Box iOS API Documentation

The Box iOS API is designed to allow developers to easily produce iOS apps that interact with the Box.net cloud services. This document gives an overview of the API.

API Overview

The Box iOS API is broken into two distinct pieces: models, which represent objects on the Box.net servers like folders, comments, and updates, and operations, which represent interactions with the Box.net server like logging in, retrieving folder information, and posting comments. The following sections will describe each of these pieces in detail.

Note that every application that accesses Box.net must have a unique Box API key. To retrieve this key, go to: https://www.box.net/developers/services and click "Create New Application" After that is completed, an API key will be assigned to your application, and this API key must be place in BoxRESTAPIFactory.m.

For more general information about the Box web API, please see:
http://developers.box.net/w/page/12923958/FrontPage

Box Models

Box Models are used to represent local copies of objects on the Box.net servers, like folders, files, and comments. In general, they are built using the operations described in the next section. A small amount of code is provided for persistifying objects so that they may be saved between sessions or offline.

Users

The currently logged in user is represented by the `BoxUser` class, which stores the user name, email, and other parameters of the user account like the max quota. Note that the password is never seen by the app, and thus is not accessible in the `BoxUser` object. Instead, the auth token is used for authentication and is needed by almost every network operation.

A number of convenience functions are provided, including the ability to save the user to a text file on disk, load the user from disk, and check if the user is logged in with proper credentials. Users are typically built via either the registration or login operations.

Comments

Comments on files and folders are represented by the BoxComment class, which stores information about the comment message, the user who created the comment, and the time at which the comment was created. Comments are typically built via the BoxGetCommentsOperation.

Updates

Folder and file updates are represented by the BoxUpdate class, which stores information about the user who caused the update, the update time, and the item that was updated. Updates are typically built via the BoxGetUpdatesOperation.

Files and Folders

Files and folders are represented by BoxFile and BoxFolder objects respectively, both of which inherit from the BoxObject class. Box objects contain generic information such as the object ID (useful for identifying the object with the Box.net servers), the name, associated user and size. Files additionally contain information about the URL of the thumbnail generated by the Box.net servers for that particular file, while folders contain information about their collaboration status and the items within the folder.

There is minimal support in the API for storing folders locally. In particular, a single folder can be stored using the saveAsCurrentFolder and retrieveSavedFolder methods. Generally, folders and files are built using the BoxFolderXMLBuilder via calls of the form

```
+ (BoxFolder *)folderForId:(NSNumber *)
                     token:(NSString *)
          responsePointer:(BoxFolderDownloadResponseType*)
           basePathOrNil:(NSIndexPath *);
```

All of the subitems of a folder are retrieved via this call. In order to retrieve the top level folder ("All files"), simply pass 0 for the folder ID. Note that this call is blocking for the duration of the network operation, and thus should not be made on the main thread.

Box Operations

In general, network operations in the Box iOS API are represented by classes inheriting from BoxOperation, and follow a well-defined lifecycle independent of the operation. Operations are created via a class method call that supplies the operation with all needed information, such as the target folder, the parameters, and the user auth token. The operation is then started either by adding the operation to

an operation queue or by explicitly calling the start method. During execution, the operation delegate receives method calls when the operation begins, when it progresses (if its multistep), and finally when it completes.

For example, consider the BoxGetCommentsOperation, which retrieves comments from the Box.net servers. To create the operation, call, for example:

```
BoxGetCommentsOperation *operation =
[BoxGetCommentsOperation operationForTargetID:0
                                   targetType:@"folder"
                                    authToken:@"myAuthToken"
                                     delegate:self];
```

Note that the delegate is typically the creator of the operation. To launch the operation, simply it add it to a NSOperationQueue:

```
[_operationQueue addOperation:operation];
```

The operation queue can be either the main queue (in which case it will run on the main thread), or an alternative queue, where the operations will run concurrently. This structure makes multithreading of long running tasks particularly easy.

The progress of the operation is passed back to the delegate as the operation executes. The delegate must implement the BoxOperationDelegateProtocol, which has a number of optional methods, the most commonly used of which are:

```
-(void)operation:(BoxOperation *)op willBeginForPath:(NSString
*)path;
```

This is called when the operation is about to execute, and can be used to activate the network activity indicator.

```
- (void)operation:(BoxOperation *)op didCompleteForPath:(NSString
*)path response:(BoxOperationResponse)response;
```

This is called when the operation has completed. The operation response tells whether or not the operation completed successfully, and if it did not complete successfully, contains useful error information. Typically, the operation object itself contains the results of the operation. For example, the BoxGetCommentsOperation has a property comments that is an array of the BoxComments that were received from the Box.net servers.

The full list of API operations is given in the next section.

Login and Registration Operations

BoxRegisterOperation: requires a username and password, and returns (if successful), a configured BoxUser.

BoxGetTicketOperation: returns the ticket used for authentication with Box.net

BoxGetAuthTokenOperation: requires the ticket received from the get ticket operation and returns a configured BoxUser.

The authentication process is described in more detail on the Box.net website here: http://developers.box.net/w/page/12923915/ApiAuthentication

Briefly, the application requests a ticket from the Box.net servers using the get ticket operation. Using the ticket, the application must access a Box.net website that authenticates the user using his or her login credentials. Once that is complete, the get auth token operation retrieves the user information. To make this easier, the Box iOS API code includes the class LoginBuilder, which handles all of this automatically.

File and Folder Manipulation Operations

BoxMoveOperation: requires the item ID, the item type (file or folder), the destination folder ID and the user auth token, and returns whether or not the move succeeded

BoxCopyOperation: requires the item ID, the item type (file or folder), the destination folder ID and the user auth token, and returns whether or not the copy succeeded

BoxRenameOperation: requires the item ID, the item type (file or folder), the new name, and the user auth token, and returns whether or not the rename succeeded

BoxDeleteOperation: requires the item ID, the item type (file or folder) and the user auth token, and returns whether or not the delete succeeded

BoxCreateFolderOperation: requires the new folder name, the parent folder ID, whether or not to share the new folder and the user auth token, and returns the ID of the newly created folder

Sharing Operations

BoxPublicShareOperation: requires the item ID, the item type (file or folder), the password for the link (if needed, otherwise pass nil), the message to include, an

array of the emails to contact, and the user auth token, and returns the URL for the shared link.

BoxPrivateShareOperation: requires the item ID, the item type (file or folder), the message to include, an array of the emails to contact, whether or not to notify the owner when the file is viewed, and the user auth token.

BoxPublicUnshareOperation: requires the item ID, the item type (file or folder), and the user auth token.

Comments Operations

BoxGetCommentsOperation: requires the item ID, the item type (file or folder), and the user auth token, and returns an array of BoxComment objects

BoxAddCommentsOperation: requires the item ID, the item type (file or folder), the comment message, and the user auth token.

BoxDeleteCommentOperation: requires the comment ID (note this is not the item ID) and the user auth token.

Updates Operations

BoxGetUpdatesOperation: requires a start time (only updates after that time will be retrieved) and the user auth token, and returns an array of BoxUpdates

Uploading and Downloading Operations

BoxDownloadOperation: requires the item ID, the local path where the downloaded file should be stored, and the user auth token.

BoxUploadOperation: requires the user model of the uploader, the ID of the parent folder, the data of the file, the filename, the file content type (i.e., the extension), whether the file should be shared, the sharing message (can be nil if not shared), and an array of emails to share the file with (can be nil if not shared).