LIFE MOBILE APPLICATION

APPLICATION PROGRAM INTERFACE: Research

Colin Man, Malina Jiang

VERSION 1.0

Last Update: 26 February 2015

TABLE OF CONTENTS

1 Document Control

1.1 Change Record

Version	Date	Author	Changes
1.0	02-02-15	Colin Man	Draft following overview + description
1.0	02-02-15	Colin Man	Draft Google Maps API
1.0	02-02-15	Malina Jiang	Draft Yelp and Google Drive API

1.2 Definitions

Term	Definition
LMA	Life Mobile Application
LBS	Location Based Services
API	Application Program Interface

Note: "LMA" is consistently used throughout all documents as the name of the mobile application, although it is understood that another name will be chosen for branding purposes.

2 INTRODUCTION

2.1 SCOPE AND PURPOSE

This document contains a description of third party API's that may potentially be incorporated into LMA, in terms of contribution to the business model as well as technical functionality and constraints.

The API's outlined allow interaction with a variety of third-party products that do not define LMA individually, but come together to form a unique platform that encompasses the totality of the mobile user's needs.

The existence of this document is imperative to the success of the product as it clearly defines the constraints of each interface and the extent to which we can leverage the product to create a novel platform. It is crucial to the development of a business plan with a clearly defined direction as well as a viable and efficient implementation. Without such definitions, it is easy to design a product that is impossible to develop on the technical side or assumes third-party functionality that does not exist.

A description of the API's will also help in defining and outlining a minimal viable product, a subset of LMA's features that form the core functionality and can stand alone if necessary.

As such, each third-party API description contains the following components:

- **Overview** – Summary of the features and purpose. This includes a description of the functionality encompassed in the API as well as its contribution to the system.

- **Structure and API Calls** – A technical description of the API and the structure of its interface. Since the way information is obtained varies widely depending on the design of each API, this is vital both in extracting constraints for the business model as well as development of the actual application in the implementation stage.

- **Technical Constraints –** Detailed summary of constraints in terms of information that can be obtained through the application user interface. Includes information such as maximum volume of requests that is critical to making important design decisions.

3 API: Google Maps

3.1 Summary

The Google Maps API is divided into four main types of interfaces, depending on the technology used to implement the product as well as to access the data.

The four API interfaces are:

- **Embed API** – The simplest of the four, which simply uses HTML and iframes to embed content into any website.
- **Web API** – Provides a set of functionality that can be accessed using standard RESTful calls. These calls can be placed from web applications (AJAX) or native applications.
- **JavaScript API –** Provides extended maps functionality in visualization and map content. Can be integrated into platforms that run mainly JavaScript.
- **iOS API** – Native integration of functionality that interfaces with the operating system directly. Can only be used by iPhone applications, but has similar functionality to the JavaScript API.

3.2 Embed API

3.2.1 Overview

The Embed API is a simple way to embed maps into an application. It allows developers to embed iframe elements into the web application and specify various features using parameters in the query portion of the src URL. Thus, information is obtained and displayed to the user through purely HTTP requests and no interaction with the API in between the requests is needed. Technical details are outlined in "Structure and API Calls".

The API allows for four different ways to display maps content:

- **Place Mode** – displays a map pin at a predefined location (landmark, business, geographic feature, or custom pin)
- **Directions Mode** – displays the path between two different points and shows the time and distance estimated.
- **Search Mode** – displays a map that contains the results to a search query with a term specified in the attributes of the iframe tag.

- **View Mode** – Provides a barebones map that contains no visual content apart from the basic map functionality. This can be either a map view or a street view.

Note that the Embed API creates a map that links to Google Maps and provides an interface to work with options that may not be in the code injected through the iframe.

If the user is logged in to their Google account, they will also be able to save the locations from the embedded map into their user account, with custom attribution information that can b specified by the application.

3.2.2 Structure and API Calls

The API call is performed through an HTTP request sent via the "src" attribute of an iframe. The call is structured as follows:

https://www.google.com/maps/embed/v1/MODE?key=API_KEY&parameters

The mode of operation (MODE) is one of place, directions, search, view, or streetview

In order to use the Embed API to obtain maps, the application must first register with Google to obtain an API key (API_KEY) that is included in the url query used to embed the map into the page. This key can be restricted to certain domains to ensure that others do not steal it.

Since the HTML to retrieve the maps is retrieved using a simple HTTP call to the web service, all parameters that have to do with obtaining data must be sent through the query string. Other properties such as width, height, border, etc. can be customized in the iframe and slightly change the rendering of the map.

The possible query parameters to the Embed API (parameters) are as follows:

- attribution_source – string that describes the site or app; allows users to see what source is attributed to saving the place on their maps
- attribution_web_url – the url of the site or app (as attribution information)
- attribution_ios_deep_link_id – a URL scheme that links to the iOS application (using iOS internal links)
- center – the center of the map view (latitude and longitude values)
- zoom – (zoom level from 0 to 21)
- maptype – either "roadmap" or "satellite" depending on the view that should be loaded
- language – the language that should be used in the embedded map
- region – the borders and labels that should be displayed

The mode-specific query parameters are as follows:

- **Place**

    - q – the place to show on the map. This value can either be a location (escaped string) or an address (does not take latitude and longitude since it must be a name location to be highlighted).
- **Direction**
    - origin – the starting point location (as before, in either escaped address format or place name)
    - destination – the end point location (as before, in either escaped address format or place name)
    - waypoints – a set of intermediary places (as before, in either escaped address format or place name) separated by the pipe character
    - mode – method of travel (choose from driving, walking, bicycling, transit, or flying). Options will be shown on map if none is chosen.
    - avoid – any obstacles that the map should avoid (tolls, ferries, highways). Different items can be separated using the pipe character.
    - units – either metric or imperial depending on the units that should be displayed
- **Search**
    - q – the search term that should be shown on the map (in escaped URL format – can include restrictions such as "in Danville near 94526")
- **Street View**
    - pano – the panorama id that should be shown (if not specified, one will be found from the location)
    - heading – the direction that the camera is facing in degrees from due north
    - pitch – the angle of the camera up or down in degrees
    - fov – the horizontal field of view in degrees

3.2.3 Technical Constraints

Using the map modes in the Embed API is not restrictive in terms of the user interaction – the user can move the map around in the iframe and they can switch view modes (Satellite, Maps, etc.). The constraints are mainly on the ease of incorporating dynamic content and on showing or controlling the maps interaction.

