let chunkSize = 5 * 1024 * 1024  // 5 MB
    var offset = 0
    var sessionId = ""

    // replace this with your desired destination path:
    let destPath = "/SwiftyDropbox_upload.txt"

    override func viewDidLoad() {
        super.viewDidLoad()

        Dropbox.authorizedClient = DropboxClient(...)

        doUpload()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func doUpload() {

        let fileSize = data.length

        print("Have \(fileSize) bytes to upload.")

        if (fileSize < chunkSize) {

            print("Using non-chunked uploading...")

            Dropbox.authorizedClient!.files.upload(path:destPath, input:data).response {
response, error in
                if let metadata = response {
                    print(metadata)
                } else if let callError = error {
                    print("upload failed")
                    switch callError as CallError {
                    case .RouteError(let boxed, let requestId):
                        print("RouteError[\(requestId)]:")
                        switch boxed.unboxed as Files.UploadError {
                        case .Path(let uploadError):
                            print("Path:")
                            switch uploadError.reason as Files.WriteError {
                            case .MalformedPath(let malformedPathError):
                                print("MalformedPath: \(malformedPathError)")
                            case .Conflict(let writeConflictError):
                                print("Conflict:")
                                switch writeConflictError {
                                case .File:
                                    print("File")
                                case .FileAncestor:
                                    print("FileAncestor")
                                case .Folder:
                                    print("Folder")
                                case .Other:
```

```
                                print("Other")
                            }
                        case .DisallowedName:
                            print("DisallowedName")
                        case .InsufficientSpace:
                            print("InsufficientSpace")
                        case .NoWritePermission:
                            print("NoWritePermission")
                        case .Other:
                            print("Other")
                        }
                    case .Other:
                        print("Other")
                    }
                case .BadInputError(let message, let requestId):
                    print("BadInputError[\(requestId)]: \(message)")
                case .HTTPError(let code, let message, let requestId):
                    print("HTTPError[\(requestId)]: \(code): \(message)")
                case .InternalServerError(let code, let message, let requestId):
                    print("InternalServerError[\(requestId)]: \(code): \(message)")
                case .OSError(let err):
                    print("OSError: \(err)")
                case .RateLimitError:
                    print("RateLimitError")
                }
            }
        }

    } else {

        print("Using chunked uploading...")

        uploadFirstChunk()

    }
}

func uploadFirstChunk() {
    let size = min(chunkSize, data.length)
    Dropbox.authorizedClient!.files.uploadSessionStart(input:
        data.subdataWithRange(NSMakeRange(0, size)))
        .response { response, error in
            if let result = response {
                self.sessionId = result.sessionId
                self.offset += size
                print("So far \(self.offset) bytes have been uploaded.")
                self.uploadNextChunk()
            } else if let callError = error {
                print("uploadSessionStart failed")
                switch callError as CallError {
                case .RouteError(let error, let requestId):
                    print("RouteError[\(requestId)]: \(error)")
                case .BadInputError(let message, let requestId):
                    print("BadInputError[\(requestId)]: \(message)")
                case .HTTPError(let code, let message, let requestId):
                    print("HTTPError[\(requestId)]: \(code): \(message)")
                case .InternalServerError(let code, let message, let requestId):
                    print("InternalServerError[\(requestId)]: \(code): \(message)")
                case .OSError(let err):
                    print("OSError: \(err)")
                case .RateLimitError:
```

```
                    print("RateLimitError")
                }
            }
        }
    }

    func uploadNextChunk() {
        if data.length - offset <= chunkSize {
            let size = data.length - offset
            Dropbox.authorizedClient!.files.uploadSessionFinish(
                cursor: Files.UploadSessionCursor(
                    sessionId: self.sessionId, offset: UInt64(offset)),
                commit: Files.CommitInfo(path:destPath),
                input: data.subdataWithRange(NSMakeRange(offset, size)))
                .response { response, error in
                    if let callError = error {
                        print("uploadSessionFinish failed")
                        switch callError as CallError {
                        case .RouteError(let boxed, let requestId):
                            print("RouteError[\(requestId)]:")
                            switch boxed.unboxed as Files.UploadSessionFinishError {
                            case .Path(let writeError):
                                print("Path: ")
                                switch writeError {
                                case .MalformedPath(let malformedPathError):
                                    print("MalformedPath: \(malformedPathError)")
                                case .Conflict(let writeConflictError):
                                    print("Conflict:")
                                    switch writeConflictError {
                                    case .File:
                                        print("File")
                                    case .FileAncestor:
                                        print("FileAncestor")
                                    case .Folder:
                                        print("Folder")
                                    case .Other:
                                        print("Other")
                                    }
                                case .DisallowedName:
                                    print("DisallowedName")
                                case .InsufficientSpace:
                                    print("InsufficientSpace")
                                case .NoWritePermission:
                                    print("NoWritePermission")
                                case .Other:
                                    print("Other")
                                }
                            case .LookupFailed(let uploadSessionLookupError):
                                print("LookupFailed:")
                                switch uploadSessionLookupError {
                                case .Closed:
                                    print("Closed")
                                case .IncorrectOffset(let uploadSessionOffsetError):
                                    print("IncorrectOffset: \(uploadSessionOffsetError)")
                                case .NotFound:
                                    print("NotFound")
                                case .NotClosed:
                                    print("NotFound")
                                case .Other:
                                    print("Other")
                                }
```

```
                            case .TooManySharedFolderTargets:
                                print("TooManySharedFolderTargets")
                            case .Other:
                                print("Other")
                            }
                        case .BadInputError(let message, let requestId):
                            print("BadInputError[\(requestId)]: \(message)")
                        case .HTTPError(let code, let message, let requestId):
                            print("HTTPError[\(requestId)]: \(code): \(message)")
                        case .InternalServerError(let code, let message, let requestId):
                            print("InternalServerError[\(requestId)]: \(code): \(message)")
                        case .OSError(let err):
                            print("OSError: \(err)")
                        case .RateLimitError:
                            print("RateLimitError")
                        }
                    } else if let result = response {
                        print("Done!")
                        print(result)
                    }
            }
        } else {
            Dropbox.authorizedClient!.files.uploadSessionAppendV2(
                cursor: Files.UploadSessionCursor(sessionId: self.sessionId, offset:
UInt64(offset)),
                input: data.subdataWithRange(NSMakeRange(offset, chunkSize)))
                .response { response, error in
                    if error == nil {
                        self.offset += self.chunkSize
                        print("So far \(self.offset) bytes have been uploaded.")
                        self.uploadNextChunk()
                    } else if let callError = error {
                        print("uploadSessionAppend failed")
                        switch callError as CallError {
                        case .RouteError(let boxed, let requestId):
                            print("RouteError[\(requestId)]:")
                            switch boxed.unboxed as Files.UploadSessionLookupError {
                            case .Closed:
                                print("Closed")
                            case .IncorrectOffset(let uploadSessionOffsetError):
                                print("IncorrectOffset: \(uploadSessionOffsetError)")
                            case .NotFound:
                                print("NotFound")
                            case .NotClosed:
                                print("NotClosed")
                            case .Other:
                                print("Other")
                            }
                        case .BadInputError(let message, let requestId):
                            print("BadInputError[\(requestId)]: \(message)")
                        case .HTTPError(let code, let message, let requestId):
                            print("HTTPError[\(requestId)]: \(code): \(message)")
                        case .InternalServerError(let code, let message, let requestId):
                            print("InternalServerError[\(requestId)]: \(code): \(message)")
                        case .OSError(let err):
                            print("OSError: \(err)")
                        case .RateLimitError:
                            print("RateLimitError")
                        }
                    }
            }
```

