

1.0

Boxit

Dropbox support in your Unity3D application

echo17

Table of Contents

Table of Contents	ii
1. Overview	1
2. Create a Dropbox Account	2
2.1 Create a New Account	2
3. Create a Dropbox App	5
3.1 Create an App	5
4. Set Up the Unity Environment	10
4.1 Create / Open Project	10
4.2 Download Boxit Plugin	10
4.3 Create a Boxit Client	12
5. Asynchronous Functions	15
5.1 Example	15
5.2 Benefits	17
5.3 Delegates	18
SuccessDelegate	18
FailureDelegate	20
WarningDelegate	21
ProgressDelegate	22
6. Linking and Unlinking	24
6.1 Linking	24
Linking Function	24

Linking Process	25
6.2 Unlinking	26
Unlink Function	26
7. Boxit Functions	28
7.1 GetAccountInfo	28
7.2 GetMetaData	29
7.3 DownloadFile	31
7.4 UploadFile	33
7.5 GetRevisions	35
7.6 Restore	37
7.7 Search	38
7.8 GetShareLink	40
7.9 GetMedia	42
7.10 GetCopyRef	43
7.11 DownloadThumbnail	45
7.12 CopyFileFromPath	46
7.13 CopyFileFromCopyRef	48
7.14 CreateFolder	50
7.15 Delete	51
7.16 Move	53
7.17 GetDelta	54
7.18 DownloadFileIfRemoteNewer	56
7.19 UploadFileIfLocalNewer	57
7.20 SyncFile	59
8. Advanced Syncing	61
9. Boxit Data Structures	63
9.1 Global Variables	63
ROOT	63
THUMBNAIL_FORMAT	63
THUMBNAIL_SIZE	64
9.2 oAuthToken	64

9.3 AccountInfo 65

9.4 QuotaInfo 65

9.5 MetaData 66

9.6 Delta 67

9.7 DeltaEntry 68

9.8 ShareLink 69

9.9 CopyRef 70

1.

Overview

Boxit is a plugin for Unity3D that allows you to connect to a Dropbox account. This can be used for numerous purposes, including:

- Backing up game data, such as stats or level progress
- Sharing game data across multiple devices, like an iPhone and an iPad, so game play can continue wherever you are
- Backing up application data, such as a database or text files
- Storing or retrieving photos
- Many more!

2.

Create a Dropbox Account

Before you can use Boxit in your application, you will need to have an active Dropbox account. This can be set up on Dropbox's website at [Dropbox.com](https://dropbox.com).

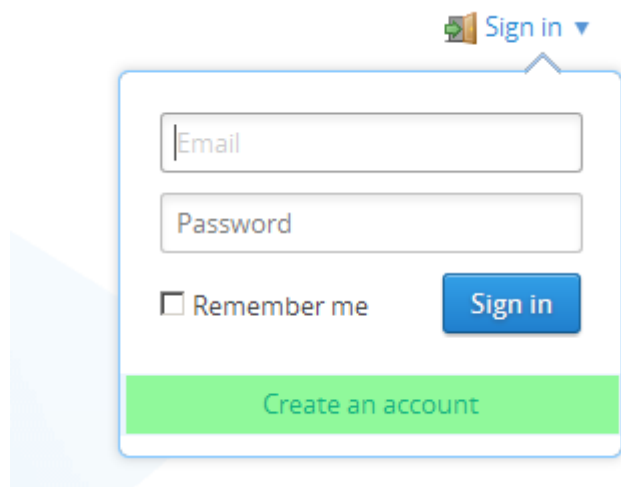


If you already have a Dropbox account, you can skip this chapter.

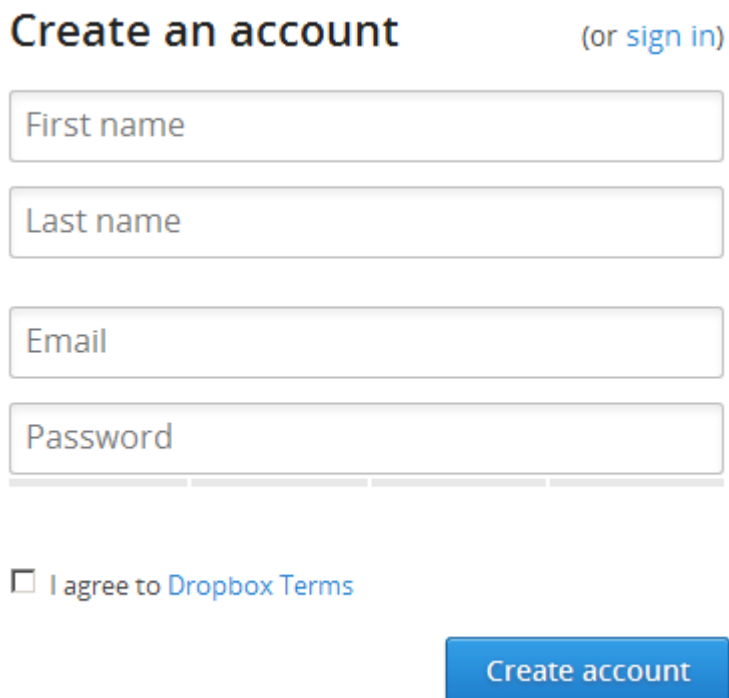
Any user that uses your application with Boxit will also need a Dropbox account, so you will need to either direct them to the Dropbox website, or provide instructions on how to do this.

2.1 Create a New Account

In a web browser, go to [Dropbox.com](https://dropbox.com). Click on the arrow next to the [Sign in](#) link in the upper right corner. Click on the [Create an Account](#) button.

Figure 2-1 Create AccountA screenshot of a web application showing a sign-in modal. At the top, there is a 'Sign in' link with a dropdown arrow. Below it, the modal contains an 'Email' input field, a 'Password' input field, a 'Remember me' checkbox, and a blue 'Sign in' button. At the bottom of the modal is a green button labeled 'Create an account'.

Fill in the account details for your new account. Be sure to agree to the Dropbox Terms.

Figure 2-2 Fill in Account InformationA screenshot of the 'Create an account' form on the Dropbox website. The form has the title 'Create an account' and a link '(or sign in)'. It includes four input fields: 'First name', 'Last name', 'Email', and 'Password'. Below these fields is a checkbox labeled 'I agree to Dropbox Terms'. At the bottom right is a blue button labeled 'Create account'.

After you create the account, Dropbox will download the program to your machine. This is not required

to use Boxit, but is a nice tool. You can now go to your home by clicking on the Dropbox logo in the upper left.

Figure 2-3 Go Home



3.

Create a Dropbox App

Before you can use Boxit, you will need to have a developer App set up in your Dropbox environment. This is only necessary for developers and is not required by the users of your application.

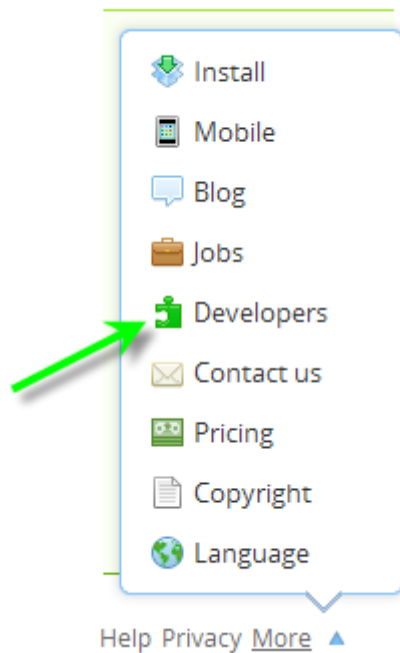


If you already have an app set up, you can skip this chapter.