It is not possible to dynamically load content into the map since the map is hosted on Google's own servers and we are merely using an iframe to load it. Any attempt to change the "src" of the iframe in order to load the new parameters will cause the entire map to reload, which does not give a good user experience (we want the search to be done within the map interface, not by refreshing and reloading the entire map, which also has a higher latency).

This interface can only be used in a website since there are many limitations with iframes on mobile devices (Safari has compatibility issues, as does the android browser). The iframe served by Google Embed API is also not optimized for mobile, so may not be as easy to use on a phone.

There are no constraints on the volume of API calls that can be made, so the use of this API is essentially unlimited.

3.2.4 Summary

When compatibility and dynamic flexibility is not an issue, using the Google Embed API is the best solution to provide map functionality. It is the easiest to use out of the four API's and provides a very fast integration of maps functionality into a website.

In terms of LMA, the Embed API can be used for various maps services in which the location of a place should remain static for the duration of the webpage load. Examples of this may include: loading directions to a certain place, keeping track of saved routes and places, etc. For additional functionality of Google Maps, we may have to resort to the Web or native API's. See below for a full description.

3.3 Web API

3.3.1 Overview

The Google Maps Web API consists of five separate components that come together to provide a comprehensive set of tools that can be used to identify the location as well as calculate location based values of the user. These API's are in the form of web services and can be called via standard HTTP requests. The return values are either in XML or JSON format. Most calls support both, depending on the technology and preference of the developer.

The five components of the Maps Web API are as follows:

- Directions API
- Distance Matrix API
- Elevation API
- Geocoding API
- Time Zone API

Each of these API's and their calls are described in detail below. They each deal with a separate part of location services and information, and contain separate requirements, constraints, and features.

3.3.2 Directions API

Overview

The Directions API includes the calls that calculate directions between locations that are passed to it. The locations are acquired through a different means (i.e. if using the web API in a mobile app, the locations can be acquired through direct access to the GPS).

This API can also calculate multi-part directions (i.e directions that include routing between more than one different locations in a certain order). It is suggested that this API be used to calculate directions ahead of time since this calculation is time intensive and will optimize the app the most of it can be done before the actual runtime.

Structure and API Calls

In order to begin using the Web API, developers must first obtain the API key provided by Google. This can be done through the developer online console and should be limited to calls from certain IP addresses or domains. HTTPS use is recommended, but not required.

The XML version of the API can be accessed at the following URL:
https://maps.googleapis.com/maps/api/directions/xml?parameters

The JSON version of the API can be accessed at the following URL:
https://maps.googleapis.com/maps/api/directions/json?parameters

Required request parameters

- **origin** – The address or textual latitude/longitude value from which you wish to calculate directions. If you pass an address as a string, the Directions service will geocode the string and convert it to a latitude/longitude coordinate to calculate directions. If you pass coordinates, ensure that no space exists between the latitude and longitude values.
- **destination** – The address or textual latitude/longitude value from which you wish to calculate directions. If you pass an address as a string, the Directions service will geocode the string and convert it to a latitude/longitude coordinate to calculate directions. If you pass coordinates, ensure that no space exists between the latitude and longitude values.

Work users must include valid client and signature parameters with their Directions requests.

Optional parameters

- **mode** – Specifies the mode of transport to use when calculating directions. Valid values and other request details are specified in Travel Modes. Defaults to driving.
- **waypoints** – Specifies an array of waypoints. Waypoints alter a route by routing it through the specified location(s). A waypoint is specified as either a latitude/longitude coordinate or as an address which will be geocoded. Waypoints are only supported for driving, walking and bicycling directions.
- **alternatives** – If true, specifies that the Directions service may provide more than one route alternative in the response. Providing route alternatives may increase the response time from the server.
- **avoid** – Indicates that the calculated route(s) should avoid the indicated features. This parameter supports the following arguments:
    - tolls indicates that the calculated route should avoid toll roads/bridges.
    - highways indicates that the calculated route should avoid highways.
    - ferries indicates that the calculated route should avoid ferries.

- **language** – Specifies the language in which to return results. If language is not supplied, the service will attempt to use the native language of the domain from which the request is sent.
- **units** – Specifies the unit system to use when displaying results.
- **region** – Specifies the region code, specified as a ccTLD ("top-level domain") two-character value.
- departure_time – Specifies the desired time of departure. You can specify the time as an integer in seconds since midnight, January 1, 1970 UTC. Alternatively, you can specify a value of now, which sets the departure time to the current time (correct to the nearest second). The departure time may be specified in two cases:
    - For transit directions: You can optionally specify one of departure_time or arrival_time. If neither time is specified, the departure_time defaults to now (that is, the departure time defaults to the current time).
    - For driving directions: Work users can specify the departure_time to receive trip duration considering current traffic conditions. The departure_time must be set to within a few minutes of the current time.
- **arrival**_time – Specifies the desired time of arrival for transit directions, in seconds since midnight, January 1, 1970 UTC. You can specify either departure_time or arrival_time, but not both. Note that arrival_time must be specified as an integer.
- **transit_mode** – Specifies one or more preferred modes of transit. This parameter may only be specified for transit directions, and only if the request includes an API key or a Work client ID. The parameter supports the following arguments:
    - bus indicates that the calculated route should prefer travel by bus.
    - subway indicates that the calculated route should prefer travel by subway.
    - train indicates that the calculated route should prefer travel by train.
    - tram indicates that the calculated route should prefer travel by tram and light rail.
    - rail indicates that the calculated route should prefer travel by train, tram, light rail, and subway. This is equivalent to transit_mode=train|tram|subway.
- **transit_routing_preference** – Specifies preferences for transit routes. Using this parameter, you can bias the options returned, rather than accepting the default best route chosen by the API. This parameter may only be specified for transit directions, and only if the request includes an API key or a Work client ID. The parameter supports the following arguments:
    - less_walking indicates that the calculated route should prefer limited amounts of walking.
    - fewer_transfers indicates that the calculated route should prefer a limited number of transfers.

