# API: Dropbox

## Overview

Dropbox allows its users to bring their photos, documents, and videos with them anywhere and share them easily. A cloud-based service, Dropbox also offers its users convenience and accessibility to their files and the ability to modify files and have the changes synced across all devices. The Dropbox API consists of three main APIs:

* Core API – Allows user to read and write files in Dropbox.
* Datastore API – Keeps application’s structured data in sync.
* Sync API – Works with mobile applications that require offline caching and syncing.

The Dropbox API as integrated with the Location-Based Services Module allows users to easily access and create files on the go. The Dropbox API also integrates with the Social module, since it allows users to share pictures and files to other users within their network. The specific applications of the API calls Dropbox offers will be detailed in the section below.

The current version of the Dropbox API supports older versions as long as the client avoids passing undocumented parameters. The API uses OAuth 2.0 protocol to authenticate API calls, but also supports OAuth 1.0.

## Structure and API Calls

The Dropbox API is structured around three sub-APIs: Core API, Datastore API, and Sync API. The three APIs and their API calls are detailed in the sections below.

### Core API

The Core API is the underlying interface for all official Dropbox mobile applications and SDKs and is the most direct way to access the API. The API may evolve in the future, but the Core API supports older versions by maintaining the return values of the API calls with given parameter values. The API requests are required to be done over SSL and all calls are UTF-8 encoded. In addition, the API requires that all dates be formatted in the standard “%a, %d 5b %Y %H:%M:%S %z”, which is supported by all languages that support strftime or strptime.

#### Error Handling

Errors are returned using standard HTTP error code syntax. The standard API errors are as follows.

* 400 – Bad input parameter. Error message should indicate which one and why.
* 401 – Bad or expired token. This may occur if the user or Dropbox revoked or expired an access token. The user must be re-authenticated to fix the issue.
* 403 – Bad OAuth request. Re-authenticating the user will not help here.
* 404 – File or folder not found at the specified file path.
* 405 – Request method not expected (should generally be GET or POST).
* 429 – Application is making too many requests, and is being rate limited. This limitation can be exceeded on a per-app or per-user basis.
* 503 – If the response includes the Retry-After header, the OAuth 1.0 app is being rate limited. Otherwise, the error code indicates a server error, and the application should retry its request.
* 507 – User is over the Dropbox storage quota.
* 5xx – Server error.

#### OAuth 1.0

OAuth 1.0 is supported for all API requests, but OAuth 2.0 is now preferred.

* “/oauth/request\_token” – Step 1 of authentications. Obtains an OAuth request token to be used for the rest of the authentication process.
* “/oauth/authorize” – Step 2 of authentication. Web endpoint that lets the user sign in to Dropbox and choose whether to grant the application the ability to access files on their behalf. The parameters are as follows:
  + Oauth\_token – Request token obtained via /oauth/request\_token (required).l
  + Oauth\_callback – User is redirected to this URL after authorizing or disallowing the application.
  + Locale – If locale specified is a supported language, Dropbox will direct users to the translated version of the authorization website.
  + Disable\_signup – When true, users will not be able to sign up for a Dropbox account via the authorization page (default is false).
* “/oauth/access\_token” – Step 3 of authentication. Application acquires an access token.

#### OAuth 2.0

This is the preferred method of authenticating API requests.

* “/oauth2/authorize” – Starts the authorization flow. The parameters are as follows:
  + Response\_type – Grant type requested, either token or code (required).
  + Client\_id – App’s key, found in the App Console (required).
  + Redirect\_uri – Where to redirect the user after the authorization has completed.
  + State – Up to 500 bytes of arbitrary data that will be passed back to your redirect URI. Used to protect against cross-site request forgery (CSRF).
  + Force\_reapprove – Whether or not to force the user to approve the app again if they’ve already done so (default is false).
  + Disable\_signup – When true, the user will not be able to sign up for a Dropbox account via the authorization page.
* “/oauth2/token” – Acquires a bearer token once the user has authorized the app. Returns a JSON-encoded dictionary including an access token, token type, and Dropbox user ID.
* “/oauth2/token\_from\_oauth1” – Used by apps transitioning from OAuth 1 to OAuth 2. Returns an OAuth 2 token for the authenticated user.

#### Access Tokens

* “disable\_access\_token” – Disables the access token used to authenticate the call. Returns an empty JSON dictionary to indicate success. Works with both OAuth 1 and OAuth 2.