3.1 Create an App

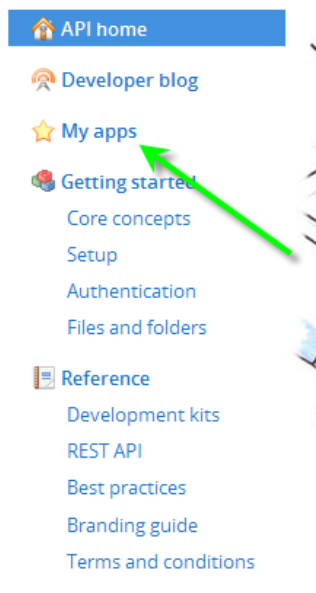
To get to the developer section of Dropbox, click on the arrow next to the [More](#) link at the bottom left of the page. Then click on the [Developers](#) link.

Figure 3-1 Developers



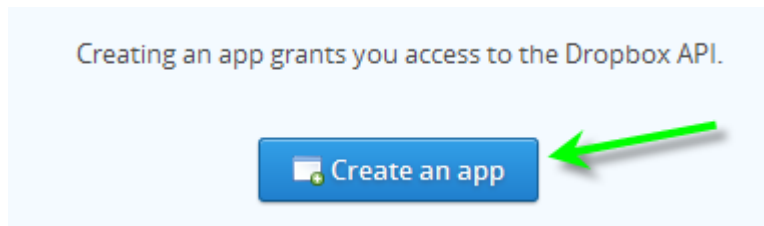
In the developer section, click on the [My Apps](#) link.

Figure 3-2 My Apps



If you don't have any apps yet, you will see the [Create an App](#) button. Click on this.

Figure 3-3 Create an App



At some point, you will be asked to agree to the Dropbox developer agreement.

Figure 3-4 User Agreement

Dropbox API Access

Before you can access the Dropbox API, you must agree to the Dropbox Developer Terms and Conditions.

Dropbox App Developer Terms and Conditions

Last Updated: 5/1/2010

These Dropbox App Developer Terms and Conditions ("Terms") constitute a legal agreement between Dropbox, Inc ("Dropbox") and you ("you"). Please read the [Terms](#), our [Privacy Policy](#) and all policies mentioned in the [Terms](#) (which are incorporated herein by reference) carefully.

YOU ACKNOWLEDGE AND AGREE THAT, BY CLICKING ON THE "I AGREE" OR "I ACCEPT" BUTTON OR BY CREATING AN APP YOU ARE INDICATING THAT YOU HAVE READ, UNDERSTAND AND AGREE TO BE BOUND BY THESE TERMS. IF YOU DO NOT AGREE TO THESE TERMS, THEN YOU HAVE NO RIGHT TO ACCESS OR USE THE DROPBOX DEVELOPER WEBSITE ("SITE"), DROPBOX KEYS OR CREATE AN APP. If you accept or agree to these Terms on behalf of a company or other legal entity, you represent and warrant that you have the authority to bind that company or other legal entity to these Terms and, in such event, "you" and "your" will refer and apply to that company or other legal entity.

1. Definitions.

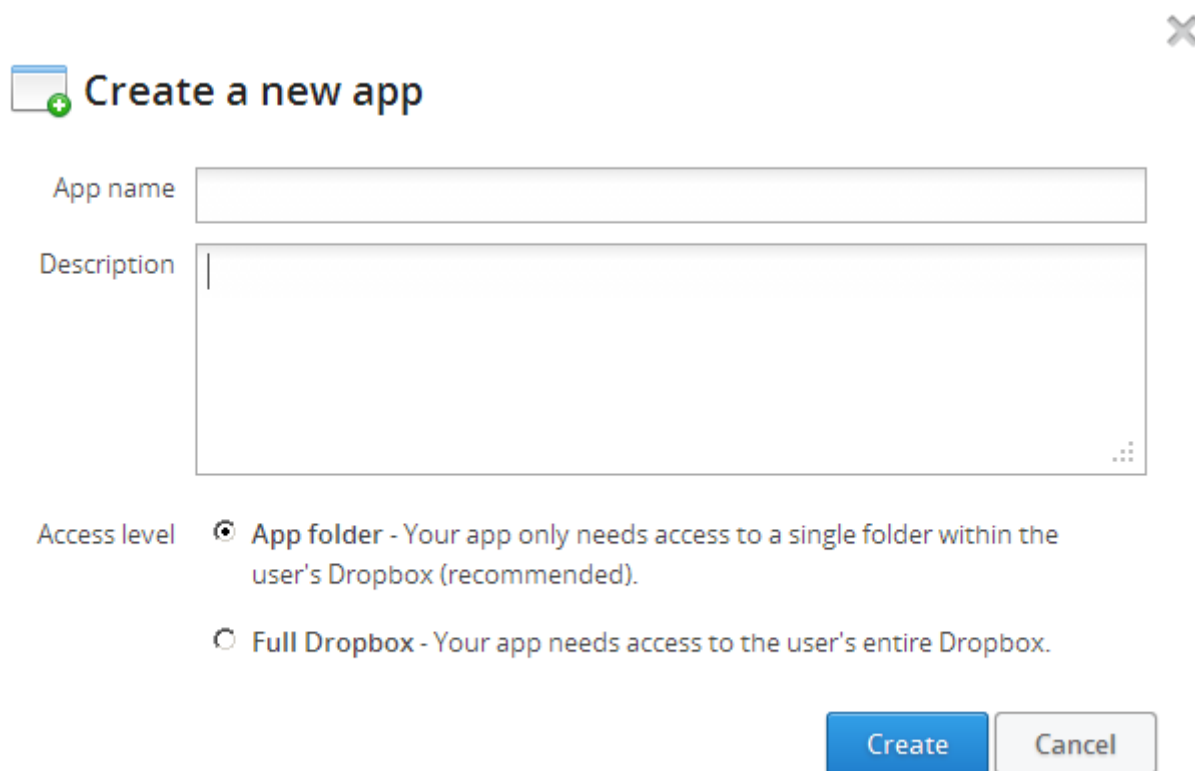
App
means an application developed by you as a Dropbox Developer that functions or is compatible with the Dropbox Services, accessed and available only via a Third Party Platform, and hosted by you or by a third party on your

☐ I agree with the Dropbox Developer terms and conditions.

Submit

Fill in your application's properties.

Figure 3-5 Create App Properties



The screenshot shows a modal window titled "Create a new app" with a close button (X) in the top right corner. The window contains the following elements:

- App name:** A text input field.
- Description:** A larger text area for describing the app.
- Access level:** Two radio button options:
 - ☒ **App folder** - Your app only needs access to a single folder within the user's Dropbox (recommended).
 - ☐ **Full Dropbox** - Your app needs access to the user's entire Dropbox.
- Buttons:** "Create" (blue) and "Cancel" (grey) buttons at the bottom right.

Be sure you set your access level here, because it cannot be changed later.

- **App Folder:** Your app will only be allowed access to its own folder and sub folders. You can't modify any files or folders outside of this structure.
- **Full Dropbox:** Your app will have access to any files or folders in the Dropbox account.

After you create your app, you can see its properties. Be sure to record the **App Key** and **App Secret**, as you will need these when developing with Boxit. You can always come back to Dropbox to get these keys later.

Figure 3-6 App Properties**Unique App**

App name	<input type="text" value="Unique App"/>
App status	Development (Apply for production status)
App key	<input type="text" value="8f85a213-80e0-4800-b900-000000000000"/>
App secret	<input type="text" value="8f85a213-80e0-4800-b900-000000000000"/>
Access type	App folder
Name of app folder	<input type="text" value="Unique App"/>



If you set your access type to App Folder, you don't need to create the application's folder. Dropbox will do this automatically for you when you upload or attempt to access your first remote file.

4.