Travel Modes – The following transport modes are available when calculating directions from one place to another.

- driving (default) indicates standard driving directions using the road network.
- walking requests walking directions via pedestrian paths & sidewalks (where available).

- bicycling requests bicycling directions via bicycle paths & preferred streets (where available).
- transit requests directions via public transit routes (where available). If you set the mode to transit, you can optionally specify either a departure_time or an arrival_time. If neither time is specified, the departure_time defaults to now (that is, the departure time defaults to the current time). You can also optionally include a transit_mode and/or a transit_routing_preference.

Note: Both walking and bicycling directions may sometimes not include clear pedestrian or bicycling paths, so these directions will return warnings in the returned result which you must display to the user.

Using Waypoints in Routes

When calculating routes using the Directions API, you may also specify waypoints for driving, walking or bicycling directions. Waypoints are not available for transit directions. Waypoints allow you to calculate routes through additional locations, in which case the returned route passes through the given waypoints.

Waypoints are specified within the waypoints parameter and consist of one or more addresses or locations separated by the pipe (|) character.

For example, the following URL initiates a Directions request for a route between Boston, MA and Concord, MA with stopovers in Charlestown and Lexington, in that order:

https://maps.googleapis.com/maps/api/directions/json?origin=Boston,MA&destination=Concord,MA&waypoints=Charlestown,MA|Lexington,MA&key=API_KEY
You can use waypoints to influence your route without adding a stopover by prefixing the waypoint with via:. Waypoints prefixed with via: will not add an additional leg to the route.

The following URL modifies the previous request such that you are routed through Lexington without stopping:

https://maps.googleapis.com/maps/api/directions/json?origin=Boston,MA&destination=Concord,MA&waypoints=Charlestown,MA|via:Lexington,MA&key=API_KEY
By default, the Directions service calculates a route through the provided waypoints in their given order. Optionally, you may pass optimize:true as the first argument within the waypoints parameter to allow the Directions service to optimize the provided route by rearranging the waypoints in a more efficient order. (This optimization is an application of the Travelling Salesman Problem.)

If you instruct the Directions service to optimize the order of its waypoints, their order will be returned in the waypoint_order field within the routes object. The waypoint_order field returns values which are zero-based.

The following example calculates a road trip route from Adelaide, South Australia to each of South Australia's main wine regions using route optimization.

https://maps.googleapis.com/maps/api/directions/json?origin=Adelaide,SA&destination=Adelaide,SA&waypoints=optimize:true|Barossa+Valley,SA|Clare,SA|Connawarra,SA|McLaren+Vale,SA&key=API_KEY
Inspection of the calculated route will indicate that the route is calculated using the following waypoint order:

"waypoint_order": [1, 0, 2, 3]

Technical Constraints

There are technical usage limits in place for the directions API. The limits are as follows:

- Traffic
    - Free Users: 2,500 directions requests per 24 hour period.
    - Work Users: 100,000 directions requests per 24 hour period.
    - Free Users: Up to 8 waypoints allowed in each request. Waypoints are not available for transit directions.
    - Work Users: 23 waypoints allowed in each request. Waypoints are not available for transit directions.
    - Free Users: 2 requests per second.
    - Work Users: 10 requests per second.
- URL
    - API URLs are restricted to approximately 2000 characters, after URL Encoding.

- ***Summary***

3.4 JavaScript API

3.4.1 Overview

3.4.2 Structure and API Calls

3.4.3 Technical Constraints

3.4.4 Summary

3.5 iOS API

3.5.1 Overview

3.5.2 Structure and API Calls

3.5.3 Technical Constraints

3.5.4 Summary

4 API: YELP

4.1 OVERVIEW

Yelp offers its users convenience and accessibility to local businesses wherever they go. In addition to business reviews, Yelp also finds events and lists for the user and allows communication between Yelpers. Yelp offers an unbiased look at the businesses within the surrounding area and offers its feedback to users so that they can make the best decisions about which businesses to patronize. The Yelp API offers the following functionality:

- Find the top results in a geographical location in response to a client query.
- Sort results according to the client query (e.g. by highest to lowest ratings, or greatest to least distance from the client location).
- Limit displayed results according to client specifications (e.g. display only businesses that offer Yelp Deals).
- Display detailed information about a Yelp Deal (e.g. savings, purchase URL).
- Identify whether a claimed has already been claimed on Yelp.com

The Yelp API as integrated with the Location-Based Services Module lends its wide database of businesses and various API calls to the task of locating the best deals in the area for clients of the Life Mobile Application. The specific applications of the API calls Yelp offers will be detailed in the section below.

The current version of the Yelp API is 2.0. Yelp v1.0 is deprecated but will but Yelp does not have any plans to turn it off as of now. Yelp currently limits API Calls to 25,000 calls, but accommodates requests for more calls. The API uses the standard, secure authorization protocol OAuth 1.0a, xAuth and offers two main sub-API's – the Search API and the Business API. As LMA focuses mainly on consumer users instead of business users, use of the Yelp API will mainly be constrained to the Business API.

4.2 STRUCTURE AND API CALLS

The Yelp API is limited to one type of API call, namely the GET request. Yelp authenticates each request using OAuth per the usual standard.

GET – Allows the client to search for local businesses. The API request takes in the following optional parameters:

- Term – Search term inputted by the client. If not included, the API will search everything.
- Limit – Number of search results to return.
- Offset – Offset the list of return businesses by this amount.
- Sort – Determines what the data will be sorted by. Options include 0 (for best matched), 1 (by distance), or 2 (most highly rated). When returning results, the

business rating is adjusted for the number of ratings to give a more comprehensive view of the business quality.
- Category_filter – Applies filter to search results.
- Radius_filter – Determines the search radius of the query, within 25 miles.
- Deals_filter – Decides whether to search only for businesses with Yelp Deals.