#### Dropbox Accounts

* “/account/info” – Retrieves information about the user’s account. Return values include the following:
  + Uid – User’s unique Dropbox ID.
  + Display\_name – User’s display name.
  + Name\_details/given\_name – User’s given name.
  + Name\_details/surname – User’s surname.
  + Name\_details/familiar\_name – Locale-dependent familiar name for the user.
  + Referral\_link – User’s referral link.
  + Country – User’s two-letter country code, if applicable.
  + Locale – Locale preference set by the user.
  + Is\_paired – If true, then there is a paired account associated with the user.
  + Team – Contains team information if the user belongs to a team.
  + Team/name – Name of the team the user belongs to.
  + Team/team\_id – ID of the team the user belongs to.
  + Quota\_info/normal – User’s used quota outside of shared folders (bytes).
  + Quota\_info/shared – User’s used quota in shared folders (bytes).
  + Quota\_info/quota – User’s total quota allocation (bytes).

#### Files and Metadata

* “/files” – Downloads a file (GET call).
* “/files/put” – Uploads a file. Parameters include the following:
  + Locale – Metadata returned on a successful upload will have its size field translated based on the given locale.
  + Overwrite – If true, any existing file will be overwritten by this upload (default is true).
  + Parent\_rev – Specifies the revision of the file being edited. If parent\_rev matches the latest version of the file on the user’s Dropbox, the file will be replaced.
  + Autorename – If true, the file being uploaded will be automatically renamed to avoid the conflict.
* “/metadata” – Retrieves file and folder metadata. The parameters are as follows:
  + File\_limit – When listing a folder, the service will not report listings containing more than the specified number of files (default is 10,000, max is 25,000).
  + Hash – Hashing of all the metadata.
  + List – If true, the folder’s metadata will include a contents field with a list of metadata entries for the contents of the folder (default is true).
  + Include\_deleted – Only applicable when list is set. If true, the contents will include the metadata of deleted children.
  + Rev – If set, only the metadata for that revision will be returned.
  + Locale – Metadata returned will have its size field translated based on the given locale.
  + Include\_media\_info – If true, each file will include a photo\_info dictionary for photos and a video\_info dictionary for videos with additional media info.
  + Include\_membership – If true, metadata for a shared folder will include a list of members and groups.

Returns the metadata for the file of folder at the given path. Return values include the following:

* + Size – Description of the file size as translated by locale.
  + Bytes – File size in bytes.
  + Path – Canonical path to the file or folder.
  + Is\_dir – Whether the given entry is a folder or not.
  + Is\_deleted – Whether the given entry is deleted.
  + Rev – Unique identifier for the current revision of a file.
  + Hash – Folder’s hash; useful for determining changes to the folder’s contents.
  + Thumb\_exists – True if the file is an image that can be converted to a thumbnail.
  + Photo\_info – Only returned when the include\_media\_info parameter is true and the file is an image.
  + Video\_info – Only returned when the include\_media\_info parameter is true and the file is a video.
  + Icon – Name of the icon used to illustrate the file type.
  + Modified – Last time the file was modified on Dropbox in standard date format.
  + Client\_mtime – For files, modification time set by the desktop client when the file was added to Dropbox in standard date format.
  + Root – Root or top-level folder depending on access level. All paths returned are relative to this root level.
  + Revision – Deprecated field that semi-uniquely determines a file (use rev instead).
  + Shared\_folder – Included for shared folders. Dictionary with the field id, and the fields membership and groups if the include\_membership parameter is passed.
  + Read\_only – For shared folders, specifies whether the user has read-only access to the folder.
  + Parent\_shared\_folder\_id – For files within a shared folder, specifies the ID of the containing shared folder.
  + Modifier – For files within a shared folder, specifies which user last modified the file.
* “/delta” – Helps you keep up with changes to files and folders in the user’s Dropbox. Returns a list of delta entries, which are instructions on how to update the local state to match the server state. Parameters include the following:
  + Cursor – Keeps track of the current state. Determines which delta entries to return (the ones that have been recorded since the cursor was returned).
  + Locale – Metadata returned will have its size field translated based on the given locale.
  + Path\_prefix – If present, filters the response to only include entries at or under the specified path.
  + Include\_media\_info – If true, each file will include a photo\_info dictionary for photos and a vide\_info dictionary for videos with additional media info.

Returns a JSON object with four fields:

* + Entries – List of delta entries.
  + Reset – If true, clears the local state before processing the delta entries.
  + Cursor – String that encodes the latest information that has been returned.
  + Has\_more – If true, then there are more entries available. Call /delta again immediately to retrieve those entries. If false, wait at least five minutes before checking again.