```
        }
    }

}
```

Caricamento di un file utilizzando la libreria Dropbox .NET

In questo esempio viene utilizzata la libreria Dropbox .NET per caricare un file su un account Dropbox, utilizzando sessioni di caricamento per file più grandi:

```
private async Task Upload(string localPath, string remotePath)
{
    const int ChunkSize = 4096 * 1024;
    using (var fileStream = File.Open(localPath, FileMode.Open))
    {
        if (fileStream.Length <= ChunkSize)
        {
            await this.client.Files.UploadAsync(remotePath, body: fileStream);
        }
        else
        {
            await this.ChunkUpload(remotePath, fileStream, (int)ChunkSize);
        }
    }
}

private async Task ChunkUpload(String path, FileStream stream, int chunkSize)
{
    ulong numChunks = (ulong)Math.Ceiling((double)stream.Length / chunkSize);
    byte[] buffer = new byte[chunkSize];
    string sessionId = null;
    for (ulong idx = 0; idx < numChunks; idx++)
    {
        var byteRead = stream.Read(buffer, 0, chunkSize);

        using (var memStream = new MemoryStream(buffer, 0, byteRead))
        {
            if (idx == 0)
            {
                var result = await this.client.Files.UploadSessionStartAsync(false,
memStream);
                sessionId = result.SessionId;
            }
            else
            {
                var cursor = new UploadSessionCursor(sessionId, (ulong)chunkSize * idx);

                if (idx == numChunks - 1)
                {
                    FileMetadata fileMetadata = await
this.client.Files.UploadSessionFinishAsync(cursor, new CommitInfo(path), memStream);
                    Console.WriteLine (fileMetadata.PathDisplay);
                }
                else
                {
                    await this.client.Files.UploadSessionAppendV2Async(cursor, false,
memStream);
                }
            }
```

```
        }
    }
}
```

Caricamento di un file tramite jQuery in JavaScript

```
// ... file selected from a file <input>
file = event.target.files[0];
$.ajax({
    url: 'https://content.dropboxapi.com/2/files/upload',
    type: 'post',
    data: file,
    processData: false,
    contentType: 'application/octet-stream',
    headers: {
        "Authorization": "Bearer <ACCESS_TOKEN>",
        "Dropbox-API-Arg": '{"path": "/test_upload.txt","mode": "add","autorename":
true,"mute": false}'
    },
    success: function (data) {
        console.log(data);
    },
    error: function (data) {
        console.error(data);
    }
})
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Caricamento di un file dal testo tramite jQuery in JavaScript

```
var data = new TextEncoder("utf-8").encode("Test");
$.ajax({
    url: 'https://content.dropboxapi.com/2/files/upload',
    type: 'post',
    data: data,
    processData: false,
    contentType: 'application/octet-stream',
    headers: {
        "Authorization": "Bearer <ACCESS_TOKEN>",
        "Dropbox-API-Arg": '{"path": "/test_upload.txt","mode": "add","autorename":
true,"mute": false}'
    },
    success: function (data) {
        console.log(data);
    },
    error: function (data) {
        console.error(data);
    }
})
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Caricamento di un file utilizzando l'SDK Dropbox di Python

Questo utilizza l' SDK Dropbox di Python per caricare un file `file_path` Dropbox dal file locale come specificato da `file_path` al percorso remoto come specificato da `dest_path` . Sceglie anche se utilizzare o meno una sessione di upload in base alle dimensioni del file:

```
f = open(file_path)
file_size = os.path.getsize(file_path)

CHUNK_SIZE = 4 * 1024 * 1024

if file_size <= CHUNK_SIZE:

    print dbx.files_upload(f.read(), dest_path)

else:

    upload_session_start_result = dbx.files_upload_session_start(f.read(CHUNK_SIZE))
    cursor =
dropbox.files.UploadSessionCursor(session_id=upload_session_start_result.session_id,
                                                  offset=f.tell())
    commit = dropbox.files.CommitInfo(path=dest_path)

    while f.tell() < file_size:
        if ((file_size - f.tell()) <= CHUNK_SIZE):
            print dbx.files_upload_session_finish(f.read(CHUNK_SIZE),
                                                  cursor,
                                                  commit)
        else:
            dbx.files_upload_session_append(f.read(CHUNK_SIZE),
                                                  cursor.session_id,
                                                  cursor.offset)
            cursor.offset = f.tell()

f.close()
```

Caricamento di un file dal testo tramite XMLHttpRequest in JavaScript

```
var path = "/test_javascript_upload.txt";
var content = "data to upload";
var accessToken = "<ACCESS_TOKEN>";
var uploadUrl = "https://content.dropboxapi.com/2/files/upload"
var result;

var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
        result = xhr.responseText;
        console.log(result);
    }
};
xhr.open("POST", uploadUrl, true);
xhr.setRequestHeader("Authorization", "Bearer " + accessToken);
xhr.setRequestHeader("Content-type", "application/octet-stream");
xhr.setRequestHeader("Dropbox-API-Arg", '{"path": "' + path + '"}');
xhr.send(content);
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Caricamento di un file da NSFileHandle utilizzando sessioni di caricamento con ogni caso di errore gestito utilizzando la libreria SwiftyDropbox