As Yelp is also a location-based service, Yelp API requests have one required parameter:

- Location – Specifies the location to be used when conducting a search. Locations can specified with an address, neighborhood, city, state, zip, or county. They can additionally be specified with geographical coordinates:

    - Cll Parameters – These parameters are formatted as "cll = latitude, longitude".
    - Bound Parameters – These parameters form a geographical bounding box and are formatted as "bounds = sw_latitude, sw_longitude | ne_latitude, ne_longitude".
    - Ll Parameters –These parameters are formatted as "ll = latitude, longitude, accuracy, altitude, altitude_accuracy".

In response to the API request, the client receives the following:

- Region – Suggested bounds for map display.
- Total – Number of businesses in search result.
- Businesses – List of business entries that fulfill the search parameters. Each business entry has the following attributes.

    - Id – Yelp Business ID.
    - Is_claimed – Whether the business has already been claimed.
    - Is_closed – Whether the business has been permanently closed.
    - Name – Business name.
    - Image_url – Business photo URL.
    - Url – Business Yelp page URL.
    - Mobile_url – Mobile business Yelp page URL.
    - Phone – Business phone number.
    - Display_phone – Phone number formatted for display.
    - Review_count – Number of reviews on the business.
    - Categories – List of category names the business is associated with.
    - Distance – Distance from search location in meters.
    - Rating – Business rating. Includes rating image associated with the business.
    - Snippet_text – Snippet text associated with the business.
    - Snippet_image_url – URL of snippet image associated with the business.

- Location – Location data of the business. Includes address, city, state, zip, country, neighborhood and other location information of interest.
- Deals – Deals offered by the business. Includes deal ID, name, URL, image URL, start and end times, popularity, restrictions, and additional details.
- Gift_certificates – Gift certificate information for the business if the business offers them. Includes gift certificate ID, URL, image URL, balance, price, and other information.
- Menu_provider – Provider of business menu.
- Menu_date_updated – Timestamp of last update.

4.3 Technical Constraints

The main constraint on the Yelp API is the limit of 25,000 API calls. However, this constraint is relatively flexible, as the Yelp API describes the process for appealing for a greater API call allotment. An increase in the volume of calls will not incur additional costs, making for greater flexibility in the LMA application design.

Additionally, since all requests must be authenticated with an OAuth token, LMA will need a centralized account with the Yelp API. The account will allow LMA to manage access to the Yelp API and is used by the API to keep track of the number of API calls.

In terms of constraints for the API call (GET) offered by the Yelp API, most of the search parameters that the API allows are optional, and are only used to pare down the search results. The one required parameter is the location of the client and/or the location that the client wishes to research. The location is passed into the search call either in the usual form as an address, city, state, and zip, or as a set of geographical coordinates. In either case, the location associated with the search query must be determined, whether automatically or manually inputted by the client.

Finally, in terms of mobile platforms, Yelp's iPhone application (yelp for iPhones >= 2.0.0 and yelp4 for iPhones >= 4.0.0) enables search, view, and check-in functions. Yelp does not have the same support for Android phones or other smart phones, which may influence the direction of the app in the future.

4.4 Summary

The Yelp API will be integrated into the LBS Module of LMA. LMA offers the user as its features the convenience of pulling up information about local businesses in order to make an informative consumer decisions. The Yelp API enables this LMA function by allowing users to search for local businesses or services and viewing the associated information that Yelp provides, such as the location, rating, and any deals that the business offers.

In addition to allowing users to proactively search for services to meet their needs, the Yelp API enables LMA to recommend services to the user based on LMA's profile of the

user. In this way, LMA extends the functionality of the Yelp API far beyond its current offerings. Since LMA also has information on the user's wish lists or to-do lists, LMA can make better recommendations for the user by matching the user's needs to local businesses nearby and calculating the optimal distribution of the user's patronage.

There are a few limitations to the Yelp API, namely the constrained volume of API calls and the need to collect information about the user's location. However, as described earlier, if the volume of API calls exceeds the initial quota, it is possible to appeal for a greater API call allowance. Additionally, since the LBS Module will need the user's location for many more applications, it will be relatively easy to gain access to such information.

5 API: Google Drive

5.1 Overview

Google Drive is a file storage and synchronization service based in the cloud that allows users to store, share, and edit files. Google Drive makes collaboration between users convenient by allowing modification of the same file by different users at the same time possible. It also makes files easily accessible no matter the location of the user. Among its many offerings, the Google Drive API offers the following functionality:

- Create and open files in the UI.
- Search for files.
- Distribute and market web applications.
- Share and collaborate on files by modifying the permissions of files.
- Export and convert Google docs.

The Google Drive API as it is integrated with LMA will provide users with cloud-based storage for their documents, pictures, and other files. It will also allow users to organize their files and bring them wherever they go.

The current version of the Google Drive API is Google Drive API v2. Google Drive API v2 is mostly compatible with v1 and Document List API v3, except for two changes: the parameter id from files.update and files.get renamed to fileId, and the parentsCollection field in files listing parents renamed to parents. However, Google Drive v2 supports migrating from these older versions. The API uses OAuth 2.0 protocol to authenticate a Google account and authorize access to user data.

5.2 Structure and API Calls

The Google Drive API is composed of many resource types, which subsequently are composed of various API calls.

5.2.1 Files

- **Get** – "Get /files/fileId"; gets a file's metadata by ID.
- **Insert** – "POST /files"; inserts a new file.
- **Patch** – "PATCH /files/fileId"; updates file metadata.
- **Update** – "Put /files/fileId"; updates file metadata and/or content.
- **Copy –** "POST /files/fileId/copy"; creates a copy of the specified file.
- **Delete** – "DELETE /files/fileId"; permanently deletes a file, skipping the trash.
- **List** – "GET /files"; lists user files.
- **Touch** – "POST /files/fileId/touch"; set's file's updated time to the current server time.
- **Trash** – "POST /files/fileId/trash"; moves file to trash.
- **Untrash** – "POST /files/fileId/untrash"; restores file from the trash.
- **Watch** – "POST /files/fileId/watch"; watches file for any changes.
- **EmptyTrash** – "DELETE /files/trash"; permanently clears user's trashed files.