The delta entries returned by the call are 2-item lists of the following forms:

* + [<path>, <metadata>] – Indicates that there is a file/folder at the given path. If the new entry includes parent folders that don’t yet exist in the local state, create the parent folders in the local state. If the new entry is a file, replace any existing file at path with the new entry. If the new entry is a folder, check what the local state has at <path>. If it is a file, replace it with the new entry. If it is a folder, apply the new <metadata> to the folder. If the path does not yet exist, create it as a folder. If the new entry is a folder with the read\_only field set to true, apply the read\_only permission recursively to all files within the shared folder.
  + [<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.
* “/delta/latest\_cursor” – Gets cursor for the server’s state.
* “/longpoll\_delta” – Long-poll endpoint to wait for changes on an account. Provides low-latency way to monitor an account for file changes. Parameters include:
  + Cursor – Delta cursor as returned from a call to /delta.
  + Timeout – Optional integer indicating a timeout, in seconds (default is 30 seconds, min is 30 seconds, max is 480 seconds).
* “/revisions” – Obtains metadata for the previous revisions of a file. Parameters include:
  + Rev\_limit – Number of recent revisions that will be returned (default is 10, max is 1,000).
  + Locale – Metadata returned will have its size field translated based on the given locale.
* “/restore” – Restores a file path to a previous revision. Parameters include:
  + Rev – Revision of the file to restore.
  + Locale – Metadata returned will have its size field translated based on the given locale.
* “/search” – 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. Parameters include:
  + Query – Search string.
  + File\_limit – Number of search results to be returned (default and max are 1,000).
  + Incldue\_deleted – If true, then files and folders that have been deleted will also be included in the search.
  + Locale – Metadata returned will have its size field translated based on the given locale.
  + Include\_membership – If true, metadata for a shared folder will include a list of members and groups.
* “/shares” – Creates and returns a shared link to a file or folder. Parameters include:
  + Locale – Use to specify language settings for user error messages and other language-specific text.
  + Short\_url – When true, the URL returned will be shortened using the Dropbox URL shortened (default is true).
* “/media” – Returns link directly to a file. Similar to /shares, except /media bypasses the Dropbox webserver. Parameters include:
  + Locale – Use to specify language settings for user error messages and other language-specific text.
* “/copy\_ref” – Creates and returns a copy\_ref to a file. Reference string can be used to copy the file to another user’s Dropbox.
* “/thumbnails” – Gets a thumbnail for an image. Parameters include:
  + Format – For images that are photos (default is jpeg, png also acceptable).
  + Size – Either xs, s, m, l, or xl (default is s).
* “/previews” – Gets a preview for a file. Parameters include:
  + Rev – Revision of the file to retrieve.
* “/chunked\_upload” – Uplaods large files to Dropbox in multiple chunks. This allows for uploads larger than the /files\_put maximum of 150 MB. Parameters include:
  + Upload\_id – Unique ID of the in-progress upload on the server.
  + Offset – Byte offset of the chunk relative to the beginning of the full file.
* “/commit\_chunked\_upload” – Completes an upload initiated by the /chunked\_upload method. Saves a file uploaded via /chunked\_upload to a user’s Dropbox. Parameters include:
  + Locale – Metadata returned on a successful upload will have its size field translated based on the given locale.
  + Overwrite – Determines whether an existing file will be overwritten by this upload (default is true).
  + Parent\_rev – Specifies the revision of the file being edited.
  + Autorename – Determines what happens when there is a conflict.
* “/shared\_folders” – Returns a list of all shared folders the authenticated user has access to or metadata about a specific shared folder. Returns a list of shared folders metadata objects.

#### File Operations

* “/fileops/copy” – Copies a file or folder to a new location. Parameters include:
  + Root – Root relative to which from\_path and to\_path are specified.
  + From\_path – File or folder to be copied from relative to root.
  + To\_path – Destination path relative to root.
  + Locale – Metadata returned will have its size field translated based on the given locale.
  + From\_copy\_ref – Specifies copy\_ref generated from a previos /copy\_ref call.
* “/fileops/create\_folder” – Creates a folder. Parameters include:
  + Root – Root relative to which path is specified.
  + Path – path to the new folder to create relative to root.
  + Locale – Metadata returned will have its size field translated based on the given locale.
* “/fileops/delete” – Deletes a file or folder. Parameters include:
  + Root – Root relative to which path is specified.
  + Path – Path to the file or folder to be deleted.
  + Locale – Metadata returned will have its size field translated based on the given locale.
* “/fileops/move” – Moves a file or folder to a new location. Parameters include:
  + Root – Root relative to which from\_path and to\_path are specified.
  + From\_path – File or folder to be moved from relative to root.
  + To\_path – Destination path relative to root.
  + Locale – Metadata returned will have its size field translated based on the given locale.