Questo utilizza la libreria SwiftyDropbox per caricare un file da un `NSFileHandle` all'account Dropbox utilizzando sessioni di caricamento, gestendo ogni caso di errore:

```
import UIKit
import SwiftyDropbox

class ViewController: UIViewController {

    // filled in later in doUpload:
    var fileHandle : NSFileHandle? = nil
    var data : NSData? = nil

    let chunkSize = 5 * 1024 * 1024  // 5 MB
    var offset = 0
    var sessionId = ""

    // replace this with your desired destination path:
    let destPath = "/SwiftyDropbox_upload.txt"

    override func viewDidLoad() {
        super.viewDidLoad()

        Dropbox.authorizedClient = DropboxClient(...)

        doUpload()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func doUpload() {

        // replace this with the path to the file you want to upload
        let filePath = "/path/to/file"
        print("Getting file at \(filePath) for uploading...")
        fileHandle = NSFileHandle.init(forReadingAtPath: filePath)!

        print("Using chunked uploading with chunk size \(chunkSize)...")
        uploadFirstChunk()

    }

    func uploadFirstChunk() {
        data = fileHandle!.readDataOfLength(chunkSize)
        let size = data!.length
        print("Have \(size) bytes to upload.")
        Dropbox.authorizedClient!.files.uploadSessionStart(input:data!)
            .response { response, error in
                if let result = response {
                    self.sessionId = result.sessionId
                    self.offset += size
                    print("So far \(self.offset) bytes have been uploaded.")
                    self.uploadNextChunk()
                } else if let callError = error {
                    print("uploadSessionStart failed")
```

```
                    switch callError as CallError {
                    case .RouteError(let error, let requestId):
                        print("RouteError[\(requestId)]: \(error)")
                    case .BadInputError(let message, let requestId):
                        print("BadInputError[\(requestId)]: \(message)")
                    case .HTTPError(let code, let message, let requestId):
                        print("HTTPError[\(requestId)]: \(code): \(message)")
                    case .InternalServerError(let code, let message, let requestId):
                        print("InternalServerError[\(requestId)]: \(code): \(message)")
                    case .OSError(let err):
                        print("OSError: \(err)")
                    case .RateLimitError:
                        print("RateLimitError")
                    }
                }
        }
    }

    func uploadNextChunk() {
        data = fileHandle!.readDataOfLength(chunkSize)
        let size = data!.length
        print("Have \(size) bytes to upload.")
        if size < chunkSize {
            print("Last chunk!")
            Dropbox.authorizedClient!.files.uploadSessionFinish(
                cursor: Files.UploadSessionCursor(sessionId: self.sessionId, offset:
UInt64(offset)),
                commit: Files.CommitInfo(path:destPath),
                input: data!)
                .response { response, error in
                    if let callError = error {
                        print("uploadSessionFinish failed")
                        switch callError as CallError {
                        case .RouteError(let boxed, let requestId):
                            print("RouteError[\(requestId)]:")
                            switch boxed.unboxed as Files.UploadSessionFinishError {
                            case .Path(let writeError):
                                print("Path: ")
                                switch writeError {
                                case .MalformedPath(let malformedPathError):
                                    print("MalformedPath: \(malformedPathError)")
                                case .Conflict(let writeConflictError):
                                    print("Conflict:")
                                    switch writeConflictError {
                                    case .File:
                                        print("File")
                                    case .FileAncestor:
                                        print("FileAncestor")
                                    case .Folder:
                                        print("Folder")
                                    case .Other:
                                        print("Other")
                                    }
                                case .DisallowedName:
                                    print("DisallowedName")
                                case .InsufficientSpace:
                                    print("InsufficientSpace")
                                case .NoWritePermission:
                                    print("NoWritePermission")
                                case .Other:
                                    print("Other")
```

```
                            }
                        case .LookupFailed(let uploadSessionLookupError):
                            print("LookupFailed:")
                            switch uploadSessionLookupError {
                            case .Closed:
                                print("Closed")
                            case .IncorrectOffset(let uploadSessionOffsetError):
                                print("IncorrectOffset: \(uploadSessionOffsetError)")
                            case .NotFound:
                                print("NotFound")
                            case .NotClosed:
                                print("NotFound")
                            case .Other:
                                print("Other")
                            }
                        case .TooManySharedFolderTargets:
                            print("TooManySharedFolderTargets")
                        case .Other:
                            print("Other")
                        }
                    case .BadInputError(let message, let requestId):
                        print("BadInputError[\(requestId)]: \(message)")
                    case .HTTPError(let code, let message, let requestId):
                        print("HTTPError[\(requestId)]: \(code): \(message)")
                    case .InternalServerError(let code, let message, let requestId):
                        print("InternalServerError[\(requestId)]: \(code): \(message)")
                    case .OSError(let err):
                        print("OSError: \(err)")
                    case .RateLimitError:
                        print("RateLimitError")
                    }
                } else if let result = response {
                    print("Done!")
                    print(result)
                }
            }
        } else {
            Dropbox.authorizedClient!.files.uploadSessionAppendV2(
                cursor: Files.UploadSessionCursor(sessionId: self.sessionId, offset:
UInt64(offset)),
                input: data!)
                .response { response, error in
                    if error == nil {
                        self.offset += self.chunkSize
                        print("So far \(self.offset) bytes have been uploaded.")
                        self.uploadNextChunk()
                    } else if let callError = error {
                        print("uploadSessionAppend failed")
                        switch callError as CallError {
                        case .RouteError(let boxed, let requestId):
                            print("RouteError[\(requestId)]:")
                            switch boxed.unboxed as Files.UploadSessionLookupError {
                            case .Closed:
                                print("Closed")
                            case .IncorrectOffset(let uploadSessionOffsetError):
                                print("IncorrectOffset: \(uploadSessionOffsetError)")
                            case .NotFound:
                                print("NotFound")
                            case .NotClosed:
                                print("NotClosed")
                            case .Other:
```

```
                                print("Other")
                            }
                        case .BadInputError(let message, let requestId):
                            print("BadInputError[\(requestId)]: \(message)")
                        case .HTTPError(let code, let message, let requestId):
                            print("HTTPError[\(requestId)]: \(code): \(message)")
                        case .InternalServerError(let code, let message, let requestId):
                            print("InternalServerError[\(requestId)]: \(code): \(message)")
                        case .OSError(let err):
                            print("OSError: \(err)")
                        case .RateLimitError:
                            print("RateLimitError")
                        }
                    }
                }
            }
        }

}
```

Caricamento di un file utilizzando l'SDK Dropbox Objective-C

Questo utilizza l' SDK Dropbox Objective-C per caricare un file locale su Dropbox come "/test.txt".

```
[[client.filesRoutes uploadUrl:@"/test.txt" inputUrl:[NSURL
fileURLWithPath:@"/local/path/to/test.txt"]] response:^(DBFILESFileMetadata *metadata,
DBFILESUploadError *uploadError, DBRequestError *error) {
    if (metadata) {
        NSLog(@"The upload completed successfully.");
        NSLog(@"File metadata:");
        NSLog(@"%@", metadata);
    } else if (uploadError) {
        NSLog(@"Something went wrong with the upload:");
        NSLog(@"%@", uploadError);
    } else if (error) {
        NSLog(@"Something went wrong with the API call:");
        NSLog(@"%@", error);
    }
}];
```

Leggi Caricamento di un file online: https://riptutorial.com/it/dropbox-api/topic/409/caricamento-di-un-file

Capitolo 3: Condivisione di un file

Examples

Invito di un membro a un file condiviso utilizzando la libreria di Dropbox Java

Questo utilizza l' SDK Java Dropbox per condividere un file in "/test.txt" con un utente specifico:

```
List<MemberSelector> newMembers = new ArrayList<MemberSelector>();
MemberSelector newMember = MemberSelector.email("<EMAIL_ADDRESS_TO_INVITE>");
newMembers.add(newMember);

List<FileMemberActionResult> fileMemberActionResults =
client.sharing().addFileMember("/test.txt", newMembers);
System.out.print(fileMemberActionResults);
```

`<EMAIL_ADDRESS_TO_INVITE>` deve essere sostituito con l'indirizzo email dell'utente da invitare. Inoltre, `newMembers` è una matrice e può contenere più utenti.