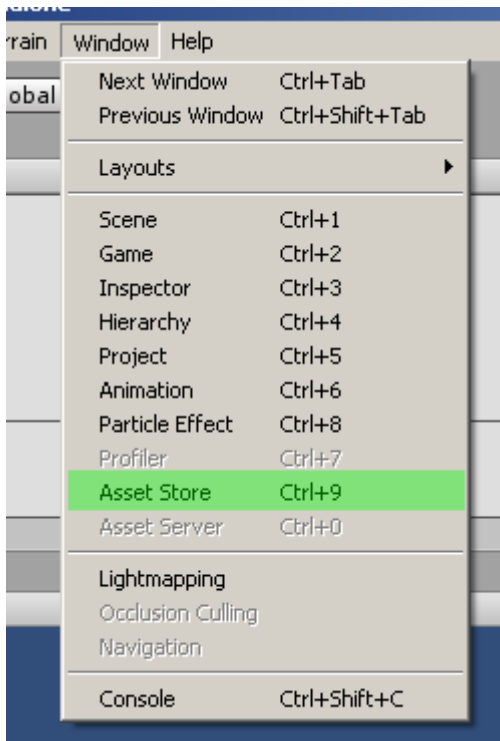
Set Up the Unity Environment

4.1 Create / Open Project

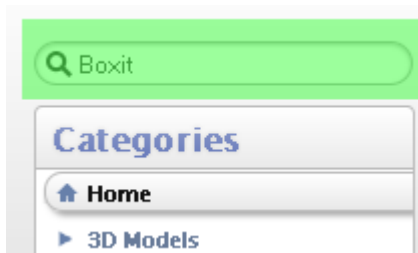
Create a new Unity project or open an existing one to import the Boxit plugin.

4.2 Download Boxit Plugin

To obtain Boxit, you must download it from the Unity Asset store. To open the store, go to the Unity menu [Window > Asset Store](#).

Figure 4-1 Asset Store

In the Unity Asset Store, search for Boxit.

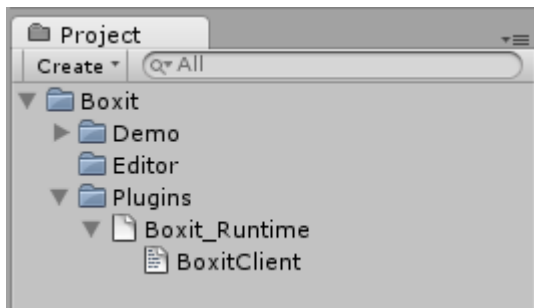
Figure 4-2 Search Boxit

If you have not purchased Boxit, you will see a **Buy** button. If you have already purchased and there is an update available, you will see an **Update** button, otherwise you will just see an **Import** button.

Figure 4-3 Buy

You should now see the Boxit files in your **Project** window.

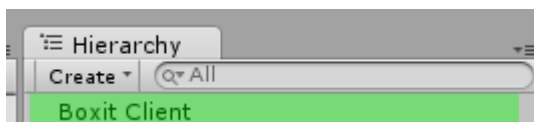
Figure 4-4 Boxit Files



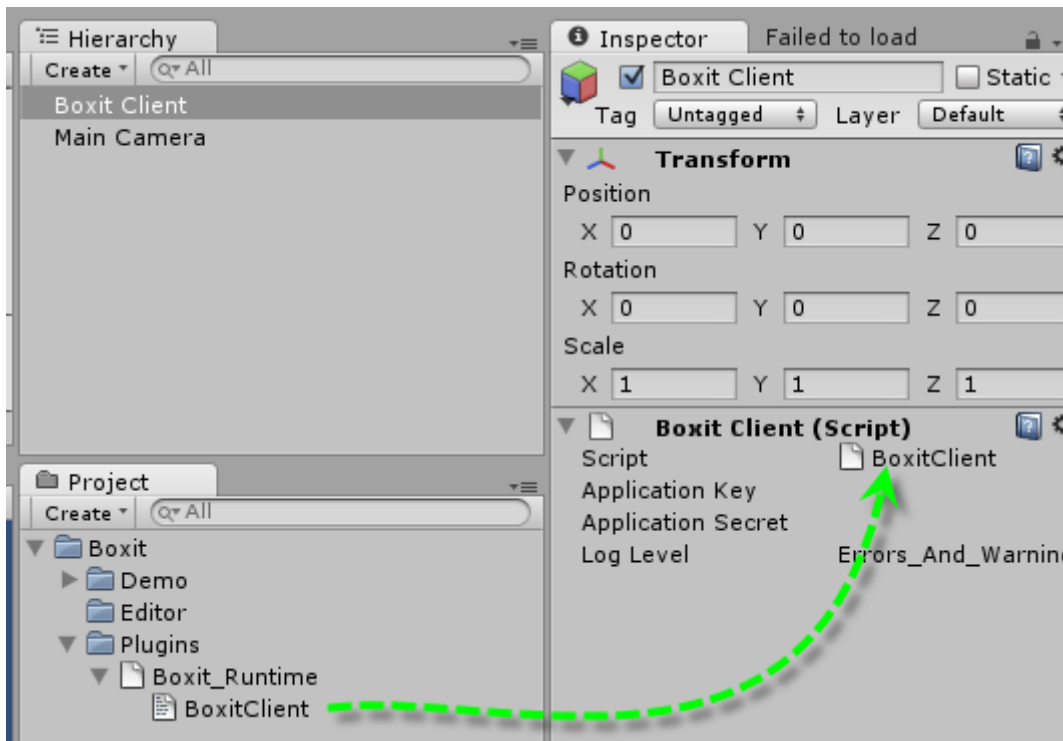
4.3 Create a Boxit Client

To create a Boxit client, create a new GameObject in your **Hierarchy** window.

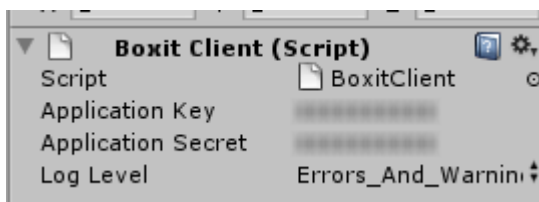
Figure 4-5 Create Client



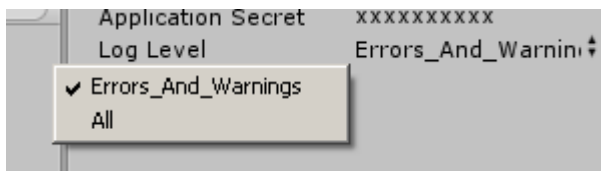
Drag the BoxitClient file from the Boxit_Runtime dll in the Plugins directory to the Boxit Client gameobject's inspector Window.

Figure 4-6 Add Client Component

Fill in the **Application Key** and **Application Secret** with the App Key and App Secret that your app uses from Dropbox.

Figure 4-7 Insert Keys

Set your Log Level to whatever level you want.

Figure 4-8 Set Log Level

- **Errors_And_Warnings:** This will display output to Unity's log whenever there is an error or a warning from Boxit.
- **All:** This will display errors, warnings, and debugging information to Unity's log. This can be useful if you want to see exactly what information is being passed to and from Dropbox.

5.

Asynchronous Functions

Boxit uses asynchronous functions to communicate with Dropbox. What this means is that requests are sent off to the Dropbox server and Unity will continue to process.

It is important to realize that Boxit will not have information immediately after calling a request to Dropbox. You should not code your calls to handle data after you communicate with Dropbox. Instead, you will handle results, errors, warnings, and progress in separate delegate functions. These functions are called at the appropriate times: upon completion of the request, upon failure of the request, when a warning occurs, or while the request progresses, respectively.