5.2.2 About

- **Get** – "GET /about"; gets information about the current user and Drive settings.

5.2.3 Changes

- **Get** – "GET /changes/changeId"; gets specific change.
- **List** – "GET /changes"; lists user changes.
- **Watch** – "POST /changes/watch"; watches changes to user's Drive.

5.2.4 Children

- **Delete** – "DELETE /files/folderId/children/childId"; removes child from folder.
- **Get** – "GET /files/folderId/children/childId"; gets specific child reference.
- **Insert** – "POST /files/folderId/children"; inserts file into folder.
- **List** – "GET /files/folderId/children"; lists a folder's children.

5.2.5 Parents

- **Delete** – "DELETE /files/folderId/parents/parentId"; removes parent from folder.
- **Get** – "GET /files/folderId/parents/parentId"; gets specific parent reference.
- **Insert** – "POST /files/folderId/parents"; adds parents folder for a file.
- **List** – "GET /files/folderId/parents"; lists a folder's parents.

5.2.6 Permissions

- **Delete** – "DELETE /files/fileId/permissions/permissionId"; deletes a permission from the file.
- **Get** – "GET /files/fileId/permissions/permissionId"; gets a permission by file ID.
- **Insert** – "POST /files/fileId/permissions"; inserts a permission for the specified file.
- **List** – "GET /files/fileId/permissions"; lists a file's permissions.

5.3 Technical Constraints

The Google Drive API imposes some limits and quotas in order to protect the safety and integrity of the Google infrastructure. Among these limits are limits to the number of requests that an API project, which are subsequently divided into two categories. The Google Drive API sets the maximum number of requests per second (project QPS) to 5 QPS and the maximum number of requests per day (project QPD) to 150,000 QPD. When the limits are exceeded the server returns an HTTP 503 status code. The API's suggestion for handling these errors is to use an exponential backoff approach, in which each time an error code is returned, there is a longer wait before retrying the failed call. This ensures that the server will not be overloaded with requests. However, given the large volume of requests allowable by the API, it is unlikely that LMA will reach these limits, at least initially, so these errors should not be a cause for great concern at this points.

The Google Drive API also imposes a quota on the number of results that can be returned in an API's response. This quota, maxResults, has a range of 1 to 1000 events, with the default being 1000 records. Again, as with the API call limit imposed by the API, this quota should not majorly affect the functionality of LMA.

Additionally, since all requests must be authenticated with an OAuth token, LMA will need a centralized account with the Google Drive API. The account will allow LMA to manage access to the Google Drive API and is used by the API to keep track of the number of API calls. Without authentication, the request is considered unauthorized and will not be considered. Google Drive API uses OAuth 2.0 protocol.

Finally, the default data format is JSON but the API also supports Atom format. As JSON is a fairly common data format, these impositions should not drastically affect the path the LMA application will take, but they should be considered when operating the API.

5.4 Summary

The Google Drive API will be integrated with projected Cloud Module of LMA. Part of LMA's core functionality is providing user's easy accessibility to all the components of their lives. The Google Drive API enables this LMA function by allowing users to store and modify files on the go and have the same files synced across all of their devices.

In addition to the basic personal benefits of having all personal files sync across all of the user's devices, the Google Drive API also has many social and collaborative benefits. Chief among them is the ability of the user to work with other users on the same projects and files simultaneously. In this way, the Google Drive API facilitates collaboration between users and promotes networking and collective efforts toward common goals.

The Google Drive API interacts with several other functions of LMA. For example, user wish lists and todo lists are compiled by LMA for common usage by all of the services it

provides. The Google Drive API offers a way for LMA to sync these lists across all applications and services within itself.

There are a few limitations to the Google Drive API, namely the constrained volume of API calls. But, as described in the previous section, the limitations set by the Google Drive API are fairly generous and should not be too constraining on the current LMA model. The additional data format requirements are also general enough that other applications will also need to resolve them.

6 API: OpenTable

6.1 Overview

OpenTable is an online restaurant reservation service that allows users to make reservations for restaurants online, read restaurant reviews, and earn points from the restaurants toward free meals and other deals. The OpenTable API offers developers the following functionality:

- Access and search data from OpenTable.
- Find single or multiple restaurants, along with information about the restaurant.
- Find URLs where reservations can be made.

The OpenTable API as it is integrated with LMA will provide users with the convenience of finding restaurants in the surrounding area as well as to view reviews and other information on the restaurants. Additionally, the OpenTable API allows users to easily make reservations at the restaurants they designate. The OpenTable API shares some functionality with the Yelp API, but is more specific to restaurants and offers a few additional services.

The current version of the OpenTable API is open source and public. The API requires no authentication and does require the creation of any accounts with the API.

6.2 Structure and API Calls

The OpenTable API is limited to one type of API call, namely the GET request. The GET request allows users to access data about restaurants in the OpenTable database. OpenTable does not require authentication for API calls.

GET – Allows the client to search for local restaurants and access data about specific restaurants

- "GET /api/stats" – Returns response detailing the number of countries, cities, and restaurants that are available in the OpenTable database.
- "GET /api/cities" – Returns the number of cities in the database, as well as the list of cities that have restaurants in the OpenTable database.
- "GET /api/restaurants" – Finds the restaurant(s) that match the information specified in the parameters.
    - Name – Name of the restaurant.
    - Address – Address line (should not contain state or city or zip).
    - State – State code.
    - City – City name.
    - Zip – Zipcode.
    - Country – Country code.

  - Page – Page (default: 1).
  - Per_page – Entries per page (options: 5, 10, 15, 25, 50, 100; default: 25).