Leggi Condivisione di un file online: https://riptutorial.com/it/dropbox-api/topic/8979/condivisione-di-un-file

Capitolo 4: Condivisione di una cartella

Examples

Condivisione di una cartella con ogni caso di errore gestito utilizzando la libreria SwiftyDropbox

Questo utilizza la libreria SwiftyDropbox per condividere una cartella, gestendo ogni caso di errore:

```
Dropbox.authorizedClient!.sharing.shareFolder(path: "/folder_path").response { response, error
in
    if let result = response {
        print("response: \(result)")
    } else if let callError = error {
        switch callError as CallError {
        case .BadInputError(let message, let requestId):
            print("BadInputError[\(requestId)]: \(message)")
        case .HTTPError(let code, let message, let requestId):
            print("HTTPError[\(requestId)]: \(code): \(message)")
        case .InternalServerError(let code, let message, let requestId):
            print("InternalServerError[\(requestId)]: \(code): \(message)")
        case .OSError(let err):
            print("OSError: \(err)")
        case .RateLimitError:
            print("RateLimitError")
        case .RouteError(let boxed, let requestId):
            print("RouteError[\(requestId)]:")
            switch boxed.unboxed as Sharing.ShareFolderError {
            case .BadPath(let sharePathError):
                print("BadPath: \(sharePathError)")
                switch sharePathError as Sharing.SharePathError {
                case .AlreadyShared:
                    print("AlreadyShared")
                case .ContainsSharedFolder:
                    print("ContainsSharedFolder")
                case .InsideAppFolder:
                    print("InsideAppFolder")
                case .InsideSharedFolder:
                    print("InsideSharedFolder")
                case .InvalidPath:
                    print("InvalidPath")
                case .IsAppFolder:
                    print("IsAppFolder")
                case .IsFile:
                    print("IsFile")
                case .Other:
                    print("Other")
                }
            case .EmailUnverified:
                print("EmailUnverified")
            case .TeamPolicyDisallowsMemberPolicy:
                print("TeamPolicyDisallowsMemberPolicy")
            case .Other:
                print("Other")
```

```
            }
        }
    }
}
```

Condividere una cartella usando l'arricciatura

```
    curl -X POST https://api.dropboxapi.com/2/sharing/share_folder \
        --header "Authorization: Bearer <ACCESS_TOKEN>" \
        --header "Content-Type: application/json" \
        --data "{\"path\": \"/folder_path\",\"member_policy\": \"team\",\"acl_update_policy\":
\"editors\",\"shared_link_policy\": \"members\",\"force_async\": false}"
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Invito di un membro a una cartella condivisa usando curl

```
curl -X POST https://api.dropboxapi.com/2/sharing/add_folder_member \
    --header "Authorization: Bearer <ACCESS_TOKEN>" \
    --header "Content-Type: application/json" \
    --data "{\"shared_folder_id\": \"<SHARED_FOLDER_ID\",\"members\": [{\"member\": {\".tag\":
\"email\",\"email\": \"<EMAIL_ADDRESS_TO_INVITE>\"},\"access_level\": {\".tag\":
\"editor\"}}],\"quiet\": false,\"custom_message\": \"Code examples\"}"
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

`<SHARED_FOLDER_ID>` deve essere sostituito con l'ID della cartella condivisa, ad esempio, come restituito da `/2/sharing/share_folder` o `/2/sharing/list_folders` .

`<EMAIL_ADDRESS_TO_INVITE>` deve essere sostituito con l'indirizzo email dell'utente da invitare. Inoltre, i `members` sono una matrice e possono contenere più utenti.

Invito di un membro a una cartella condivisa con ogni caso di errore gestito utilizzando la libreria SwiftyDropbox

```
let toInvite = [Sharing.AddMember(member:
Sharing.MemberSelector.Email("<EMAIL_ADDRESS_TO_INVITE>"))]

Dropbox.authorizedClient!.sharing.addFolderMember(sharedFolderId: "<SHARED_FOLDER_ID>",
members: toInvite).response { response, error in

    if (response != nil) {
        print("Invited member.")
    } else if let callError = error {
        switch callError as CallError {
        case .BadInputError(let message, let requestId):
            print("BadInputError[\(requestId)]: \(message)")
        case .HTTPError(let code, let message, let requestId):
            print("HTTPError[\(requestId)]: \(code): \(message)")
        case .InternalServerError(let code, let message, let requestId):
            print("InternalServerError[\(requestId)]: \(code): \(message)")
        case .OSError(let err):
            print("OSError: \(err)")
```

```
        case .RateLimitError:
            print("RateLimitError")
        case .RouteError(let boxed, let requestId):
            print("RouteError[\(requestId)]:")
            switch boxed.unboxed as Sharing.AddFolderMemberError {
            case .AccessError(let sharedFolderAccessError):
                    print("AccessError")
                    switch sharedFolderAccessError {
                    case .EmailUnverified:
                        print("EmailUnverified")
                    case .InvalidId:
                        print("InvalidId")
                    case .NoPermission:
                        print("NoPermission")
                    case .NotAMember:
                        print("NotAMember")
                    case .TeamFolder:
                        print("TeamFolder")
                    case .Unmounted:
                        print("Unmounted")
                    case .Other:
                        print("Other")
                }
            case .BadMember(let addMemberSelectorError):
                switch addMemberSelectorError {
                case .GroupDeleted:
                    print("GroupDeleted")
                case .GroupNotOnTeam:
                    print("GroupNotOnTeam")
                case .InvalidDropboxId(let invalidDropboxId):
                    print("InvalidDropboxId: \(invalidDropboxId)")
                case .InvalidEmail(let invalidEmail):
                    print("InvalidEmail: \(invalidEmail)")
                case .UnverifiedDropboxId(let unverifiedDropboxId):
                    print("UnverifiedDropboxId: \(unverifiedDropboxId)")
                case .Other:
                    print("Other")
                }
            case .CantShareOutsideTeam:
                print("CantShareOutsideTeam")
            case .EmailUnverified:
                print("EmailUnverified")
            case .InsufficientPlan:
                print("InsufficientPlan")
            case .NoPermission:
                print("NoPermission")
            case .RateLimit:
                print("RateLimit")
            case .TooManyMembers(let limit):
                print("TooManyMembers: \(limit)")
            case .TooManyPendingInvites(let limit):
                print("TooManyPendingInvites: \(limit)")
            case .Other:
                print("Other")
            }
        }
    }

}
```

`<SHARED_FOLDER_ID>` deve essere sostituito con l'ID della cartella condivisa, ad esempio, come

restituito da `sharing.shareFolder` o `sharing.listFolders` .

`<EMAIL_ADDRESS_TO_INVITE>` deve essere sostituito con l'indirizzo email dell'utente da invitare. Inoltre, i membri sono una matrice e possono contenere più utenti.

Condivisione di una cartella tramite HttpWebRequest in PowerShell

```
$url = "https://api.dropboxapi.com/2/sharing/share_folder"

$req = [System.Net.HttpWebRequest]::Create($url)
$req.headers["Authorization"] = "Bearer <ACCESS_TOKEN>"
$req.Method = "POST"
$req.ContentType = "application/json"
$enc = [system.Text.Encoding]::UTF8
$params = @{path="/new shared folder path"} | ConvertTo-Json -compress
$params = $enc.GetBytes($params)
$req.GetRequestStream().Write($params, 0, $params.Length)

$res = $req.GetResponse()
Write-Host "Response Status Code: "$res.StatusCode
Write-Host "Response Status Description: "$res.StatusDescription
$readStream = new-object System.IO.StreamReader $res.GetResponseStream()
$result = $readStream.ReadToEnd() | ConvertFrom-Json
Write-Host $result
$readStream.Close()
$res.Close()
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso.

Invito di un membro a una cartella condivisa tramite jQuery in JavaScript

```
$.ajax({
    url: 'https://api.dropboxapi.com/2/sharing/add_folder_member',
    type: 'POST',
    processData: false,
    data: JSON.stringify({"shared_folder_id": "84528192421","members": [{"member": {".tag":
"email","email": "justin@example.com"},"access_level": {".tag": "editor"}},{"member": {".tag":
"dropbox_id","dropbox_id": "dbid:AAEufNrMPSPe0dMQijRP0N_aZtBJRm26W4Q"},"access_level":
{".tag": "viewer"}}],"quiet": false,"custom_message": "Documentation for launch day"}),
    contentType: 'application/json',
    headers: {
        "Authorization": "Bearer <ACCESS_TOKEN>"
    },
    success: function(data) {
        console.log(data);
    },
    error: function(data) {
        console.error(data);
    }
})
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Leggi Condivisione di una cartella online: https://riptutorial.com/it/dropbox-api/topic/411/condivisione-di-una-cartella