5.1 Example

Here is an example of Boxit requesting a file be downloaded from Dropbox. Note that we don't handle the file being downloaded immediately after calling `DownloadFile`, but instead handle the processing of the file in the success delegate.

```
Boxit.BoxitClient client;

void Awake()
{
    client.DownloadFile(Boxit.ROOT.Dropbox,
```

```

        "Test Folder/Test File.txt",
        "Local Test Folder/Subfolder A/Test
File.txt",

        Success,
        Failure,
        Warning,
        Progress);
    }

    private void Success(long requestID, string fullFilePath)
    {
        Debug.Log("Success! File downloaded to: " + fullFilePath);
    }

    private void Failure(long requestID, string error)
    {
        Debug.Log("Failed to download file. Reason: " + error);
    }

    private void Warning(long requestID, string warning)
    {
        Debug.Log("Warning occurred when downloading file: " + warning);
    }

    private void Progress(long requestID, float progress)
    {
        Debug.Log("Download Progress: " + (progress * 100) + "%");
    }

```

You can see that if we wanted to use the downloaded file, we would not put the code for this after the `DownloadFile` call. Instead we'd put this code in the `Success` delegate function.

In this example, the `DownloadFile` function can take up to four delegates, but only the success delegate is required. If you don't supply the failure delegate, Boxit will output to Unity's error log. If you don't supply the warning delegate, Boxit will output to Unity's warning log. If you don't supply a progress delegate, then Boxit will not send any progress notifications.



Using a progress delegate can be useful if you want to show a progress bar or some other way to inform your users that a download or upload is occurring.

5.2 Benefits

Using asynchronous calls lets you continue to update your application, animate graphics, etc. You won't be locked up while the file downloads, uploads, or requests are processed. Sometimes these requests can take a long time, so this lets you run your code in parallel.

It can be tricky to code your application in an asynchronous environment, so caution must be exercised so that you don't try to process information that has not been received.



See [Advanced Syncing](#) for more information on how to sync your data in an asynchronous environment.

5.3 Delegates

Each Boxit function can have one or more delegates that are called when the procedure completes, fails, or is in progress.

SuccessDelegate

Description: This function is called when the Boxit procedure has completed successfully. The result type depends on the function being called. For instance, [GetMetaData](#) will return a [MetaData](#) structure, whereas [GetAccountInfo](#) will return an [AccountInfo](#) structure.

```
void SuccessDelegate (long requestID, <T> results)
```

Parameters:

- long **requestID**: The request ID assigned to the procedure. The requestID is just the current date/time in string format. This value helps in debugging where multiple requests might be running in parallel.
- <T> **results**: The results type will vary among the functions, returning data relevant to whatever is being called.

Returns:

Nothing is returned by a SuccessDelegate.

Usage:

Example 1:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetAccountInfo(Success);
}

private void Success(long requestID, Boxit.AccountInfo accountInfo)
{
    // process accountInfo here
}
```

Example 2:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetMetaData(Boxit.ROOT.sandbox,
                      "Folder A/Test.txt",
                      Success);
}

private void Success(long requestID, Boxit.MetaData metaData)
{
    // process metaData here
}
```

FailureDelegate

Description: This function is called if the procedure fails. If you don't supply a FailureDelegate, then Boxit will send the error to Unity's log as an error.

```
void FailureDelegate (long requestID, string error)
```

Parameters:

- long **requestID**: The request ID assigned to the procedure. The requestID is just the current date/time in string format. This value helps in debugging where multiple requests might be running in parallel.
- string **error**: The error description that caused the procedure to fail.

Returns:

Nothing is returned by a FailureDelegate.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetAccountInfo(Success, Failure);
}

void Success(long requestID, Boxit.AccountInfo accountInfo)
{
    // process accountInfo here
}
```



```
void Failure(long requestID, string error)
{
    // process failure here
}
```

WarningDelegate

Description: The warning delegate is called if there are any non-critical issues that inhibit the Boxit procedure from completing. Not many Boxit functions have a warning delegate. If no warning delegate is supplied, Boxit will send the message to Unity's log as a warning.

```
void WarningDelegate (long requestID, string warning)
```

Parameters:

- long **requestID**: The request ID assigned to the procedure. The requestID is just the current date/time in string format. This value helps in debugging where multiple requests might be running in parallel.
- string **warning**: The warning message that describes the reason the Boxit procedure could not complete.

Returns:

Nothing is returned by a WarningDelegate.

Usage:

```
Boxit.BoxitClient client;

void Awake()
```

```

{
    client.DownloadFile(Boxit.ROOT.sandbox,
                        "test.txt",
                        "Local Folder A/test.txt",
                        Success,
                        null,
                        Warning);
}

void Success(long requestID, string localFilePath)
{
    // handle the downloaded file here
}

void Warning(long requestID, string warning)
{
    // handle the warning here
}

```

ProgressDelegate

Description: The progress delegate is called as a procedure is being performed. This can be useful for feedback during long running operations such as [DownloadFile](#) or [UploadFile](#). If no progress delegate is supplied, then nothing is fed back to your application.

```
void ProgressDelegate (long requestID, float progress)
```

Parameters:

- long **requestID**: The request ID assigned to the procedure. The requestID is just the current date/time in string format. This value helps in debugging where multiple requests might be running

in parallel.

- float **progress**: The progress of the procedure. This value will be between zero (0) and one (1.0f).

Returns:

Nothing is returned by a ProgressDelegate.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.DownloadFile(Boxit.ROOT.sandbox,
                        "test.txt",
                        "Local Folder A/test.txt",
                        Success,
                        null,
                        null,
                        Progress);
}

void Success(long requestID, string localFilePath)
{
    // handle the downloaded file here
}

void Progress(long requestID, float progress)
{
    // handle the progress here, perhaps with a progress bar or some
    other visual aid.
}
```

6.

Linking and Unlinking

6.1 Linking

Before your application can access your Dropbox folder, you have to link the app. This only needs to be done one time. Once an app is linked, it will connect to Dropbox automatically with the stored Access keys.



Anytime you create a new Boxit client in your scenes, Boxit will remember your linked Access keys for the App Key / Secret pair, so you don't need to link your app in each scene, just once for the whole application.

Linking Function

Description: Links your application to a user's Dropbox account.

```
long Link (SuccessDelegate <OAuthToken> success, [FailureDelegate  
failure = null])
```

Parameters:

- **SuccessDelegate success**: The function to call when the link completes. This delegate will have an **oAuthToken** structure with the Access token from Dropbox. You don't need to store this token anywhere since Boxit it does this for you.
- **FailureDelegate failure** (optional): The function to call if the link fails. If this delegate is not provided, then Boxit will just output any errors to Unity's log. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

private void SomeFunction()
{
    client.Link(Success);
}

private void Success(long requestID, Boxit.oAuthToken token)
{
    Debug.Log("Link Successful!");
}
```

Linking Process

Linking is actually a complicated process that involves many steps, all which happen behind the scenes to make your life easier:

1. Dropbox is sent your application key and secret with a request for an authorization token
2. Boxit uses the authorization token, sending the user to a web browser to log in and authenticate
3. After authenticating, Boxit requests the Access token from Dropbox
4. All future Boxit requests are signed with the Access token so no further login is required

As of version 1, Dropbox requires all authentication be done through either a web browser or their own application. Boxit handles this by opening a web browser and allowing the user to log in. Unity will go into a pause state, awaiting the finalization of the login. When the user comes back to your application, Boxit assumes that the user has successfully logged in, otherwise you will get an error when Boxit requests the Access token from Dropbox.

6.2 Unlinking

In the rare case where your user requests to be unlinked from Dropbox, you can call the Unlink function in the Boxit client. This will remove the Access keys from the player prefs associated with the app key and secret. Any future Boxit requests will be denied until the user links to Dropbox again.

Unlink Function

Description: Deletes the Access token from the user's player pref's, effectively disconnecting the application from Dropbox.