  Returns the number of restaurants that match the parameters provided, as well as the per_page and current_page settings. The call also returns a list of restaurants that match the parameters. Since all of the parameters are optional, if no parameters are specified, the call returns all the restaurants in the database.
- "GET /api/restaurants/:id" – Returns a single restaurant and its information. Each restaurant record has the following attributes:

  - Id – ID of the restaurant.
  - Name – Name of the restaurant.
  - Address – Address of the restaurant.
  - City – City location.
  - State – State location.
  - Area – General area of the restaurant.
  - Postal_code – Zip code of the location.
  - Country – Country location.
  - Phone – Phone number of the restaurant.
  - Reserve_url – Reservation link for the restaurant.
  - Mobile_reserve_url – Mobile reservation link for the restaurant.

6.3 Technical Constraints

The OpenTable API places some constraints on the number of API calls that can be made. It imposes a limit of 1000 requests per hour per IP Address. However, this limitation should not be too constraining to LMA at this point in time.

Additionally, the default data format for the OpenTable API is JSON. As JSON is a fairly common data format, these impositions should not drastically affect the path the LMA application will take, but they should be considered when operating the API. The API also supports JSONP.

6.4 Summary

The OpenTable API will be integrated with the LBS Module of LMA. The LBS Module's primary goal is to use the user's location to deduce information about the user in order to suggest services and make recommendations to the user. The OpenTable API works in a similar function as does the Yelp API in that both locate services for the user and offers comprehensive information about each service that matches the user's search parameters so that LMA can best help the user decide which services to patronize.

In addition to the basic functionality of searching that the Yelp API also provides, the OpenTable API allows users to access the reservation site of the restaurant for the

convenience of the user. The OpenTable API also keeps track of a user's points, which can then be used toward free meals or other deals. It also can bring up a restaurant's reviews to help users make informed decisions.

There are a few limitations to the OpenTable API, namely the constrained volume of API calls. But, as described in the previous section, the limitations set by the OpenTable API should not be too constraining on the current LMA model. The data format requirements are also general enough that other applications will also need to resolve them. Additionally, the OpenTable API does not require authentication or API tokens, so the process of bringing up functionality in the API may be less involved than it might be with other APIs.

7 API: SPOTIFY

7.1 OVERVIEW

Spotify offers users the convenience of accessing their music anytime no matter their location. It also allows users to listen to artists, albums, and playlists for free and categorize their music into playlists or folders. Spotify broadens the musical horizons of its users by making new music easily accessible through its various browse features. The Spotify API offers the following functionality:

- Fetch data from the Spotify music catalog.
- Manage user's playlists and saved music.
- Returns data about artists, albums, and tracks from the Spotify catalogue.
- Provides access to user-related data such as playlists and music saved in the "Your Music" library.

The Spotify API as it is integrated with LMA will provide users with easy accessibility to their music. It will also allow users to organize their music, find new music, and bring it with them wherever they go.

The API uses OAuth 2.0 protocol to authenticate an application and authorize access to make requests.

7.2 STRUCTURE AND API CALLS

The Spotify API is composed of many resource types, which subsequently are composed of various API calls.

7.2.1 Albums

- "GET /v1/albums/{id}" – Gets an album. The id refers to the Spotify ID for the album. The track object has the following attributes:
    - Artists – The artist who performed the track.
    - Available_markets – List of countries in which the track can be played.
    - Disc_number – Disc number (usually 1).
    - Duration_ms – Track length in milliseconds.
    - Explicit – Whether or not the track has explicit lyrics.
    - External_urls – External URLs for the track.
    - Href – Link to the full details of the track.
    - Id – Spotify ID for the track.
    - Name – Name of the track.
    - Preview_url – URL to a 30 second preview of the track.
    - Track_number – Number of the track.
    - Type – The object type (in this case, "track").
    - Uri – Spotify URI for the track.

- "GET /v1/albums?ids={ids}" – Gets several albums. The ids refer to a list of Spotify IDs for the albums, with a limit of 20 IDs. The album object has the following attributes.

    - Album_type – Type of the album (one of "album", "single", or "compilation").
    - Artists – Artists of the albums.
    - Available_markets – Markets in which the album is available. Yses ISO 3166-1 alpha-2 country codes. An album's availability in a market is determined by the availability of one or more of its tracks in the same market.
    - Copyrights – Copyright statements of the album.
    - External_ids – Known external IDs for the album.
    - External_urls = Known external URLs for this album.
    - Genres – List of genres used to classify the album. If the album is not classified, the list is empty.
    - Href – Link to the full details of the album.
    - Id – Spotify ID for the album.
    - Images – Cover art of the album in various sizes, with the widest first.
    - Name – Name of the album.
    - Popularity – Popularity of the album, between 0 and 100, with 100 being the most popular. The popularity is calculated from the individual popularities of the tracks.
    - Release_date – Date the album was released, to various degrees of precision.
    - Release_date_precision – Precision to which the release_date value is known.
    - Tracks – Tracks of the album.
    - Type – The object type, "album".
    - Uri – the Spotify URI for the album.

- "GET /v1/albums/{id}/tracks" – Gets the album's tracks. The id refers to the Spotify ID for the album. The request returns a list of track objects. The query parameters are the following:

    - Limit – Maximum number of tracks to return, with the default being 20, and the minimum being 1.
    - Offset – The index of the first track to return, with the default being 0 (the first object). Use with limit to get the next set of tracks.

  Both of the query parameters are optional.
- "GET /v1/artists/{id}/albums? – Gets an artist's albums.
- "GET /v1/browse/new-releases" – Gets a list of new releases.
- "GET /v1/search?type=album" – Searches for an album.