Capitolo 5: Download di un file

Examples

Download di un file tramite arricciatura

Questo scarica un file dall'API Dropbox sul percorso remoto `/Homework/math/Prime_Numbers.txt` sul percorso locale `Prime_Numbers.txt` nella cartella corrente:

```
curl -X POST https://content.dropboxapi.com/2/files/download \
    --header "Authorization: Bearer <ACCESS_TOKEN>" \
    --header "Dropbox-API-Arg: {\"path\": \"/Homework/math/Prime_Numbers.txt\"}" \
    -o "./Prime_Numbers.txt"
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso.

Download di un file con informazioni sull'avanzamento utilizzando la libreria SwiftyDropbox

Adattato dal tutorial , utilizza la libreria SwiftyDropbox per scaricare un file, con una callback di avanzamento sul metodo di download per ottenere informazioni sull'avanzamento:

```
// Download a file
let destination : (NSURL, NSHTTPURLResponse) -> NSURL = { temporaryURL, response in
    let fileManager = NSFileManager.defaultManager()
    let directoryURL = fileManager.URLsForDirectory(.DocumentDirectory, inDomains:
.UserDomainMask)[0]
    // generate a unique name for this file in case we've seen it before
    let UUID = NSUUID().UUIDString
    let pathComponent = "\(UUID)-\(response.suggestedFilename!)"
    return directoryURL.URLByAppendingPathComponent(pathComponent)
}

Dropbox.authorizedClient!.files.download(path: "/path/to/Dropbox/file", destination:
destination)

    .progress { bytesRead, totalBytesRead, totalBytesExpectedToRead in

        print("bytesRead: \(bytesRead)")
        print("totalBytesRead: \(totalBytesRead)")
        print("totalBytesExpectedToRead: \(totalBytesExpectedToRead)")

    }

    .response { response, error in

        if let (metadata, url) = response {
            print("*** Download file ***")
            print("Downloaded file name: \(metadata.name)")
            print("Downloaded file url: \(url)")
        } else {
            print(error!)
        }
```

```
    }
```

Puoi quindi utilizzare le informazioni sull'andamento non elaborato per ripristinare l'interfaccia utente di avanzamento nella tua app.

Download di un file usando la libreria Dropbox di Python

Questo utilizza l' SDK Dropbox di Python per scaricare un file dall'API Dropbox nel percorso remoto `/Homework/math/Prime_Numbers.txt` nel file locale `Prime_Numbers.txt` :

```
import dropbox
dbx = dropbox.Dropbox("<ACCESS_TOKEN>")

with open("Prime_Numbers.txt", "wb") as f:
    metadata, res = dbx.files_download(path="/Homework/math/Prime_Numbers.txt")
    f.write(res.content)
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso.

Download di un file con ogni caso di errore gestito utilizzando la libreria SwiftyDropbox

```
Dropbox.authorizedClient!.files.download(path: path, destination: destination).response {
response, error in
    if let (metadata, url) = response {
        print("*** Download file ***")
        print("Downloaded file name: \(metadata.name)")
        print("Downloaded file url: \(url)")
    } else if let callError = error {
        switch callError as CallError {
            case .RouteError(let boxed, let requestId):
                print("RouteError[\(requestId)]:")
                switch boxed.unboxed as Files.DownloadError {
                case .Path(let fileLookupError):
                    print("PathError: ")
                    switch fileLookupError {
                    case .MalformedPath(let malformedPathError):
                        print("MalformedPath: \(malformedPathError)")
                    case .NotFile:
                        print("NotFile")
                    case .NotFolder:
                        print("NotFolder")
                    case .NotFound:
                        print("NotFound")
                    case .RestrictedContent:
                        print("RestrictedContent")
                    case .Other:
                        print("Other")
                    }
                case .Other:
                    print("Other")
            }
            case .BadInputError(let message, let requestId):
                print("BadInputError[\(requestId)]: \(message)")
```

```
            case .HTTPError(let code, let message, let requestId):
                print("HTTPError[\(requestId)]: \(code): \(message)")
            case .InternalServerError(let code, let message, let requestId):
                print("InternalServerError[\(requestId)]: \(code): \(message)")
            case .OSError(let err):
                print("OSError: \(err)")
            case .RateLimitError:
                print("RateLimitError")
            }

    }
}
```

Download di un file tramite curl in C ++

```
#include <stdio.h>
#include <curl/curl.h>

int main (int argc, char *argv[])
{
      CURL *curl;
      CURLcode res;

      /* In windows, this will init the winsock stuff */
      curl_global_init(CURL_GLOBAL_ALL);

      /* get a curl handle */
      curl = curl_easy_init();
      if(curl) {

            printf ("Running curl test.\n");

            struct curl_slist *headers=NULL; /* init to NULL is important */
            headers = curl_slist_append(headers, "Authorization: Bearer <ACCESS_TOKEN>");
            headers = curl_slist_append(headers, "Content-Type:");
            headers = curl_slist_append(headers, "Dropbox-API-Arg: {\"path\":\"/test.txt\"}");
            curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);

            curl_easy_setopt(curl, CURLOPT_URL,
"https://content.dropboxapi.com/2/files/download");
            curl_easy_setopt(curl, CURLOPT_POSTFIELDS, "");

            /* Perform the request, res will get the return code */
            res = curl_easy_perform(curl);
            /* Check for errors */
            if(res != CURLE_OK)
                  fprintf(stderr, "curl_easy_perform() failed: %s\n",
                    curl_easy_strerror(res));

            /* always cleanup */
              curl_easy_cleanup(curl);

              printf ("\nFinished curl test.\n");

      }
      curl_global_cleanup();

      printf ("Done!\n");
      return 0;
```

```
}
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso.

Download di un pezzo di un file tramite arricciatura utilizzando le richieste di recupero intervallo

Questo scarica solo un pezzo di un file, utilizzando le richieste di recupero di intervalli , dall'API Dropbox nel percorso remoto `/Homework/math/Prime_Numbers.txt` al percorso locale `Prime_Numbers.txt.partial` nella cartella corrente:

```
curl -X GET https://content.dropboxapi.com/2/files/download \
    --header "Authorization: Bearer <ACCESS_TOKEN>" \
    --header "Dropbox-API-Arg: {\"path\": \"/Homework/math/Prime_Numbers.txt\"}" \
    --header "Range:bytes=0-10" \
    -o "./Prime_Numbers.txt.partial"
```

L'intervallo specificato, `0-10` , indica all'API di restituire solo i primi 10 byte. Se l'API risponde con un codice di stato 206, ciò indica che l'intervallo è stato accettato e che è stato restituito solo l'intervallo di richieste parziale.

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso.

Download di un file utilizzando la libreria Dropbox .NET

Questo utilizza l' SDK Dropbox .NET per scaricare un file dall'API Dropbox nel percorso remoto `/Homework/math/Prime_Numbers.txt` nel file locale `Prime_Numbers.txt` :

```
using (var response = await client.Files.DownloadAsync("/Homework/math/Prime_Numbers.txt"))
{
    using (var fileStream = File.Create("Prime_Numbers.txt"))
    {
        (await response.GetContentAsStreamAsync()).CopyTo(fileStream);
    }