```
void Unlink ( )
```

Parameters:

None

Returns:

No Return

Usage:

```
Boxit.BoxitClient client;  
  
void Awake()  
{  
    client.Unlink();  
}
```

7.

Boxit Functions

7.1 GetAccountInfo

Description: Retrieves the account information for the logged in user.

```
long GetAccountInfo (SuccessDelegate <AccountInfo> success, [  
FailureDelegate failure = null] , [string locale = null])
```

Parameters:

- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will have an **AccountInfo** structure with the data from Dropbox.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetAccountInfo(Success);
}

private void Success(long requestID, Boxit.AccountInfo accountInfo)
{
    // process accountInfo here
}
```

7.2 GetMetaData

Description: Retrieves file and folder metadata

```
long GetMetaData (ROOT root, string remotePath, SuccessDelegate <
MetaData> success, [FailureDelegate failure = null], [long fileLimit
= 10000], [string hash = null], [bool list = true], [bool
includeDeleted = false], [string rev = null], [string locale =
null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.

- string **remotePath**: The path to the file or folder. This path is relative to the root specified.
- [SuccessDelegate](#) **success**: Delegate that is called when the procedure completes. This delegate will have a [MetaData](#) structure with the data from Dropbox.
- [FailureDelegate](#) **failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- long **fileLimit** (optional): When listing a folder, the service will not report listings containing more than the specified amount of files and will instead respond with a 406 (Not Acceptable) status response. Default = 10,000. Max = 25,000.
- string **hash** (optional): Each call to [GetMetaData](#) on a folder will return a hash field, generated by hashing all of the metadata contained in that response. On later calls to [GetMetaData](#), you should provide that value via this parameter so that if nothing has changed, the result will be a null [MetaData](#) structure instead of the full, potentially very large, folder listing. This parameter is ignored if the specified path is associated with a file or if list = false. A folder shared between two users will have the same hash for each user. Default = null.
- bool **list** (optional): If true, the folder's [MetaData](#) will include a contents field with a list of metadata entries for the contents of the folder. If false, the contents field will be null. This parameter is only relevant to folders, not files. Default = true.
- bool **includeDeleted** (optional): If this parameter is set to true, then files and folders that have been deleted will also be included in the [GetMetaData](#) call. Default = false.
- string **rev** (optional): If you include a particular revision number, then only the [MetaData](#) for that revision will be returned. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
```

```

{
    client.GetMetaData(Boxit.ROOT.sandbox,
                      "Folder A/Test.txt",
                      Success);
}

private void Success(long requestID, Boxit.MetaData metaData)
{
    // process metaData here. If metaData == null, then no change
    occurred given the hash code
}

```

7.3 DownloadFile

Description: Downloads a file to your local application's persistent data path (working directory).

```

long DownloadFile (ROOT root, string remotePath, string localPath,
                  SuccessDelegate <string> success, [FailureDelegate failure = null],
                  [WarningDelegate warning = null], [ProgressDelegate progress =
null], [bool overwriteLocal = true], [string rev = null])

```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- string **remotePath:** The path to the file on the Dropbox server. This path is relative to the root specified.
- string **localPath:** The path to the file on the local device. This path is relative to the application's

persistent data path (working directory).

- **SuccessDelegate success**: Delegate that is called when the procedure completes. This delegate will have a string result with the full path to the local file.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- **WarningDelegate warning** (optional): This delegate is called if the local file exists and `overwriteLocal` is set to false. If no warning delegate is specified, then Boxit will output the message to Unity's warning log. Default = null.
- **ProgressDelegate progress** (optional): This delegate is called as the procedure runs, giving an update to the progress incrementally between zero (0) and one (1.0f). If no progress delegate is specified, then nothing will be sent back to the application. Default = null.
- **bool overwriteLocal** (optional): If true, the download will wipe out the local file and replace it with the downloaded file. If this is set to false and the file exists, a warning will occur. Default = true.
- **string rev** (optional): The revision of the file to retrieve. If no revision is specified, then the most recent revision will be downloaded. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.DownloadFile(Boxit.ROOT.sandbox,
                        "test.txt",
                        "Local Folder A/test.txt",
                        Success,
                        null,
                        Warning,
                        Progress);
}
```

```

}

void Success(long requestID, string localFilePath)
{
    // handle the downloaded file here
}

void Warning(long requestID, string warning)
{
    // handle the warning here
}

void Progress(long requestID, float progress)
{
    // handle the progress here. Perhaps with a progress bar or some
    other visual aid.
}

```

7.4 UploadFile

Description: Uploads a file to the Dropbox server.

```

long UploadFile (ROOT root, string localPath, string remotePath,
    SuccessDelegate <MetaData> success, [FailureDelegate failure =
    null], [ProgressDelegate progress = null], [bool overwriteRemote =
    true], [string parentRev = null], [string locale = null])

```

Parameters:

- **ROOT root**: The root relative to which path is specified. Valid values are `Boxit.ROOT.sandbox` and `Boxit.ROOT.dropbox`.
- string **localPath**: The path to the file on the local device. This path is relative to the application's persistent data path (working directory).
- string **remotePath**: The path to the file on the Dropbox server. This path is relative to the root specified.
- **SuccessDelegate success**: Delegate that is called when the procedure completes. This delegate will have a **MetaData** result with the properties of the uploaded file.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- **ProgressDelegate progress** (optional): This delegate is called as the procedure runs, giving an update to the progress incrementally between zero (0) and one (1.0f). If no progress delegate is specified, then nothing will be sent back to the application. Default = null.
- bool **overwriteRemote** (optional): If true, the existing file will be overwritten by the new one. If false, the new file will be automatically renamed (for example, `test.txt` might be automatically renamed to `test (1).txt`). The new name can be obtained from the returned metadata. Default = true.
- string **parentRev** (optional): The revision of the file you're editing. If `parentRev` matches the latest version of the file on the user's Dropbox, that file will be replaced. Otherwise, the new file will be automatically renamed (for example, `test.txt` might be automatically renamed to `test (conflicted copy).txt`). If you specify a revision that doesn't exist, the file will not save (error 400). Get the most recent rev by performing a call to **GetMetaData**. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
```

```

{
    client.UploadFile(Boxit.ROOT.sandbox,
                      "Local Folder/A/image.png",
                      "image.png",
                      Success);
}

public Success(long requestID, Boxit.Metadata metaData)
{
    // process after the file uploads
}

```

7.5 GetRevisions

Description: Obtains [Metadata](#) for the previous revisions of a file.



Only revisions up to thirty days old are available (or more if the Dropbox user has [Pack-Rat](#)). You can use the revision number in conjunction with the [Restore](#) call to revert the file to its previous state.

```

long GetRevisions (ROOT root, string remotePath, SuccessDelegate
<List<Metadata>> success, [FailureDelegate failure = null], [long
revLimit = 10], [string locale = null])

```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and

Boxit.ROOT.dropbox.

- string **remotePath**: The path to the file on the Dropbox server. This path is relative to the root specified.
- [SuccessDelegate](#) **success**: Delegate that is called when the procedure completes. This delegate will have a list of [MetaData](#) containing all the revisions of a file.
- [FailureDelegate](#) **failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- long **revLimit** (optional): When listing a file, the service will not report listings containing more than the amount specified and will instead respond with a 406 (Not Acceptable) status response. Default = 10. Max = 1,000.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetRevisions(Boxit.ROOT.sandbox,
                        "image.png",
                        Success);
}

public Success(long requestID, List<Boxit.MetaData> revisionList)
{
    // process the list of revisions
}
```


7.6 Restore

Description: Restores a file path to a previous revision.



Unlike downloading a file at a given revision and then re-uploading it, this call happens entirely on the Dropbox server. It also saves a lot of bandwidth.