7.2.2 Artists

- "GET /v1/artists/{id}" – Gets an artist. The id refers to the Spotify ID for the artist. The API call returns an artist object. An artist object has the following attributes:

  - External_urls – Known external URLs for the artist.
  - Followers – Information about the followers of an artist.
  - Genres – List of genres that the artist is associated with. If the artist has not yet been classified, the list is empty.
  - Href – Link to the full details of the artist.
  - Id – Spotify ID for the artist.
  - Images – Images of the artist in various sizes, with the widest first.
  - Name – Name of the artist.
  - Popularity – Popularity of the artist, between 0 and 100, with 100 being the most popular. The artist's popularity is calculated from the popularity of the artist's tracks.
  - Type – The object type, "artist".
  - Uri – Spotify URI for the artist.

- "GET /v1/artists?ids={ids}" – Gets several artists. The ids refers to a list of Spotify IDs for the artists, with a maximum of 50 IDs. Returns an artist object, or null if the object is not found.
- "GET /v1/artists/{id}/albums" – Gets an artist's albums. The id refers to the Spotify ID for the artist. The query parameters are as follows:

  - Album_type – List of keywords that will be used to filter the response. If not supplied, all album types will be returned.
  - Market – An ISO 3166 alpha-2 country code that limits the response to a geographical market.
  - Limit – Number of album objects to return, with the default at 20, the minimum at 1, and the maximum at 50.
  - Offset – Index of the first album to return, with the default being 0 (the first album)

  .
- "GET /v1/artists/{id}/top-tracks" – Gets an artist's top tracks. The id refers to the Spotify ID for the artist. Returns track objects upon success. There is a single, required query parameter:

  - Country – An ISO 3166-1 alpha-2 country code.

- "GET /v1/artists/{id}/related-artists" – Gets an artist's related artists. The id refers to the Spotify ID for the artist. Upon success, responds with an array of up to 20 artist objects.
- "PUT /v1/me/following" – Follow Artists or Users.
- "DELETE /v1/me/following" – Unfollow Artists or Users.
- "GET /v1/me/following/contains" – Checks if Current User follows Users or Artists.
- "GET /v1/search?type=artist" – Searches for an artist.

7.2.3 Browse

- "GET /v1/browse/featured-playlists" – Gets a list of featured playlists. Returns a playlist object upon success. The query parameters are as follows:

    - Authorization – Required parameter. Valid access token from the Spotify Accounts service.
    - Country – An ISO 3166-2 alpha-2 country code.
    - Limit – Maximum number of items to return, with the default being 20, the minimum 1, and the maximum 50.
    - Offset – Index of the first item to return, with the default being 0 (the first object).

    The playlist attributes are as follows:

    - Collaborative – True if the owner allows other users to modify the playlist.
    - External_urls – Known external URLs for this playlist.
    - Href – Link to the full details of the playlist.
    - Id – Spotify ID for the playlist.
    - Images – Images for the playlist. Contains up to three images, in order of descending size.
    - Name – Name of the playlist.
    - Owner – User who owns the playlist.
    - Public – Playlist's public/private status (true if the playlist is public).
    - Tracks – Collection of tracks objects containing a link to the full details of the playlist's tracks, along with the total number of tracks in the playlist.
    - Type – Object type, "playlist".
    - Uri – Spotify URI for the artist.

    The paging object attributes are as follows:

    - Href – Link to the full result of the request.
    - Items – Requested data.
    - Limit – Maximum number of items in the response, as set by the query or by default.
    - Next – URL to the next page of items.
    - Offset – Offset of the items returned.
    - Previous – URL to the previous page of items.
    - Total – Total number of items available to return.

- "GET" /v1/browse/new-releases" – Gets a list of new releases. Returns an array of album objects upon success. The query parameters are as follows:

    - Authorization – Valid access token from the Spotify Accounts service.
    - Country – An ISO 3166-1 alpha-2 country code.

- Limit – Maximum number of items to return, with the default being 20, the minimum 1, and the maximum 50.
- Offset – Index of the first item to return, with the default being 0 (the first object).

7.2.4 Follow

- "PUT /v1/me/following" – Follows Artists or Users. The query parameters are as follows:

  - Authorization – Valid access token from the Spotify Accounts service. Modifying the list of artists or users the current users follows requires authorization of the user-follow-modify scope.
  - Content-Type – Required if IDs are passed in the request body (application/json).
  - Type – ID Type, either artist or user.
  - Ids – List of the artist or the user Spotify IDs.

- "DELETE /v1/me/following" – Unfollow Artists or Users. The query parameters are as follows:

  - Authorization – Valid access token from the Spotify Accounts service. Modifying the list of artists or users the current users follows requires authorization of the user-follow-modify scope.
  - Content-Type – Required if IDs are passed in the request body (application/json).
  - Type – ID Type, either artist or user.
  - Ids – List of the artist or the user Spotify IDs.

- "GET /v1/me/following/contains" – Checks if Current User Follows Users or Artists. The query parameters are as follows:

  - Authorization – Valid access token from the Spotify Accounts service. Getting details of artists or users the current users follows requires authorization of the user-follow-read scope.
  - Content-Type – Required if IDs are passed in the request body (application/json).
  - Type – ID Type, either artist or user.
  - Ids – List of the artist or the user Spotify IDs.

- "PUT /v1/users/{owner_id}/playlists/{playlist_id}/followers" – Follows a Playlist. Returns a Boolean on success, true if the playlist will be included in the user's public playlists, and false if it will remain private. The user must be granted the playlist-modify-private scope in order to follow playlists privately. The query parameters are as follows:

  - Owner-id – Spotify user ID of the person who owns the playlist.

- Playlist_id – Spotify ID of the playlist. Any playlist can be followed, regardless of its public or private status, as long as the playlist ID is provided.
- Authorization – Valid access token from the Spotify Accounts service. Following a playlist publicly requires authorization of the playlist-modify-public scope. Following the same playlist privately requires the playlist-modify-private scope.
- Content-Type – Content type of the request body (application/json).

- "DELETE /v1/users/{owner_id}/playlists/{playlist_id}/followers" – Unfollow a Playlist. The query parameters are as follows:

  - Owner-id – Spotify user ID of the person who owns the playlist.
  - Playlist_id – Spotify ID of the playlist.
  - Authorization – Valid access token from the Spotify Accounts service. Following a playlist publicly requires authorization of the playlist-modify-public scope. Following the same playlist privately requires the playlist-modify-private scope.
  - Content-Type – Content type of the request body (application/json).