}
```

Download di un file utilizzando la libreria Dropbox .NET con monitoraggio dell'avanzamento

Questo utilizza l' SDK di Dropbox .NET per scaricare un file dall'API Dropbox sul `path` remoto del file locale "Test", mentre vengono monitorati i progressi:

```
var response = await client.Files.DownloadAsync(path);
ulong fileSize = response.Response.Size;
const int bufferSize = 1024 * 1024;

var buffer = new byte[bufferSize];
```

```
using (var stream = await response.GetContentAsStreamAsync())
{
    using (var file = new FileStream("Test", FileMode.OpenOrCreate))
    {
        var length = stream.Read(buffer, 0, bufferSize);

        while (length > 0)
        {
            file.Write(buffer, 0, length);
            var percentage = 100 * (ulong)file.Length / fileSize;
            // Update progress bar with the percentage.
            // progressBar.Value = (int)percentage
            Console.WriteLine(percentage);

            length = stream.Read(buffer, 0, bufferSize);
        }
    }
}
```

Download di un file con metadati tramite Richieste in PHP

```
$response = Requests::post("https://content.dropboxapi.com/2/files/download", array(
    'Authorization' => "Bearer <ACCESS_TOKEN>",
    'Dropbox-Api-Arg' => json_encode(array('path' => '/test.txt')),
));

$fileContent = $response->body;
$metadata = json_decode($response->headers['Dropbox-Api-Result'], true);

echo "File " . $metadata["name"] . " has the rev " . $metadata["rev"] . ".\n";
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Download di un file con metadati tramite curl in PHP

```
<?php

function dbx_get_file($token, $in_filepath, $out_filepath)
    {
    $out_fp = fopen($out_filepath, 'w+');
    if ($out_fp === FALSE)
        {
        echo "fopen error; can't open $out_filepath\n";
        return (NULL);
        }

    $url = 'https://content.dropboxapi.com/2/files/download';

    $header_array = array(
        'Authorization: Bearer ' . $token,
        'Content-Type:',
        'Dropbox-API-Arg: {"path":"' . $in_filepath . '"}'
    );

    $ch = curl_init();

    curl_setopt($ch, CURLOPT_URL, $url);
```

```
    curl_setopt($ch, CURLOPT_POST, TRUE);
    curl_setopt($ch, CURLOPT_HTTPHEADER, $header_array);
    curl_setopt($ch, CURLOPT_FILE, $out_fp);

    $metadata = null;
    curl_setopt($ch, CURLOPT_HEADERFUNCTION, function ($ch, $header) use (&$metadata)
        {
        $prefix = 'dropbox-api-result:';
        if (strtolower(substr($header, 0, strlen($prefix))) === $prefix)
            {
            $metadata = json_decode(substr($header, strlen($prefix)), true);
            }
        return strlen($header);
        }
    );

    $output = curl_exec($ch);

    if ($output === FALSE)
        {
        echo "curl error: " . curl_error($ch);
        }

    curl_close($ch);
    fclose($out_fp);

    return($metadata);
    } // dbx_get_file()

$metadata = dbx_get_file("<ACCESS_TOKEN>", '/test.txt', 'test.txt');
echo "File " . $metadata['name'] . " has the rev " . $metadata['rev'] . ".\n";

?>
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Download di un file utilizzando la libreria Dropbox Java

Questo utilizza l' SDK Java Dropbox per scaricare un file dall'API Dropbox sul percorso remoto `/Homework/math/Prime_Numbers.txt` nel file locale `Prime_Numbers.txt` :

```
String localPath = "Prime_Numbers.txt";
OutputStream outputStream = new FileOutputStream(localPath);
FileMetadata metadata = client.files()
        .downloadBuilder("/Homework/math/Prime_Numbers.txt")
        .download(outputStream);
```

Download di un file utilizzando la libreria Dropbox Objective-C con tracciamento dell'avanzamento

Questo utilizza l' SDK Dropbox Objective-C per scaricare un file da Dropbox in "/test.txt".

```
[[[client.filesRoutes downloadData:@"/test.txt"] response:^(DBFILESFileMetadata *metadata,
DBFILESDownloadError *downloadError, DBRequestError *error, NSData *fileData) {
    if (metadata) {
        NSLog(@"The download completed successfully.");
```

```
        NSLog(@"File metadata:");
        NSLog(@"%@", metadata);
        NSLog(@"File data length:");
        NSLog(@"%lu", (unsigned long)[fileData length]);
    } else if (downloadError) {
        NSLog(@"Something went wrong with the data:");
        NSLog(@"%@", downloadError);
    } else if (error) {
        NSLog(@"Something went wrong with the API call:");
        NSLog(@"%@", error);
    }
}] progress:^(int64_t bytesWritten, int64_t totalBytesWritten, int64_t
totalBytesExpectedToWrite) {
    // Here we can monitor the progress of the transfer:
    NSLog(@"bytesWritten: %lld, totalBytesWritten: %lld, totalBytesExpectedToWrite: %lld",
bytesWritten, totalBytesWritten, totalBytesExpectedToWrite);
}];
```

Leggi Download di un file online: https://riptutorial.com/it/dropbox-api/topic/408/download-di-un-file

Capitolo 6: Elenco di una cartella

Examples

Elenco della cartella radice tramite arricciatura

Elenca la cartella root, identificata dalla stringa vuota `""` per Dropbox API v2, usando curl, usando / files / list_folder :

```
curl -X POST https://api.dropboxapi.com/2/files/list_folder \
    --header "Authorization: Bearer <ACCESS_TOKEN>" \
    --header "Content-Type: application/json" \
    --data "{\"path\": \"\"}"
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Nota che la risposta potrebbe contenere `has_more=true` , nel qual caso la tua app dovrebbe richiamare su / files / list_folder / continuare a continuare ad ottenere più voci.

Elenco della cartella radice tramite curl in PHP e l'estensione cURL

```
<?php

$parameters = array('path' => '','include_deleted' => true,'recursive' => true);

$headers = array('Authorization: Bearer <ACCESS_TOKEN>',
                 'Content-Type: application/json');

$curlOptions = array(
        CURLOPT_HTTPHEADER => $headers,
        CURLOPT_POST => true,
        CURLOPT_POSTFIELDS => json_encode($parameters),
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_VERBOSE => true
    );

$ch = curl_init('https://api.dropboxapi.com/2/files/list_folder');
curl_setopt_array($ch, $curlOptions);

$response = curl_exec($ch);
echo $response;

curl_close($ch);

?>
```

Nota che la risposta potrebbe contenere `has_more=true` , nel qual caso la tua app dovrebbe richiamare su / files / list_folder / continuare a continuare ad ottenere più voci.

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Elenco della cartella radice usando la libreria SwiftyDropbox, distinguendo i file e le cartelle nella risposta

```
Dropbox.authorizedClient!.files.listFolder(path: "").response { response, error in
    print("*** List folder ***")
    if let result = response {
        print("Folder contents:")
        for entry in result.entries {
            print(entry.name)
            if let file = entry as? Files.FileMetadata {
                print("\tThis is a file with path: \(file.pathLower) and size: \(file.size)")
            } else if let folder = entry as? Files.FolderMetadata {
                print("\tThis is a folder with path: \(folder.pathLower)")
            }
        }
    } else if let callError = error {
        switch callError {
        case .RouteError(let boxed, _):
            switch boxed.unboxed {
            case .Path(let lookupError):
                print("lookupError:")
                print(lookupError)
            case .Other:
                print("Other")
            }
        default:
            print("default")
        }
    }
}
```

Nota che la risposta potrebbe contenere `ListFolderResult.hasMore=true` , nel qual caso la tua app dovrebbe richiamare usando `listFolderContinue` per continuare a ricevere più voci.