```
long Restore (ROOT root, string remotePath, string rev,
             SuccessDelegate <MetaData> success, [FailureDelegate failure =
             null], [string locale = null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are `Boxit.ROOT.sandbox` and `Boxit.ROOT.dropbox`.
- string **remotePath:** The path to the file on the Dropbox server. This path is relative to the root specified.
- string **rev:** The revision of the file to restore to. You can get a list of file revisions by calling [GetRevisions](#).
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will have a **MetaData** result for the restored file.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```

Boxit.BoxitClient client;

void Awake()
{
    client.Restore(Boxit.ROOT.sandbox,
                  "image.png",
                  "63098a9de2",
                  Success);
}

public Success(long requestID, Boxit.Metadata metaData)
{
    // process the restore file
}

```

7.7 Search

Description: Returns [Metadata](#) for all files and folders whose filename contains the given search string as a substring.



Searches are limited to the folder path and its sub-folder hierarchy provided in the call.

```

long Search (ROOT root, string remotePath, string query,
            SuccessDelegate <List<Metadata>> success, [FailureDelegate failure =
            null], [long fileLimit = 1000], [bool includeDeleted = false],

```

```
[string locale = null])
```

Parameters:

- **ROOT root**: The root relative to which path is specified. Valid values are `Boxit.ROOT.sandbox` and `Boxit.ROOT.dropbox`.
- string **remotePath**: The path to the folder to begin searching at on the Dropbox server. This path is relative to the root specified.
- string **query**: The string to search for in the files and folders.
- **SuccessDelegate success**: Delegate that is called when the procedure completes. This delegate will have a list of **MetaData** for the files that qualified for the search.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- long **fileLimit** (optional): No more than fileLimit search results will be returned. Default = 1,000. Max = 1,000.
- bool **includeDeleted** (optional): If this parameter is set to true, then files and folders that have been deleted will also be included in the search. Default = false.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.Search(Boxit.ROOT.dropbox,
                  "Remote Folder A/Subfolder/",
```

```

        "FindMe",
        Success);
}

public Success(long requestID, List<Boxit.Metadata> metaData)
{
    // process the search results
}

```

7.8 GetShareLink

Description: Creates and returns a [ShareLink](#) to files or folders users can use to view a preview of the file in a web browser.



The difference from [GetMedia](#) is that this does not bypass the Dropbox webservice, used to provide a preview of the file. You cannot stream files with this link.

```

long GetShareLink (ROOT root, string remotePath, SuccessDelegate <
ShareLink> success, [FailureDelegate failure = null], [bool shortUrl
= true], [string locale = null])

```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- **string remotePath:** The path to the file or folder. This path is relative to the root specified.
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will

have a [ShareLink](#) result.

- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- bool **shortUrl** (optional): When true, the url returned will be shortened using the Dropbox url shortener. If false, the url will link directly to the file's preview page. Default = true.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetShareLink(Boxit.ROOT.dropbox,
                        "test.txt",
                        Success);
}

public Success(long requestID, Boxit.ShareLink shareLink)
{
    // process the share link here
}
```

7.9 GetMedia

Description: Returns a [ShareLink](#) directly to a file.



Similar to [GetShareLink](#). The difference is that this bypasses the Dropbox webservice, used to provide a preview of the file, so that you can effectively stream the contents of your media.

```
long GetMedia (ROOT root, string remotePath, SuccessDelegate <
ShareLink> success, [FailureDelegate failure = null], [string locale
= null])
```

Parameters:

- **ROOT root**: The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- string **remotePath**: The path to the file or folder. This path is relative to the root specified.
- **SuccessDelegate success**: Delegate that is called when the procedure completes. This delegate will have a [ShareLink](#) result.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetMedia(Boxit.ROOT.dropbox,
                    "test.txt",
                    Success);
}

public Success(long requestID, Boxit.ShareLink shareLink)
{
    // process the share link here
}
```

7.10 GetCopyRef

Description: Creates and returns a [CopyRef](#) to a file. This reference string can be used to copy that file to another user's Dropbox by passing it in as the fromCopyRef parameter on [CopyFileFromCopyRef](#).

```
long GetCopyRef (ROOT root, string remotePath, SuccessDelegate <
CopyRef> success, [FailureDelegate failure = null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- **string remotePath:** The path to the file. This path is relative to the root specified.
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will

have a [CopyRef](#) result.

- [FailureDelegate failure](#) (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetCopyRef(Boxit.ROOT.dropbox,
                     "test.txt",
                     Success);
}

public Success(long requestID, Boxit.CopyRef copyRef)
{
    // process the copy ref here
}
```


7.11 DownloadThumbnail

Description: Gets a thumbnail for an image and stores it in a Texture2D that Unity can use.

```
long DownloadThumbnail (ROOT root, string remotePath,
    SuccessDelegate <Texture2D> success, [FailureDelegate failure =
    null], [ProgressDelegate progress = null], [THUMBNAIL_FORMAT format
    = THUMBNAIL_FORMAT.jpeg], [THUMBNAIL_SIZE size =
    THUMBNAIL_SIZE.small])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- string **remotePath:** The path to the image file on the Dropbox server. This path is relative to the root specified.
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will have a Texture2D result that can be used directly in Unity.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- **ProgressDelegate progress** (optional): This delegate is called as the procedure runs, giving an update to the progress incrementally between zero (0) and one (1.0f). If no progress delegate is specified, then nothing will be sent back to the application. Default = null.
- **THUMBNAIL_FORMAT format** (optional): The image format for the thumbnail. Default = THUMBNAIL_FORMAT.jpeg.
- **THUMBNAIL_SIZE size** (optional): The size of the thumbnail. Default = THUMBNAIL_SIZE.small.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.DownloadThumbnail(Boxit.ROOT.sandbox,
                            "logo.png",
                            Success,
                            null,
                            null,
                            Boxit.THUMBNAIL_FORMAT.png
                            Boxit.THUMBNAIL_SIZE.xl);
}

void Success(long requestID, Texture2D thumbnail)
{
    // handle the thumbnail here
}
```

7.12 CopyFileFromPath

Description: Copies a file or folder from a path to another path on the Dropbox server.



Using this function will save you from having to download, then re-upload to another path. This is faster and saves a lot of bandwidth.


```

Success);
}

void Success(long requestID, Boxit.Metadata metaData)
{
    // handle the metaData here
}

```

7.13 CopyFileFromCopyRef

Description: Copies a file from a [CopyRef](#) to another path on the Dropbox server.



Using this function can let you save a file into another user's Dropbox.

```

long CopyFileFromCopyRef (ROOT root, string fromCopyRef, string
toRemotePath, SuccessDelegate <Metadata> success, [FailureDelegate
failure = null], [string locale = null])

```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- string **fromCopyRef:** The [CopyRef](#) of the file to be copied. This can be obtained by calling [GetCopyRef](#).
- string **toRemotePath:** The path to the file where the file should be copied on the Dropbox server. This path is relative to the root specified.
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will have a [Metadata](#) result of the file's final location.

- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.CopyFileFromCopyRef(Boxit.ROOT.sandbox,
                               "Ak250Ht2eGhneHJ3Nnp0Ng",
                               "Remote Path/Folder A/logo.png",
                               Success);
}

void Success(long requestID, Boxit.MetaData metaData)
{
    // handle the metaData here
}
```

7.14 CreateFolder

Description: Creates a folder on the Dropbox server.



If you nest folders, they will all be created at once, so you don't need to make repeated calls.