### Datastore API

The Datastore API keeps track of an app’s per-user data, such as settings, bookmarks, or game stats, in sync across multiple devices and operating systems. Datastores are simple embedded databases, which are synced to Dropbox. Dropbox apps can work with both datastores and files at the same time, and the Datastore API includes functionality to handle both. Datastores are categorized generally as either local datastores, shared datastores, or private/shareable datastores.

Local datastores can be created without a linked Dropbox account, so they can be used without requiring the user to log in with Dropbox. They are stored on the local device and are not synced to Dropbox. If the user later chooses to link, the datastores will be migrated to the user’s Dropbox account. If the user uses multiple devices, the data will be merged once the devices link. Local datastores are stored in a compressed delta format, which is a minimal set of changes which have the same effect as the operations your app performed on a datastore.

Shared datastores can be shared across multiple Dropbox accounts. The unit of sharing is a single datastore, but one or more datastores may be shared between accounts. Any Dropbox account with the correct permissions will be able to open the shared datastore by ID. There are two available principals to whom you may apply a role:

* DBPrincipalPublic – Role will apply to all Dropbox users.
* DBPrincipalTeam – Role will apply to everyone on the user’s team (only applicable for Dropbox for Business accounts).

There are four available roles:

* DBRoleNone – Principal has no access to this datastore.
* DBRoleViewer – Principal is able to view this datastore.
* DBRoleEditor – Principal is able to edit this datastore
* DBRoleOwner – Principal is the owner of this datastore.

Datastores can be created in two different ways. Datastores with private IDs are created using DBDatastoreManager.openOrCreateDatastore:error:. Two different devices can create datastores with the same private ID while offline, and their data will be merged when they come online. Datastores with shareablele IDs are created using DBDatastoreManager.createDatastore: which allows them to be shared between users. Shareable IDs are unique across Dropbox.

The app can store up to 5MB of data across all its datastores without counting against the user’s storage quota. The size of a datastore is calculated by summing the size of all records, plus 1,000 bytes for the datastore itself. The size of a record is calculated by summing the size of all values in all fields, plus 100 bytes for the record itself. The size of a field is a fixed 100 bytes for the field itself plus: for strings or bytes values, the length in bytes of the value; for List values, 20 bytes for each list element plus the size of each element; for all other types, no additional contribution.

* Maximum record size – 100 KiB
* Maximum number of records per datastore – 100,000
* Maximum datastore size – 10 MiB
* Maximum size of a single –sync: call – 2 MiB

All classes and methods in the API are thread-safe. Changes made to a datastore are atomic and are immediately visible to other threads accessing the same datastore.

#### DBAccountManager

The account manager is responsible for linking new users and persisting account information across runs of your app.

##### Class Methods

* (void)setSharedManager:(DBAccountManager \*)sharedManager – Convenient place to store your app’s account manager.
* (DBAccountManager \*)sharedManager – Convenient place to get your app’s account manager.

##### Properties

* @property (nonatomic, readonly) DBAccount \*linkedAccount – Most recently linked account, or nil if there are no accounts currently linked.
* @property (nonatomic, readonly) NSArray \*linkedAccounts – All currently linked accounts, or nil if there are no accounts currently linked.

##### Instance Methods

* (id)initWithAppKey:(NSString \*)key secret:(NSString \*)secret – Create a new account manager with your app’s app key and secret.
* (void)linkFromController:(UIViewController \*)rootController – Begins the process for linking new accounts.
* (DBAccount \*)handleOpenURL:(NSURL \*)url – Call this method in your app delegate’s -application:openURL:sourceApplication:annotation: method in order to complete the link process.
* (void)addObserver:(id)observer block:(DBAccountManagerObserver)block – Add block as an observer to get called whenever a new account is linked or an existing account is unlinked.
* (void)removeObserver:(id)observer – Use this method to remove all blocks associated with observer.

##### Constants

* typedef void (^DBObserver)() – G eneric block type used for observing changes throughout the Sync API.
* typedef void (^DBAccountManagerObserver)(DBAccount \*account) – Observer for the linkedAccount property.

#### DBAccount

The account represents a particular user who has linked their account to your app.

##### Properties

* @property (nonatomic, readonly) NSString \*userId – User id of the account. This can be used to associate metadata with a given account.
* @property (nonatomic, readonly, getter=isLinked) BOOL linked – Whether the account is currently linked.
* @property (nonatomic, readonly) DBAccountInfo \*info – Information about the user of this account, or nil if no info is available.

