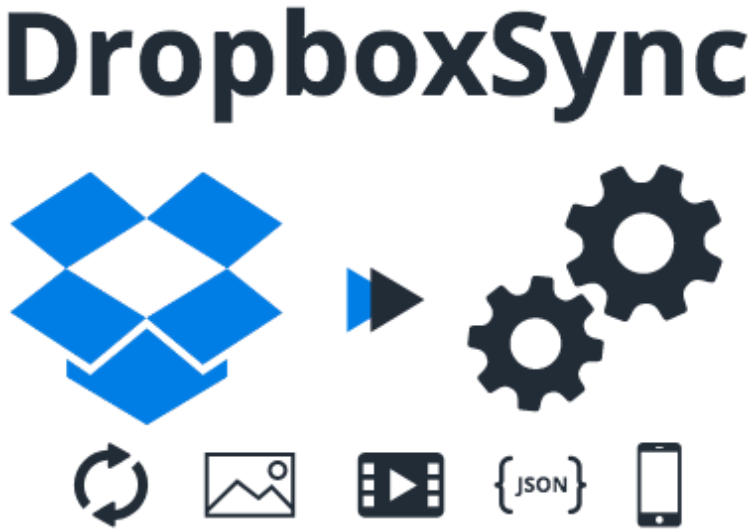
DropboxSync v4 Documentation

Go to Tutorial



Downloading

GetFileAsLocalCachedPath()

```
public async void GetFileAsLocalCachedPath(string dropboxPath, Progress<TransferProgressRe
port> progressCallback, Action<string> successCallback, Action<Exception> errorCallback, b
ool useCachedFirst = false, bool useCachedIfOffline = true, bool receiveUpdates = false, C
ancellationToken? cancellationToken = null)
```

Asynchronously retrieves file from Dropbox and returns path to local filesystem cached copy.

Parameter	Description
dropboxPath	Path to file on Dropbox
progressCallback	Progress callback with download percentage and speed
successCallback	Callback for receiving downloaded file path
errorCallback	Callback that is triggered if any exception happened
useCachedFirst	Serve cached version (if it exists) before event checking Dropbox for newer version?
useCachedIfOffline	Use cached version if no Internet connection?
receiveUpdates	If `true`, then when there are remote updates on Dropbox, callback function `successCallback` will be triggered again with updated version of the file.
cancellationToken	Cancellation token that can be used to cancel download

Example usage

```
_cancellationTokenSource = new CancellationTokenSource();

DropboxSync.Main.GetFileAsLocalCachedPath(inputField.text,
  new Progress<TransferProgressReport>((report) => {
      statusText.text = $"Downloading: {report.progress}% {report.bytesPerSecondFormatte
d}";
  }),
  (localPath) => {
      // success
      print($"Completed");
      statusText.text = $"<color=green>Local path: {localPath}</color>";
  }, (ex) => {
      // exception
      if(ex is OperationCanceledException){
          Debug.Log("Download cancelled");
          statusText.text = $"<color=orange>Download canceled.</color>";
      }else{
          Debug.LogException(ex);
          statusText.text = $"<color=red>Download failed.</color>";
      }
  },
  cancellationToken: _cancellationTokenSource.Token);
```

Example usage (async)

```
_cancellationTokenSource = new CancellationTokenSource();

try {
  var localPath = await DropboxSync.Main.GetFileAsLocalCachedPathAsync(inputField.text,
  new Progress<TransferProgressReport>((report) => {
      print($"Downloading: {report.progress}% {report.bytesPerSecondFormatted}");
  }), _cancellationTokenSource.Token);

  print($"Completed");
}catch(Exception ex){
  if(ex is OperationCanceledException){
    print("Download cancelled");
  }else{
    Debug.LogException(ex);
  }
}
```

GetFileAsBytes()

```
public void GetFileAsBytes(string dropboxPath, Progress<TransferProgressReport> progressCallback, Action<byte[]> successCallback, Action<Exception> errorCallback, bool useCachedFirst = false, bool useCachedIfOffline = true, bool receiveUpdates = false, CancellationToken? cancellationToken = null)
```