```
long CreateFolder (ROOT root, string remotePath, SuccessDelegate <
MetaData> success, [FailureDelegate failure = null], [string locale
= null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- string **remotePath:** The path of the folder to create on the Dropbox server. This path is relative to the root specified.
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will have a **MetaData** result of the newly created folder.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.CreateFolder(Boxit.ROOT.sandbox,
                        "Folder A/Folder B/Folder C",
                        Success);
}

void Success(long requestID, Boxit.Metadata metaData)
{
    // handle the metaData here
}
```

7.15 Delete

Description: Deletes a file or folder on the Dropbox server.

```
long Delete (ROOT root, string remotePath, SuccessDelegate <Metadata
> success, [FailureDelegate failure = null], [string locale = null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- **string remotePath:** The path of the file or folder to delete on the Dropbox server. This path is relative to the root specified.
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will

have a [MetaData](#) result of the deleted file or folder.

- [FailureDelegate failure](#) (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.Delete(Boxit.ROOT.sandbox,
                  "Folder A/Folder B",
                  Success);
}

void Success(long requestID, Boxit.MetaData metaData)
{
    // handle the metaData here
}
```


7.16 Move

Description: Moves a file or folder to a new location on the Dropbox server.



This function saves you from having to download a file, then delete the remote file, then re-upload to a new location. It is much faster and saves a lot of bandwidth.

```
long Move (ROOT root, string fromRemotePath, string toRemotePath,
SuccessDelegate <MetaData> success, [FailureDelegate failure =
null], [string locale = null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- string **fromRemotePath:** The path of the file or folder to move on the Dropbox server. This path is relative to the root specified.
- string **toRemotePath:** The path of the file or folder to move to on the Dropbox server. This path is relative to the root specified.
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will have a **MetaData** result of the file or folder's final location.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.Move(Boxit.ROOT.sandbox,
                "Folder A/Folder B/test.txt",
                "Folder A/Folder B/test2.txt",
                Success);
}

void Success(long requestID, Boxit.MetaData metaData)
{
    // handle the metaData here
}
```

7.17 GetDelta

Description: A way of letting you keep up with changes to files and folders in a user's Dropbox. You can periodically call [GetDelta](#) to get a list of [DeltaEntry](#), which are instructions on how to update your local state to match the server's state. If you pass a cursor, then only the changes that have occurred since that cursor will be returned, otherwise all changes will be returned.

```
long GetDelta (string cursor, SuccessDelegate <Delta> success, [
FailureDelegate failure = null], [string locale = null])
```

Parameters:

- string **cursor**: A string that is used to keep track of your current state. On the next call pass in this value to return delta entries that have been recorded since the cursor was returned.
- [SuccessDelegate](#) **success**: Delegate that is called when the procedure completes. This delegate will have a [Delta](#) result that lists all the changes that have occurred since the cursor.
- [FailureDelegate](#) **failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- string **locale** (optional): Use to specify language settings for user error messages and other language specific text. Click [here](#) for a description of the locale value. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.GetDelta("",
                    Success);
}

void Success(long requestID, Boxit.Delta delta)
{
    // handle the delta here
}
```

7.18 DownloadFileIfRemoteNewer

Description: Downloads a file to your local application's persistent data path (working directory) if the remote version of the file is more recent than the local version.

```
long DownloadFileIfRemoteNewer (ROOT root, string remotePath, string
localPath, SuccessDelegate <string> success, [FailureDelegate
failure = null], [The internal link is invalid. progress = null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- string **remotePath:** The path to the file on the Dropbox server. This path is relative to the root specified.
- string **localPath:** The path to the file on the local device. This path is relative to the application's persistent data path (working directory).
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will have a string result with the full path to the local file.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- **ProgressDelegate progress** (optional): This delegate is called as the procedure runs, giving an update to the progress incrementally between zero (0) and one (1.0f). If no progress delegate is specified, then nothing will be sent back to the application. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.DownloadFileIfRemoteNewer(Boxit.ROOT.sandbox,
                                     "test.txt",
                                     "Local Folder A/test.txt",
                                     Success);
}

void Success(long requestID, string localFilePath)
{
    // handle the downloaded file here
}
```

7.19 UploadFileIfLocalNewer

Description: Uploads a file from your applications persistent data path (working directory) to the Dropbox server only if the local version of the file is more recent than the remote version.

```
long UploadFileIfLocalNewer (ROOT root, string localPath, string
remotePath, SuccessDelegate <MetaData> success, [FailureDelegate
failure = null], [ProgressDelegate progress = null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.

- string **localPath**: The path to the file on the local device. This path is relative to the application's persistent data path (working directory).
- string **remotePath**: The path to the file on the Dropbox server. This path is relative to the root specified.
- **SuccessDelegate success**: Delegate that is called when the procedure completes. This delegate will have a **MetaData** result of the uploaded file.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- **ProgressDelegate progress** (optional): This delegate is called as the procedure runs, giving an update to the progress incrementally between zero (0) and one (1.0f). If no progress delegate is specified, then nothing will be sent back to the application. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.UploadFileIfLocalNewer(Boxit.ROOT.sandbox,
                                  "Local Folder A/test.txt",
                                  "test.txt",
                                  Success);
}

void Success(long requestID, Boxit.MetaData metaData)
{
    // handle the metaData here
}
```

7.20 SyncFile

Description: Makes sure that the most recent version of a file is in the local and remote locations. If the local file is newer, then Boxit will upload the file. If the remote file is newer, then Boxit will download the file.

```
long SyncFile (ROOT root, string localPath, string remotePath,
  SuccessDelegate <DateTime> success, [FailureDelegate failure =
  null], [ProgressDelegate progress = null])
```

Parameters:

- **ROOT root:** The root relative to which path is specified. Valid values are Boxit.ROOT.sandbox and Boxit.ROOT.dropbox.
- string **localPath:** The path to the file on the local device. This path is relative to the application's persistent data path (working directory).
- string **remotePath:** The path to the file on the Dropbox server. This path is relative to the root specified.
- **SuccessDelegate success:** Delegate that is called when the procedure completes. This delegate will have a DateTime result that stores the current time stamp of both the local and remote file.
- **FailureDelegate failure** (optional): This delegate is called if the procedure fails. If no delegate is supplied, Boxit will output to Unity's error log. Default = null.
- **ProgressDelegate progress** (optional): This delegate is called as the procedure runs, giving an update to the progress incrementally between zero (0) and one (1.0f). If no progress delegate is specified, then nothing will be sent back to the application. Default = null.

Returns:

long value representing the Boxit request ID. This is useful for debugging purposes in the case where you have many requests running in parallel.

Usage:

```
Boxit.BoxitClient client;

void Awake()
{
    client.SyncFile(Boxit.ROOT.sandbox,
                    "Local Folder A/test.txt",
                    "test.txt",
                    Success);
}

void Success(long requestID, DateTime dateTime)
{
    // handle the dateTime here
}
```


8.

Advanced Syncing

If your application will be run on multiple devices that share the same Dropbox files, then you will need to employ some form of advanced synchronization.

The basic steps for this are:

1. Check to see if the remote file(s) is more recent than the local file(s). If it is, download the remote file. This will ensure that any changes made by other devices will be pulled in locally before making updates.
2. Update the file(s) that you downloaded.
3. Upload the file(s) back to Dropbox for the next device to use.

Example Code:

```
Boxit.BoxitClient client;

void UpdateAndSync()
{
    // 1) Download the remote file if it is more recent
    client.DownloadFileIfRemoteNewer(Boxit.ROOT.sandbox,
                                     "RemoteFolder/file.txt",
                                     "LocalFolder/file.txt",
                                     DownloadSuccess,
                                     Failure);
}
```

```
private void DownloadSuccess(long requestID, string localFilePath)
{
    // 2) Update the file's contents, or do whatever processing is
    necessary here

    // 3) Upload the file back to the Dropbox server
    client.UploadFile(Boxit.ROOT.sandbox,
        "LocalFolder/file.txt",
        "RemoteFolder/file.txt",
        UploadSuccess,
        Failure);
}

private void UploadSuccess(long requestID, Boxit.MetaData metaData)
{
    // process the completion of the update and sync
}

private void Failure(long requestID, string error)
{
    // the update and sync did not succeed
}
```

9.