##### Instance Methods

* (void)unlink – Unlinks a user’s account from your app.
* (void)addObserver:(id)observer block:(DBObserver)block – Add block as an observer of an account to get notified whenever the account’s linked or info properties change.
* (void)removeObserver:(id)observer – Remove all blocks associated with observer by the addObserver:block: method.

#### DBAccountInfo

Information about a user’s account.

##### Properties

* @property (nonatomic, readonly) NSString \*displayName – Recommended string to display to identify an account.
* @property (nonatomic, readonly) NSString \*username – User’s name.
* @property (nonatomic, readonly) NSString \*orgName – User’s organization’s name if available, or nil otherwise.

#### DBError

DBError class is a subclass of NSError that always has domain set to DBErrorDomain.

##### Instance Methods

* (DBErrorCode)dbErrorCode – Same as code.

##### Constants

* DBErrorInternal – Internal error or assertion in the SDK (Fatal).
* DBErrorCache – Failure accessing local cached data (Fatal).
* DBErrorCache – Attempt to use an object or send a request after shutdown (Fatal).
* DBErrorClosed – Use of an object which has been closed (Fatal).
* DBErrorDeleted – Use of an object which has been deleted. Used for synchronous local state transitions, not remote deletion (see DBErrorNotFound). (Fatal).
* DBErrorBadType – Attempt to access a value of the wrong type (Fatal).
* DBErrorSizeLimit – Exceeding a fixed limit, such as maximum Datastore size. Not used for account quota which is subject to change (see DBErrorQuota). (Fatal).
* DBErrorBadIndex – Bad index into a list (Fatal).
* DBErrorIllegalArgument – Illegal argument to an API method (Fatal).
* DBErrorBadKey – Bad key in an internal map lookup (Fatal).
* DBErrorBadState – An object is in a bad state for an attempted operation (Fatal).
* DBErrorMemory – Out of memory (Fatal).
* DBErrorSystem – Error for the OS, when accessing private files or other OS resources (Fatal).
* DBErrorNotCached – Unable to read a file because it is unavailable in the cache (Fatal).
* DBErrorInvalidOperation – Attempt to perform an illegal operation, such as opening a directory, or deleting the root.
* DBErrorNotFound – File, folder, or datastore does not exist.
* DBErrorExists – Operation failed because the target already exists.
* DBErrorAlreadyOpen – Attempt to open a file or datastore which is already open.
* DBErrorParent – Parent directories are missing or not directories.
* DBErrorDiskSpace – Out of disk space for file storage. Applies to local disk space, not Dropbox quota (see DBErrorQuota).
* DBErrorDisallowed – The app attempted an operation that isn't allowed by its access level, or that the user does not have permission to perform.
* DBErrorFileIO – An error accessing a file (outside of the SDK's cache).
* DBErrorCancelled – An operation was cancelled.
* DBErrorReadOnly – An operation try to act on a read-only file or folder.
* DBErrorNetwork – An error occurred making a network request.
* DBErrorTimeout – A connection timed out.
* DBErrorNoConnection – No network connection available.
* DBErrorSSL – Failure in SSL security or unable to verify the server's SSL certificate. Often caused by an out-of-date clock.
* DBErrorServer – The server reported an error.
* DBErrorAuth – The user has not authorized this app, or has unlinked this app.
* DBErrorQuota – The user's Dropbox space is full.
* DBErrorRequest – Server indicated that a request is invalid.
* DBErrorResponse – The server returned an invalid response.
* DBErrorRetryLater – The client should wait a while and then repeat the request.
* DBErrorParamsNoThumb – No thumbnail is available.

#### DBException

The DBException class is a subclass of NSException that always has name set to DBExceptionName. A DBException is raised by a failure in an API method which indicates programming errors or internal SDK problems.

##### Properties

* @property (nonatomic, readonly) DBError \*error – Information about the error which caused this exception to be raised.

#### DBDatastoreManager

The datastore manager lets you list, create, open, and delete datastores.

##### Class Methods

* (DBDatastoreManager \*)managerForAccount:(DBAccount \*)account – Gets the datastore manager for an account that has been linked via the account manager.
* (DBDatastoreManager \*)localManagerForAccountManager:(DBAccountManager \*)accountManager – Gets the local datastore manager for the accountManager.
* (DBDatastoreManager \*)sharedManager – Convenient place to get your app’s datastore manager.
* (void)setSharedManager:(DBDatastoreManager \*)manager – Set your app’s datastore manager using this method. Retrieve it any time with sharedManager.

##### Properties

* @property (nonatomic, readonly, getter=isShutDown) BOOL shutdown – Whether the datastore manager is currently shut down.
* @property (nonatomic, readonly) DBAccount \*account – The account object this manager was created with. Will be nil if this is the local manager.
* @property (nonatomic, readonly) BOOL isLocal – Whether this is the local manager.