Tentativo di elencare una cartella inesistente utilizzando la libreria SwiftyDropbox, come esempio di gestione degli errori

```
// List folder
Dropbox.authorizedClient!.files.listFolder(path: "/nonexistantpath").response { response,
error in
    print("*** List folder ***")
    if let result = response {
        print("Folder contents:")
        for entry in result.entries {
            print(entry.name)
        }
    } else if let callError = error {
        switch callError {
        case .RouteError(let boxed, _):
            switch boxed.unboxed {
            case .Path(let lookupError):
                print("lookupError:")
                print(lookupError)
            case .Other:
                print("Other")
            }
```

```
        default:
            print("default")
        }
    }
}
```

Elenco dei file usando l'estensione PHP e cURL

```
<?php

$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, "https://api.dropboxapi.com/2/files/list_folder");
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_CAINFO, "cacert.pem");
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
curl_setopt($ch, CURLOPT_POSTFIELDS, "{\"path\":\"/<FOLDER PATH>\"}");
curl_setopt($ch, CURLOPT_POST, 1);

$headers = array();
$headers[] = "Authorization: Bearer <ACCESS_TOKEN>";
$headers[] = "Content-Type: application/json";
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);

$result = curl_exec($ch);
if (curl_errno($ch)) {
    echo 'Error:' . curl_error($ch);
}
curl_close ($ch);


$json = json_decode($result, true);
foreach ($json['entries'] as $data) {
    echo 'File Name: ' . $data['name'];
}

?>
```

Nota che la risposta potrebbe contenere `has_more=true` , nel qual caso la tua app dovrebbe richiamare su / files / list_folder / continuare a continuare ad ottenere più voci.

`<ACCESS_TOKEN>` **deve essere sostituito** con il token di accesso OAuth 2.

Leggi Elenco di una cartella online: https://riptutorial.com/it/dropbox-api/topic/412/elenco-di-una-cartella

Capitolo 7: Ottenere i metadati del file

Examples

Ottieni i metadati del file per un file, incluse le informazioni multimediali, usando la libreria SwiftyDropbox

```
Dropbox.authorizedClient!.files.getMetadata(path: "/test.jpg", includeMediaInfo:
true).response { response, error in
    if let result = response as? Files.FileMetadata {
        print(result.name)

        if result.mediaInfo != nil {
            switch result.mediaInfo! as Files.MediaInfo {
            case .Pending:
                print("Media info is pending...")
            case .Metadata(let mediaMetadata):
                print(mediaMetadata.dimensions)
                print(mediaMetadata.location)
                print(mediaMetadata.timeTaken)
            }
        }
    } else {
        print(error!)
    }
}
```

Ottieni i metadati del file per un file, incluse le informazioni multimediali, usando arricciatura

```
curl -X POST https://api.dropboxapi.com/2/files/get_metadata \
    --header "Authorization: Bearer <ACCESS_TOKEN>" \
    --header "Content-Type: application/json" \
    --data "{\"path\": \"/test.jpg\",\"include_media_info\": true}"
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Gestione dell'errore durante l'acquisizione dei metadati per un percorso non esistente mediante la libreria Dropbox .NET

Questo esempio utilizza la libreria Dropbox .NET per provare a ottenere i metadati per un elemento in un determinato percorso e verifica un errore `NotFound` :

```
try {
    var metadata = await this.client.Files.GetMetadataAsync("/non-existant path");
    Console.WriteLine(metadata.Name);
} catch (Dropbox.Api.ApiException<Dropbox.Api.Files.GetMetadataError> e) {

    if (e.ErrorResponse.IsPath) {
        var pathError = e.ErrorResponse.AsPath.Value;
```

```
        if (pathError.IsNotFound) {
            Console.WriteLine ("File or folder not found.");
        } else {
            Console.WriteLine (pathError);
        }
    } else {
        Console.WriteLine (e.ErrorResponse);
    }
}
```

Leggi Ottenere i metadati del file online: https://riptutorial.com/it/dropbox-api/topic/413/ottenere-i-metadati-del-file

Capitolo 8: Ottenere informazioni sull'account

Examples

Ottenere informazioni sull'account per l'utente collegato tramite arricciatura

```
curl -X POST https://api.dropboxapi.com/2/users/get_current_account \
    --header "Authorization: Bearer <ACCESS_TOKEN>"
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso.

Ottenere informazioni sull'account per l'utente collegato tramite curl in C ++

```
#include <stdio.h>
#include <curl/curl.h>


int main (int argc, char *argv[])
{
      CURL *curl;
      CURLcode res;

      /* In windows, this will init the winsock stuff */
      curl_global_init(CURL_GLOBAL_ALL);

      /* get a curl handle */
      curl = curl_easy_init();
      if(curl) {

            printf ("Running curl test.\n");

            struct curl_slist *headers=NULL; /* init to NULL is important */
            headers = curl_slist_append(headers, "Authorization: Bearer <ACCESS_TOKEN>");
            headers = curl_slist_append(headers, "Content-Type: application/json");
            curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);

            curl_easy_setopt(curl, CURLOPT_URL,
"https://api.dropbox.com/2/users/get_current_account");
            curl_easy_setopt(curl, CURLOPT_POSTFIELDS, "null");

            /* Perform the request, res will get the return code */
            res = curl_easy_perform(curl);

            /* Check for errors */
            if(res != CURLE_OK)
                  fprintf(stderr, "curl_easy_perform() failed: %s\n",
                    curl_easy_strerror(res));

            /* always cleanup */
              curl_easy_cleanup(curl);

              printf ("\nFinished curl test.\n");
```

```
      }
      curl_global_cleanup();

      printf ("Done!\n");
      return 0;

}
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso.

Ottenere informazioni sull'account usando la libreria Dropbox di Python

Questo utilizza l' SDK Dropbox di Python per ottenere le informazioni sull'account dell'utente dall'API Dropbox.

```
import dropbox
dbx = dropbox.Dropbox("<ACCESS_TOKEN>")
dbx.users_get_current_account()
```

`<ACCESS_TOKEN>` dovrebbe essere sostituito con il token di accesso.

Ottenere informazioni sull'utilizzo dello spazio per l'utente collegato tramite curl in PHP

```
<?php

$headers = array("Authorization: Bearer <ACCESS_TOKEN>",
                 "Content-Type: application/json");

$ch = curl_init('https://api.dropboxapi.com/2/users/get_space_usage');
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, "null");
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);

curl_close($ch);
echo $response;

?>
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Ottenere informazioni sull'account per l'utente collegato tramite HttpWebRequest in PowerShell

```
$url = "https://api.dropboxapi.com/2/users/get_current_account"

$req = [System.Net.HttpWebRequest]::Create($url)
$req.headers["Authorization"] = "Bearer <ACCESS_TOKEN>"
$req.Method = "POST"
```

```
$res = $req.GetResponse()
Write-Host "Response Status Code: "$res.StatusCode
Write-Host "Response Status Description: "$res.StatusDescription
$readStream = new-object System.IO.StreamReader $res.GetResponseStream()
$result = $readStream.ReadToEnd() | ConvertFrom-Json
Write-Host $result
$readStream.Close()
$res.Close()
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso.

Ottenere informazioni sull'account tramite jQuery in JavaScript