Boxit Data Structures

9.1 Global Variables

ROOT

Description: The root from which to start remotely. Valid values:

- **sandbox:** Starts the path at the application's folder. This is the only valid value if your application's access level is set to **App Folder**. If your access is set to **Full Dropbox**, then you can use sandbox as well.
- **dropbox:** Starts the path at the user's Dropbox root. This can only be used if the application's access level is set to **Full Dropbox**.

THUMBNAIL_FORMAT

Description: The image format for the downloaded thumbnail. Valid values:

- **jpeg:** Good for photos.
- **png:** Good for screenshots, illustrations, logos, etc.

THUMBNAIL_SIZE

Description: The size of the downloaded thumbnail. Valid values:

- **small:** 32 x 32
- **medium:** 64 x 64
- **large:** 127 x 128
- **s:** 64 x 64
- **m:** 128 x 128
- **l:** 640 x 480
- **xl:** 1024 x 768

9.2 oAuthToken

Description: Stores the token and secret pair for authorization.

Members:

string **Token:** The token of the authorization pair.

string **Secret:** The secret of the authorization pair.

Methods:

string **ToString():** returns a string description of the [oAuthToken](#).

Properties:

bool **IsEmpty:** returns whether the token has any data in it.

9.3 AccountInfo

Description: Stores information about a user's account.

Members:

string **Referral_Link**: The user's referral link.

string **Display_Name**: The user's display name.

string **UID**: The user's unique Dropbox ID.

string **Country**: The user's two-letter country code, if available.

string **Email**: The user's email address.

[QuotaInfo](#) **Quota_Info**: The user's quota information.

Methods:

string **ToString()**: returns a string description of the [AccountInfo](#).

9.4 QuotaInfo

Description: The user's quota information for their account.

Members:

long **Shared**: The user's used quota in shared folders (bytes).

long **Quota**: The user's total quota allocation (bytes).

long **Normal**: The user's used quota outside of shared folders (bytes).

Methods:

string **ToString()**: returns a string description of the [QuotaInfo](#).

9.5 MetaData

Description: Stores file and folder information.

Members:

string **Size**: A human-readable description of the file size (translated by locale).

string **Rev**: A unique identifier for the current revision of a file. This field is the same rev as elsewhere in the API and can be used to detect changes and avoid conflicts.

bool **Thumb_Exists**: True if the file is an image that can be converted to a thumbnail via the DownloadThumbnail call.

long **Bytes**: The file size in bytes.

string **Modified**: The last time the file was modified on Dropbox, in the standard date format (not included for the root folder).

string **Client_Mtime**: For files, this is the modification time set by the desktop client when the file was added to Dropbox, in the standard date format. Since this time is not verified (the Dropbox server stores whatever the desktop client sends up), this should only be used for display purposes (such as sorting) and not, for example, to determine if a file has changed or not.

string **Path**: Returns the canonical path to the file or directory.

bool **Is_Dir**: Whether the given entry is a folder or not.

string **Icon**: The name of the icon used to illustrate the file type in Dropbox's icon library.

string **Root**: The root or top-level folder depending on your access level. All paths returned are relative

to this root level. Permitted values are either `dropbox` or `app_folder`.

string **Mime_Type**: The Mime type of the file or folder.

string **Hash**: A folder's hash is useful for indicating changes to the folder's contents in later calls to [GetMetaData](#). This is roughly the folder equivalent to a file's rev.

bool **Is_Deleted**: Whether the given entry is deleted (only included if deleted files are being returned).

List<[MetaData](#)> **Contents**: A list of [MetaData](#) contents for each of the files in a folder.

Methods:

string **ToString()**: returns a string description of the [MetaData](#) information.

Properties:

DateTime **ModifiedDate**: DateTime representation of the Modified member.

DateTime **UTCDateModified**: UTC DateTime representation of the Modified member.

9.6 Delta

Description: A way of letting you keep up with changes to files and folders in a user's Dropbox. You can periodically call [GetDelta](#) to get a list of [DeltaEntry](#), which are instructions on how to update your local state to match the server's state.

Members:

string **Cursor**: A string that encodes the latest information that has been returned. On the next call to [GetDelta](#), pass in this value.

bool **Has_More**: If true, then there are more entries available; you can call [GetDelta](#) again immediately

to retrieve those entries. If 'false', then wait for at least five minutes (preferably longer) before checking again.

bool **Reset**: If true, clear your local state before processing the delta entries. Reset is always true on the initial call to [GetDelta](#) (i.e. when no cursor is passed in). Otherwise, it is true in rare situations, such as after server or account maintenance, or if a user deletes their app folder.

List<[DeltaEntry](#)> **Entries**: A list of [DeltaEntry](#).

Methods:

string **ToString**(): returns a string description of the [Delta](#) information.

9.7 DeltaEntry

Description: Each delta entry is a 2-item list of one of the following forms:

[<path>, <[MetaData](#)>] - Indicates that there is a file/folder at the given path. You should add the entry to your local path. The metadata value is the same as what would be returned by the [GetMetaData](#) call, except folder [MetaData](#) doesn't have hash or contents fields. To correctly process [Delta](#) entries:

- If the new entry includes parent folders that don't yet exist in your local state, create those parent folders in your local state.
- If the new entry is a file, replace whatever your local state has at path with the new entry.
- If the new entry is a folder, check what your local state has at <path>. If it's a file, replace it with the new entry. If it's a folder, apply the new <[MetaData](#)> to the folder, but do not modify the folder's children.

[<path>, null] - Indicates that there is no file/folder at the given path. To update your local state to match, anything at path and all its children should be deleted. Deleting a folder in your Dropbox will sometimes send down a single deleted entry for that folder, and sometimes separate entries for the folder and all child paths. If your local state doesn't have anything at path, ignore this entry.

Note: Dropbox treats file names in a case-insensitive but case-preserving way. To facilitate this, the `<path>` values above are lower-cased versions of the actual path. The `<MetaData>` value has the original case-preserved path.

Members:

string **Path**: Path of the file or folder.

[MetaData](#) **MetaData**: The [MetaData](#) for the file or folder.

Methods:

string **ToString()**: returns a string description of the [DeltaEntry](#) information.

9.8 ShareLink

Description: Stores a Dropbox link to files or folders users can use to view a preview of the file in a web browser. The link can be used publicly and directs to a preview page of the file. For compatibility reasons, it returns the link's expiration date in Dropbox's usual date format. All links are currently set to expire far enough in the future so that expiration is effectively not an issue.

Members:

string **Url**: Link to the file.

string **Expires**: Expiration date of the link.

Methods:

string **ToString()**: returns a string description of the [ShareLink](#) information.

Properties:

DateTime **ExpiresDate**: DateTime representation of the Expires string.

DateTime **UTCDateExpires**: UTC DateTime representation of the Expires string.

9.9 CopyRef

Description: Stores a copyRef to a file. This reference string can be used to copy that file to another user's Dropbox by passing it in as the fromCopyRef parameter on the [CopyFileFromCopyRef](#) function.

Members:

string **Copy_Ref**: Reference to the file for copying.

string **Expires**: Expiration date of the copy ref.

Methods:

string **ToString()**: returns a string description of the [CopyRef](#) information.

Properties:

DateTime **ExpiresDate**: DateTime representation of the Expires string.

DateTime **UTCDateExpires**: UTC DateTime representation of the Expires string.