- "GET /v1/users/{user_id}/playlists/{playlist_id}/followers/contains" Checks if Users Follow a Playlist. The request parameters are as follows:

  - Authorization – Valid access token from the Spotify Accounts service. Checking if a user publicly follows a playlist does not require any scopes. To check if a usr is privately following a playlist, the user must be granted access to the playlist-read-private scope.
  - Owner-id – Spotify user ID of the person who owns the playlist.
  - Playlist_id – Spotify ID of the playlist.
  - Ids – List of Spotify User IDs that you want to check to see if they follow the playlist, with the maximum being 5 ids.

7.2.5 Library

- "PUT /v1/me/tracks?ids={ids}" – Saves tracks for the user. A track can only be saved once, so duplicate IDs are ignored. The request parameters are as follows:

  - Authorization – Valid access token from the Spotify Accounts service. Modification of the current user's "Your Music" collection requires authorization of the user-library-modify scope.
  - Content-Type – Content type of the request body (application/json).
  - Ids – List of the Spotify IDs.

- "GET /v1/me/tracks" – Gets user's saved tracks and returns a list of track objects. The request parameters are as follows:

  - Valid access token from the Spotify Accounts service. The user-library-read scope must have been authorized by the user.

  - Limit – Maximum number of objects to return, with the default being 20, the minimum 1, and the maximum 50.
  - Offset – Index of the first object to return, with the default being 0 (the first object).

- "DELETE /v1/me/tracks?ids={ids}" – Removes user's saved tracks. The request parameters are as follows:

  - Valid access token from the Spotify Accounts service. Modification of the current user's "Your Music" collection requires authorization of the user-library-modify scope.
  - Content-Type – Content type of the request body (application/json).
  - Ids – List of Spotify IDs.

- "GET /v1/me/tracks/contains?ids={ids}" – Checks user's saved tracks. The request parameters are as follows:

  - Ids – List of Spotify IDs for the tracks, with the maximum at 50 IDs.
  - Authorization – Valid access token from the Spotify Accounts service. The user-library-read scope must have been authorized by the user.

7.2.6 Playlists

- "GET /v1/browse/featured-playlists" – Gets a list of featured playlists.
- "PUT /v1/users/{owner_id}/playlists/{playlist_id}/followers" – Follow a Playlist.
- "DELETE /v1/users/{owner_id}/playlists/{playlist_id}/followers" – Unfollow a Playlist.
- "GET /v1/search?type=playlist" – Searches for a playlist.
- "GET /v1/users/{user_id}/playlists" – Gets a list of the user's playlists. The request parameters are as follows:

  - User_id – User's Spotify ID.
  - Authorization – Valid access token from the Spotify Accounts service. Private paylists are only retrievable for the current user and requires the playlist-read-private scope to have been authorized by the user.
  - Limit – Maximum number of playlists to return, with the default being 20, the minimum 1, and the maximum 50.
  - Offset – Index of the first playlist to return, with the default being 0 (the first object).

- "GET /v1/users/{user_id}/playlists/{playlist_id}" – Gets a playlist. The request parameters are as follows:

  - User_id – User's Spotify ID.
  - Playlist_id – Spotify ID for the playlist.

    - Authorization – Valid access token from the Spotify Accounts service. Both Public and Private playlists belonging to any user are retrievable on provision of a valid access token.
    - Fields – Filters for the query. If omitted, all fields are returned.

- "GET /v1/users/{user_id}/playlists/{playlist_id}/tracks" – Gets a playlist's tracks. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Both Public and Private playlists belonging to any user are retrievable on provision of a valid access token.
    - Fields – Filters for the query. If omitted, all fields are returned.
    - Limit – Maximum number of tracks to return, with the default being 100, the minimum 1, and the maximum 100.
    - Offset – Index of the first track to return, with the default being 0 (the first track).

- "POST /v1/users/{user_id}/playlists" – Creates a playlist. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Authorization – Valid access token from the Spotify Accounts service. Creating a public playlist for a user requires authorization of the playlist-modify-public scope, while creating a private playlist requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).
    - Name – Name of the new playlist. The name does not have to be unique.
    - Public – Boolean value that is true if the playlist will be public, and false if it will be private.

- "PUT /v1/users/{user_id}/playlists/{playlist_id}" – Changes a playlist's details.
- "POST /v1/users/{user_id}/playlists/{playlist_id}/tracks" – Adds tracks to a playlist. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Adding tracks to the current user's public playlists requires authorization of the playlist-modify-public scope, while adding tracks to the current user's private playlist requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).
    - Uris – List of Spotify track URIs to add.

    - Position – Position to insert the tracks. Tracks are added in the order they are listed in the query string or request body.

- "DELETE /v1/users/{user_id}/playlists/{playlist_id}/tracks" – Removes tracks from a playlist. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Removing tracks from the current user's public playlists requires authorization of the playlist-modify-public scope, while removing tracks from the current user's private playlist requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).
    - Tracks – Spotify URIs of the tracks to remove and their positions in the playlist.

- "PUT /v1/users/{user_id}/playlists/{playlist_id}/tracks" – Replaces a playlist's tracks. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Setting tracks in the current user's public playlists requires authorization of the playlist-modify-public scope, while setting tracks in the current user's private playlist requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).
    - Uris – List of Spotify track URIs to set.

- "GET /v1/users/{user_id}/playlists/{playlist_id}/followers/contains" – Checks if Users Follow a Playlist.

7.3 Technical Constraints

7.4 Summary

8 API: DROPBOX

8.1 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.

8.2 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.

8.2.1 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.

8.2.2 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* – *B*egins the process for linking new accounts.
- (DBAccount *)handleOpenURL:(NSURL *)*url* – *C*all 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 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 *)*fields*inserted:(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` and`NSData – L`ength 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

8.3 Technical Constraints

8.4 Summary