Asynchronously retrieves file from Dropbox and returns it as byte array

Parameter	Description
dropboxPath	Path to file on Dropbox
progressCallback	Progress callback with download percentage and speed
successCallback	Callback for receiving downloaded file bytes
errorCallback	Callback that is triggered if any exception happened
useCachedFirst	Serve cached version (if it exists) before event checking Dropbox for newer version?
useCachedIfOffline	Use cached version if no Internet connection?
receiveUpdates	If `true`, then when there are remote updates on Dropbox, callback function `successCallback` will be triggered again with updated version of the file.
cancellationToken	Cancellation token that can be used to cancel download

GetFile<T>()

```
public void GetFile<T>(string dropboxPath, Progress<TransferProgressReport> progressCallback, Action<T> successCallback, Action<Exception> errorCallback, bool useCachedFirst = false, bool useCachedIfOffline = true, bool receiveUpdates = false, CancellationToken? cancellationToken = null)
```

Retrieves file from Dropbox and returns it as T (T can be string, Texture2D or any type that can be deserialized from text using JsonUtility)

Parameter	Description
dropboxPath	Path to file on Dropbox
progressCallback	Progress callback with download percentage and speed
successCallback	Callback for receiving downloaded object T (T can be string, Texture2D or any type that can be deserialized from text using JsonUtility)

errorCallback	Callback that is triggered if any exception happened
useCachedFirst	Serve cached version (if it exists) before event checking Dropbox for newer version?
useCachedIfOffline	Use cached version if no Internet connection?
receiveUpdates	If `true`, then when there are remote updates on Dropbox, callback function `successCallback` will be triggered again with updated version of the file.
cancellationToken	Cancellation token that can be used to cancel download

Example usage

```
DropboxSync.Main.GetFile<Texture2D>(EXAMPLE_IMAGE_PATH,
  new Progress<TransferProgressReport>((progress) => {}),
  (texture) => {
    UpdatePicture(texture);
  }, (ex) => {
    Debug.LogError($"Error getting picture from Dropbox: {ex}");
  }, receiveUpdates: true, useCachedFirst:true);
```

Uploading

UploadFile(string localFilePath)

```
public async void UploadFile(string localFilePath, string dropboxPath, Progress<TransferPr
ogressReport> progressCallback, Action<Metadata> successCallback, Action<Exception> errorC
allback, CancellationToken? cancellationToken)
```

Uploads file from specified filepath in local filesystem to Dropbox

Parameter	**Description**
localFilePath	Path to local file
dropboxPath	Upload path on Dropbox
progressCallback	Progress callback with upload percentage and speed
successCallback	Callback for receiving uploaded file Metadata
errorCallback	Callback that is triggered if any exception happened

cancellationToken	Cancellation token that can be used to cancel the upload

Example usage

```
_cancellationTokenSource  = new CancellationTokenSource();

DropboxSync.Main.UploadFile(LOCAL_FILE_PATH, DROPBOX_UPLOAD_PATH,
  new Progress<TransferProgressReport>((report) => {
    print($"Uploading file {report.progress}% {report.bytesPerSecondFormatted}");
  }), (metadata) => {
    // success
    print($"Upload completed:\n{metadata}");
  }, (ex) => {
    // exception
    if(ex is OperationCanceledException){
      print("Upload cancelled");
    }else{
      Debug.LogException(ex);
    }
  }, _cancellationTokenSource.Token);
```

Example usage (async)

```
_cancellationTokenSource  = new CancellationTokenSource();

try {
  var metadata = await DropboxSync.Main.UploadFileAsync(
    LOCAL_FILE_PATH, DROPBOX_UPLOAD_PATH,
    new Progress<TransferProgressReport>((report) => {
      print($"Uploading file {report.progress}% {report.bytesPerSecondFormatted}");
  }), _cancellationTokenSource.Token);

  print($"Upload completed:\n{metadata}");

}catch(Exception ex){
  if(ex is OperationCanceledException){
    print("Upload cancelled");
  }else{
    Debug.LogException(ex);
  }
}
```

UploadFile(byte[] bytes)

```
public async void UploadFile(byte[] bytes, string dropboxPath, Progress<TransferProgressRe
port> progressCallback, Action<Metadata> successCallback, Action<Exception> errorCallback,
CancellationToken? cancellationToken)
```

Uploads byte array to Dropbox

Parameter	Description
bytes	Bytes to upload
dropboxPath	Upload path on Dropbox
progressCallback	Progress callback with upload percentage and speed
successCallback	Callback for receiving uploaded file Metadata
errorCallback	Callback that is triggered if any exception happened
cancellationToken	Cancellation token that can be used to cancel the upload