##### Instance Methods

* (DBDatastore \*)openDefaultDatastore:(DBError \*\*)error – Opens the default datastore for this account, or creates it if it doesn’t exist.
* (NSArray \*)listDatastores:(DBError \*\*)error – Lists the DBDatastoreInfo for each of the user’s datastores, including the default datastore if it has been created.
* (NSDictionary \*)listDatastoreInfo:(DBError \*\*)error – Gets a map of ID to the [DBDatastoreInfo](https://www.dropbox.com/developers/datastore/docs/ios#DBDatastoreInfo) for each of the user’s datastores, including the default datastore if it has been created.
* (DBDatastoreManager \*)migrateToAccount:(DBAccount \*)account error:(DBError \*\*)error – Returns a new DBDatastoreManager created by migrating a local DBDatastoreManager to the given account.
* (DBDatastore \*)openDatastore:(NSString \*)datastoreId error:(DBError \*\*)error – Open an existing datastore by its ID.
* (DBDatastore \*)createDatastore:(DBError \*\*)error – Creates and opens a new datastore with a unique ID.
* (DBDatastore \*)openOrCreateDatastore:(NSString \*)datastoreId error:(DBError \*\*)error – Opens the datastore with the given ID, creating it if it does not already exist.
* (BOOL)deleteDatastore:(NSString \*)datastoreId error:(DBError \*\*)error – Deletes a datastore with the given ID.
* (BOOL)uncacheDatastore:(NSString \*)datastoreId error:(DBError \*\*)error – Removes a datastore from the local cache.
* (void)addObserver:(id)obj block:(DBObserver)block – Add a block to be called when a datastore is added or removed.
* (void)removeObserver:(id)obj – Remove all blocks associated with the given observer.
* (void)shutdown – Shuts down the datastore manager, which stops all syncing.

#### DBDatastore

A datastore is a simple, syncable database for app data. Interactions with data in the datastore are done through tables. Changes made to the datastore are visible immediately. While a datastore is open, it will monitor for remote changes and download them when possible.

##### Class Methods

* (BOOL)isValidId:(NSString \*)datastoreId – Returns YES if datastoreId is a valid ID for a DBDatastore, or NO otherwise.
* (BOOL)isValidShareableId:(NSString \*)datastoreId – Returns YES if datastoreId is a valid ID for a shareable DBDatastore, or NO otherwise.
* (DBDatastore \*)openDefaultStoreForAccount:(DBAccount \*)account error:(DBError \*\*)error – Opens the default datastore for this account.
* (DBDatastore \*)openDefaultLocalStoreForAccountManager:(DBAccountManager \*)accountManager error:(DBError \*\*)error – Opens the local default datastore for this account manager.

##### Properties

* @property (nonatomic, copy) NSString \*title – Set the title for this datastore. Will be nil if no title is set. Setting it to nil will delete the title field.
* @property (nonatomic, readonly) NSDate \*mtime – The last modified time for this datastore, or nil if no data has been synced yet.
* @property (nonatomic, readonly) NSUInteger size – The current size of this datastore in bytes.
* @property (nonatomic, readonly) NSUInteger recordCount – The total number of records in this datastore.
* @property (nonatomic, readonly) NSUInteger unsyncedChangesSize – The size in bytes of changes that will be queued for upload by the next call to sync:.
* @property (nonatomic, readonly, getter=isOpen) BOOL open – Whether the datastore is currently open.
* @property (nonatomic, readonly) DBDatastoreStatus \*status – The current sync status of the datastore.
* @property (nonatomic, readonly) NSString \*datastoreId – The ID for this datastore.
* @property (nonatomic, readonly) DBDatastoreManager \*manager – The datastore manager for this datastore.
* @property (nonatomic, readonly) DBRole effectiveRole – The effective role the current user has for this datastore.
* @property (nonatomic, readonly) BOOL isWritable – Whether this datastore can be written (i.e., role is owner or editor).
* @property (nonatomic, readonly) BOOL isShareable – Whether this datastore can be shared.