```
jQuery.ajax({
    url: 'https://api.dropboxapi.com/2/users/get_current_account',
    type: 'POST',
    headers: {
        "Authorization": "Bearer <ACCESS_TOKEN>"
    },
    success: function (data) {
        console.log(data);
    },
    error: function (error) {
        console.log(error);
    }
})
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Ottenere informazioni sull'account utilizzando la libreria Dropbox Objective-C

Questo utilizza l' SDK Dropbox Objective-C per ottenere le informazioni sull'account dell'utente dall'API Dropbox.

```
[[client.usersRoutes getCurrentAccount] response:^(DBUSERSFullAccount *account, DBNilObject
*_, DBRequestError *error) {
    if (account) {
        NSLog(@"%@", account);
    } else if (error) {
        NSLog(@"%@", error);
    }
}];
```

Leggi Ottenere informazioni sull'account online: https://riptutorial.com/it/dropbox-api/topic/410/ottenere-informazioni-sull-account

Capitolo 9: Ottenere un collegamento condiviso per un file o una cartella

Examples

Creazione di un collegamento condiviso per una cartella usando la libreria Dropbox di Python

Questo utilizza l' SDK Dropbox di Python per creare un collegamento condiviso per una cartella:

```
import dropbox
dbx = dropbox.Dropbox("<ACCESS_TOKEN>")
shared_link_metadata = dbx.sharing_create_shared_link_with_settings("/Testing")
print shared_link_metadata.url
```

`<ACCESS_TOKEN>` dovrebbe essere sostituito con il token di accesso.

Recupero di un collegamento condiviso esistente per un file specifico usando curl

```
curl -X POST https://api.dropboxapi.com/2/sharing/list_shared_links \
    --header "Authorization: Bearer <ACCESS_TOKEN>" \
    --header "Content-Type: application/json" \
    --data "{\"path\": \"/test.txt\", \"direct_only\": true}"
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Creazione di un collegamento condiviso per un file utilizzando la libreria SwiftyDropbox

```
Dropbox.authorizedClient!.sharing.createSharedLink(path: "/test.txt").response({ response,
error in
    if let link = response {
        print(link.url)
    } else {
        print(error!)
    }
})
```

Creare un collegamento condiviso per un file usando curl

```
curl -X POST https://api.dropboxapi.com/2/sharing/create_shared_link_with_settings \
    --header "Authorization: Bearer <ACCESS_TOKEN>" \
    --header "Content-Type: application/json" \
    --data "{\"path\": \"/Prime_Numbers.txt\",\"settings\": {\"requested_visibility\":
\"public\"}}"
```

`<ACCESS_TOKEN>` dovrebbe essere sostituito con il token di accesso.

Creazione di un collegamento condiviso per un file con impostazioni di scadenza e visibilità utilizzando la libreria Dropbox di Python

Questo utilizza l' SDK Dropbox di Python per creare un collegamento condiviso per un file e fornisce anche una visibilità e una scadenza richieste nelle impostazioni :

```
import datetime

import dropbox

dbx = dropbox.Dropbox("<ACCESS_TOKEN>")

expires = datetime.datetime.now() + datetime.timedelta(days=30)
requested_visibility = dropbox.sharing.RequestedVisibility.team_only
desired_shared_link_settings =
dropbox.sharing.SharedLinkSettings(requested_visibility=requested_visibility, expires=expires)

shared_link_metadata = dbx.sharing_create_shared_link_with_settings("/test.txt",
settings=desired_shared_link_settings)

print(shared_link_metadata)
```

`<ACCESS_TOKEN>` dovrebbe essere sostituito con il token di accesso.

Creazione di un collegamento condiviso per un file utilizzando la libreria Java Dropbox

Questo utilizza l' SDK Java Dropbox per creare un collegamento condiviso per un file nel percorso Dropbox /test.txt:

```
try {
    SharedLinkMetadata sharedLinkMetadata =
client.sharing().createSharedLinkWithSettings("/test.txt");
    System.out.println(sharedLinkMetadata.getUrl());
} catch (CreateSharedLinkWithSettingsErrorException ex) {
    System.out.println(ex);
} catch (DbxException ex) {
    System.out.println(ex);
}
```

Ciò presuppone che il `client` sia un oggetto `DbxClientV2` preesistente e autorizzato e che `DbxClientV2` alcune eccezioni di base per la gestione dell'output.

Ottenere un collegamento condiviso per un file usando la libreria Dropbox .NET

Questo esempio utilizza la libreria Dropbox .NET per ottenere un collegamento condiviso per un file, creando uno nuovo o recuperando uno esistente:

```
SharedLinkMetadata sharedLinkMetadata;
try {
    sharedLinkMetadata = await this.client.Sharing.CreateSharedLinkWithSettingsAsync (path);
} catch (ApiException<CreateSharedLinkWithSettingsError> err) {
    if (err.ErrorResponse.IsSharedLinkAlreadyExists) {
        var sharedLinksMetadata = await this.client.Sharing.ListSharedLinksAsync (path, null,
true);
        sharedLinkMetadata = sharedLinksMetadata.Links.First();
    } else {
        throw err;
    }
}
Console.WriteLine (sharedLinkMetadata.Url);
```

Creare un collegamento condiviso per un file usando curl in PHP

```
<?php

$parameters = array('path' => '/test.txt');

$headers = array('Authorization: Bearer <ACCESS_TOKEN>',
                 'Content-Type: application/json');

$curlOptions = array(
        CURLOPT_HTTPHEADER => $headers,
        CURLOPT_POST => true,
        CURLOPT_POSTFIELDS => json_encode($parameters),
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_VERBOSE => true
    );

$ch = curl_init('https://api.dropboxapi.com/2/sharing/create_shared_link_with_settings');
curl_setopt_array($ch, $curlOptions);

$response = curl_exec($ch);
echo $response;

curl_close($ch);

?>
```

`<ACCESS_TOKEN>` deve essere sostituito con il token di accesso OAuth 2.

Recupero di un collegamento condiviso esistente per un file specifico utilizzando la libreria Dropbox Java

Questo utilizza l' SDK Java Dropbox per recuperare in modo specifico un collegamento condiviso esistente per /Testing/test.txt:

```
ListSharedLinksResult listSharedLinksResult = client.sharing()
        .listSharedLinksBuilder()
        .withPath("/Testing/test.txt").withDirectOnly(true)
        .start();
System.out.println(listSharedLinksResult.getLinks());
```

Leggi Ottenere un collegamento condiviso per un file o una cartella online: https://riptutorial.com/it/dropbox-api/topic/414/ottenere-un-collegamento-condiviso-per-un-file-o-una-cartella

Titoli di coda

S. No	Capitoli	Contributors
1	Introduzione a Dropbox API	Community, eckes, Greg
2	Caricamento di un file	Greg, smarx, user277754
3	Condivisione di un file	Greg
4	Condivisione di una cartella	Greg
5	Download di un file	Cristiam Mercado, Greg, Tom
6	Elenco di una cartella	Greg, Jens Kohl, TheExeLab
7	Ottenere i metadati del file	Greg
8	Ottenere informazioni sull'account	Greg
9	Ottenere un collegamento condiviso per un file o una cartella	Greg