Keeping in Sync

NOTE: Synchronization is only one way: Dropbox to Local cache

KeepSynced()

```
public void KeepSynced(string dropboxPath, Action<EntryChange> syncedCallback)
```

Keep Dropbox file or folder synced (one-way: from Dropbox to Local cache)

Parameter	Description
dropboxPath	File or folder path on Dropbox
syncedCallback	Callback that is triggered after change is synced from Dropbox

Example usage

```
void Start() {
  DropboxSync.Main.KeepSynced(DROPBOX_PATH, OnChangeSynced);
}

void OnChangeSynced(DBXSync.EntryChange change){
```

```
    print($"Change synced: {change}");
}
```

UnsubscribeFromKeepSyncCallback()

```
public void UnsubscribeFromKeepSyncCallback(string dropboxPath, Action<EntryChange> synced
Callback)
```

Unsubscribe specified callback from getting synced changes (if there will be no callbacks listening then syncing will automatically stop as well)

Parameter	Description
dropboxPath	File or folder path on Dropbox
syncedCallback	Callback that you wish to unsubscribe

StopKeepingInSync()

```
public void StopKeepingInSync(string dropboxPath)
```

Stop keeping in sync Dropbox file or folder

Parameter	Description
dropboxPath	File or folder path on Dropbox

IsKeepingInSync()

```
public bool IsKeepingInSync(string dropboxPath)
```

Checks if currently keeping Dropbox file of folder in sync

Parameter	Description
dropboxPath	File or folder path on Dropbox

Operations

CreateFolder()

```
public async void CreateFolder(string dropboxFolderPath, Action<Metadata> successCallback,
Action<Exception> errorCallback, bool autorename = false)
```

Creates folder on Dropbox

Parameter	Description
dropboxFolderPath	Folder to create
successCallback	Callback for receiving created folder Metadata
errorCallback	Callback for receiving exceptions
autorename	Should autorename if conflicting paths?

ListFolder()

```
public async void ListFolder(string dropboxFolderPath, Action<List<Metadata>> successCallb
ack, Action<Exception> errorCallback, bool recursive = false)
```

Get contents of the folder on Dropbox

Parameter	Description
dropboxFolderPath	Path to folder on Dropbox
successCallback	Callback for receiving a List of file's and folder's Metadata - contents on the folder
errorCallback	Callback for receiving exceptions
recursive	Include all subdirectories recursively?

Move()

```
public async void Move(string fromDropboxPath, string toDropboxPath, Action<Metadata> succ
essCallback, Action<Exception> errorCallback, bool autorename = false)
```

Move file or folder from one path to another

Parameter	Description
fromDropboxPath	From where to move
toDropboxPath	Where to move
successCallback	Callback for receiving moved object Metadata
errorCallback	Callback for receiving exceptions
autorename	Should autorename if conflicting paths?

Delete()

```
public async void Delete(string dropboxPath, Action<Metadata> successCallback, Action<Exce
ption> errorCallback)
```

Delete file or folder on Dropbox

Parameter	Description
dropboxPath	Path to delete
successCallback	Callback for receiving deleted object Metadata
errorCallback	Callback for receiving exceptions

PathExists()

```
public async void PathExists(string dropboxPath, Action<bool> successCallback, Action<Exce
ption> errorCallback)
```

Checks if file or folder exists on Dropbox

Parameter	Description
dropboxPath	Path to file or folder
successCallback	Callback for receiving boolean result
errorCallback	Callback for receiving exceptions

GetMetadata()

```
public async void GetMetadata(string dropboxPath, Action<Metadata> successCallback, Action
<Exception> errorCallback)
```

Get Metadata for file or folder on Dropbox

Parameter	Description
dropboxPath	Path to file or folder
successCallback	Callback for receiving file's or folder's Metadata
errorCallback	Callback for receiving exceptions

ShouldUpdateFileFromDropbox()

```
public async void ShouldUpdateFileFromDropbox(string dropboxFilePath, Action<bool> success
Callback, Action<Exception> errorCallback)
```

Checks if Dropbox has different version of the file (always returns true if file is not cached locally)

Parameter	Description
dropboxFilePath	Path to file on Dropbox
successCallback	Callback for receiving boolean result
errorCallback	Callback for receiving exceptions