##### Instance Methods

* (void)close – Close a datastore when you’re done using it to indicate that you are no longer interested in receiving updates for this datastore.
* (NSArray \*)getTables:(DBError \*\*)error – Get all the tables in this datastore that contain records.
* (DBTable \*)getTable:(NSString \*)tableId – Get a table with the specified ID, which can be used to insert or query records. If this is a new table ID, the table will not be visible until a record is inserted.
* (NSDictionary \*)sync:(DBError \*\*)error – Apply all outstanding changes to the datastore, and also incorporate remote changes in.
* (void)addObserver:(id)observer block:(DBObserver)block – Add block as an observer when the status of the datastore changes.
* (void)removeObserver:(id)observer – Remove all blocks registered for the given observer.
* (DBRole)getRoleForPrincipal:(NSString \*)principal – Get the role specified by the ACL for a principal (shareable datastores only).
* (void)setRoleForPrincipal:(NSString \*)principal to:(DBRole)role – Assign a role to a principal in the ACL (shareable datastores only).
* (void)deleteRoleForPrincipal:(NSString \*)principal – Delete any role for a principal from the ACL(shareable datastores only).
* (NSDictionary \*)listRoles – Return the ACL in the form of a mapping from principals to roles (as NSIntegers).

##### Constants

* NSUInteger DBDatastoreSizeLimit – Maximum size in bytes of a datastore.
* NSUInteger DBDatastoreUnsyncedChangesSizeLimit – Maximum size in bytes of changes that can be queued up between calls to sync:.
* NSUInteger DBDatastoreRecordCountLimit – Maximum number of records in a datastore.
* NSUInteger DBDatastoreBaseSize – Size in bytes of a datastore before accounting for the size of its records.
* NSUInteger DBDatastoreBaseUnsyncedChangesSize – Size in bytes of unsynced changes before accounting for the size of each change.
* NSUInteger DBDatastoreBaseChangeSize – Size in bytes of a change before accounting for the size of its values.
* typedef enum DBRole - Enum giving the possible roles a principal can have with respect to a DBDatastore.
* NSString \* const DBPrincipalTeam – Principal used to set or retrieve the role for Dropbox for Business team members.
* NSString \* const DBPrincipalPublic – Principal used to set or retrieve the role for the general public.

#### DBDatastoreInfo

The datastore info class contains basic information about a datastore.

##### Properties

* @property (nonatomic, readonly) NSString \*datastoreId – ID for this datastore.
* @property (nonatomic, readonly) NSString \*title – Title for this datastore, or nil if none is set.
* @property (nonatomic, readonly) NSDate \*mtime – Last modified time for this datastore, or nil if none is set.
* @property (nonatomic, readonly) DBRole role – Role the current user has for this datastore.
* @property (nonatomic, readonly) BOOL isShareable – Whether this datastore is shareable.
* @property (nonatomic, readonly) BOOL isWritable – Whether this datastore can be written (i.e., role is owner or editor).

#### DBDatastoreStatus

Sync status for a DBDatastore, including any errors that are preventing syncing.

##### Properties

* @property (nonatomic, readonly) BOOL connected – Whether the API is in active communication with the server so that remote changes are likely to be visible quickly, and local changes can be uploaded soon.
* @property (nonatomic, readonly) BOOL downloading – Whether there are remote changes that need to be downloaded from the server.
* @property (nonatomic, readonly) BOOL uploading – Whether there are local changes that need to be uploaded to the server. Always set for a local datastore that has any changes at all.
* @property (nonatomic, readonly) BOOL incoming – Whether there are remote changes that will be incorporated by the next call to – [DBDatastore sync:].
* @property (nonatomic, readonly) BOOL outgoing – Whether there are local changes that haven’t yet been committed by a call to – [DBDatastore sync:].
* @property (nonatomic, readonly) BOOL needsReset – Whether the local datastore needs to be reset with a call to – [DBDatastore close:] followed by – [DBDatastoreManager uncacheDatastore:].
* @property (nonatomic, readonly) DBError \*uploadError – Latest error preventing local datastore state from being uploaded, or nil if there is no error.
* @property (nonatomic, readonly) DBError \*downloadError – Latest error preventing remote datastore state from being downloaded, or nil if there is no error.
* @property (nonatomic, readonly) DBError \*anyError – An error (downloadError or uploadError) affecting this datastore, or nil if there is no error.

#### DBTable

A collection of records that lets you query for existing records or insert new ones. In addition to querying and inserting records, you can also set custom conflict resolution rules.

##### Class Methods

* (BOOL)isValidId:(NSString \*)tableId – Returns YES if tableId is a valid ID for a DBTable, or NO otherwise.

##### Properties

* @property (nonatomic, readonly) NSString \*tableId – ID of the table.
* @property (nonatomic, readonly) DBDatastore \*datastore – Datastore that contains this table.

##### Instance Methods

* (NSArray \*)query:(NSDictionary \*)filter error:(DBError \*\*)error – Returns records matching the provided filter, or all records if filter is nil.
* (DBRecord \*)getRecord:(NSString \*)recordId error:(DBError \*\*)error – Returns a record with the given recordId, or nil if that record doesn’t exist or an error occurred.
* (DBRecord \*)getOrInsertRecord:(NSString \*)recordId fields:(NSDictionary \*)fieldsinserted:(BOOL \*)inserted error:(DBError \*\*)error – Returns a record with the given recordId (unmodified), or inserts a new record with the initial set of fields if it doesn’t exist already.
* (DBRecord \*)insert:(NSDictionary \*)fields – Insert a new record with the initial set of fields into this table with a unique record ID.
* (void)setResolutionRule:(DBResolutionRule)rule forField:(NSString \*)field – Sets pattern as the resolution pattern for conflicts involving the given fieldname.

##### Constants

* typedef enum DBResolutionRule – Enum to specify how conflicts are resolved on a field.
  + DBResolutionRemote – Resolves conflicts by always taking the remote change.
  + DBResolutionLocal – Resolves conflicts by always taking the local change.
  + DBResolutionMax – Resolves conflicts by taking the largest value, based on type-specific ordering.
  + DBResolutionMin – Resolves conflicts by taking the smallest value, based on type-specific ordering.
  + DBResolutionSum – Resolves conflicts by preserving additions or subtractions to a numeritcal value, which allows you to treat it as a counter or accumulator without losing updates.

#### DBRecord

A record represents an entry in a particular table and datastore. Each record has a unique ID, and contains a set of fields, each of which has a name and a value. Fields can hold values of the following types: NSNumber, NSString, NSData, NSDate, and NSArray. Changes to the record are immediately visible to other record objects.

##### Class Methods

* (BOOL)isValidId:(NSString \*)recordId – Returns YES if recordId is a valid ID for a DBRecord, or NO otherwise.
* (BOOL)isValidFieldName:(NSString \*)name – Returns YES if name is a valid name for a field in a DBRecord, or NO otherwise.

##### Properties

* @property (nonatomic, readonly) NSString \*recordId – ID of the record.
* @property (nonatomic, readonly) DBTable \*table – Table that contains this record.
* @property (nonatomic, readonly) NSDictionary \*fields – Fields of this record.
* @property (nonatomic, readonly) NSUInteger size – Size of this record in bytes.
* @property (nonatomic, readonly, getter=isDeleted) BOOL deleted – Whether this record is deleted.

##### Instance Methods

* (id)objectForKey:(NSString \*)key – Get the value of a single field.
* (DBList \*)getOrCreateList:(NSString \*)fieldname – Returns the current list at the given field, or returns an empty list if no value is set.
* (void)update:(NSDictionary \*)fieldsToUpdate – Update all the fields in the provided dictionary with the values that they map to.
* (void)setObject:(id)obj forKey:(NSString \*)fieldname – Update a single field with the provided value.
* (void)removeObjectForKey:(NSString \*)fieldname – Remove a single field from the record.
* (void)deleteRecord – Delete this record.

##### Constants

* NSUInteger DBRecordSizeLimit – Maximum size in bytes of a record.
* NSUInteger DBRecordBaseSize – The size in bytes of a record before accounting for the size of its fields.
* NSUInteger DBFieldBaseSize – Size in bytes of a field before accounting for the sizes of its values.
  + NSString andNSData – Length in bytes of the value.
  + NSArray – Sum of the size of each list item, where each item's size is computed as the size of the item value plus DBListItemBaseSize.
  + Other types – No additional contribution to the size of the field.

#### DBList

An object that allows you to modify a list that is set as a value on a record. Lists can contain the same values as records, except for other lists. Any changes you make to the list are intelligently merged with changes made remotely.

##### Properties

* @property (nonatomic, readonly) NSArray \*values – Returns all objects in the list.

##### Instance Methods

* (NSUInteger)count – Returns the total number of items in the list.
* (id)objectAtIndex:(NSUInteger)index – Returns the object at the given index.
* (void)insertObject:(id)obj atIndex:(NSUInteger)index – Inserts an object at the given index, moving other objects further down the list.
* (void)removeObjectAtIndex:(NSUInteger)index – Removes the object at the given index.
* (void)addObject:(id)obj – Adds an object to the end of the list.
* (void)removeLastObject – Removes the last object from the list.
* (void)replaceObjectAtIndex:(NSUInteger)index withObject:(id)obj – Replaces the item at the given index with the given object.
* (void)moveObjectAtIndex:(NSUInteger)oldIndex toIndex:(NSUInteger)newIndex – Moves the object from the given old index, so that it appears at the given new index.

##### Constants

* NSUInteger DBListItemBaseSize – Size in bytes of a list item before accounting for the size of its value

## Technical Constraints

## Summary