Introduction

Google provides documentation for anyone who wants to understand Google's Data protocols for reading and writing data on the internet. Google Data organizes information and makes it accessible in contexts other than a web browser and accessible to services outside of Google.

Google Data provides a secure means for developers to write new applications that let end users access and update the data stored by many Google products. Developers can use the Google Data Protocol directly, or any of the supported programming languages provided by the client libraries.

Google Client Applications

Google Apps is a service that provides customizable versions of several products under the Google domain name. Google Apps feature several Web applications with functionality similar to traditional office suite software.

Google Apps includes the following:

Google Contacts
Google Contacts is a contact management tool available in Gmail, or as a standalone service.
Google Calendar
Google Calendar (Gcal) is a free time-management web application. Users are required to have a Google Account in order to use this application.
Google Docs
Google Docs is a free Web-based office suite and data storage service. It allows users to create and edit documents and spreadsheets online while collaborating in real-time with other users.
YouTube
YouTube is a video-sharing website where users can upload, share, and view videos.
Picasa
Picasa is an image organizer and viewer for organizing and editing digital photos, which also features an integrated photo-sharing website under the same name.
Google+
Provides social networking and messaging.
Google Books
Google Books searches the full text of books that Google has scanned, converted to text using optical character recognition, and stored in its digital database.
Google Moderator
Google Moderator uses crowdsourcing to rank user-submitted questions, suggestions, and ideas.
Google Cloud
Google Cloud Connect is a free cloud computing plug-in for Windows Microsoft Office that can automatically store and synchronize any Microsoft Word document, PowerPoint presentation, or Excel spreadsheet to Google Docs in Microsoft Office or Google Docs formats.
Google Big-Query
Google BigQuery for Developers is a RESTful web service that enables interactive analysis of massively large datasets working in conjunction with Google Storage. It is an Infrastructure as a Service (IaaS) that may be used to complement MapReduce.
Google Health
Google Health is a personal health information centralization service. The service allows users to store their health records with Google and access them later.
Google Translate
Google Translate is a free statistical machine translation service that translates text, documents,

and web pages.

Google Client Accessibility

The Google API client library is designed to ensure accessibility to the above applications from different platforms. It facilitates building collaborative use-cases, and builds custom and platform-specific applications.

Feasible Use Cases

Feasible use cases of Google API client include the following types of applications:

- •Translation
- •Book search
- •Multiplayer Games
- •Compute intensive applications
- •Health related applications

Google API Client Library

Google API client libraries are available to help write client applications that access a chosen API. For each language, the client library provides tools and an abstraction layer, letting you construct queries and use response data without having to create HTTP requests or process HTTP responses by hand. Each client library provides classes that correspond to the elements and data types that the API uses.

The Google API client library supports the **ATOM + XML Json** data model. JacksonFactory, GsonFactory classes for the Json model with AtomParser and XmlHttpParser classes for Atom + XML.

The usage of data models is application specific. Latitude, Perdition and Moderator use the Json data model; while the Calendar, Document List, and Picasa all use the ATOM+ XML data model.

Google API Java Client

The Java client library is flexible, efficient, and powerful for accessing HTTP-based APIs on the web. It features a powerful **OAuth 2.0** and **OAuth 1.0a** library with a consistent interface.
Its **XML** and **JSON** data models are lightweight, efficient, and support any given data schema. This diagram illustrates the Google API Java client:

Figure 1: Google API Client

Authentication and Authorization Supported

Third-party applications often require limited access to a user's Google Account for certain types of activity. To ensure that user data is not abused, all requests for access must be approved by the account holder. Access control has two components: authentication and authorization.

Authentication services allow users to sign into an application using their Google Account. Some services also allow users to sign in using another account, such as an **OpenID** login.

Authorization services let users provide applications with access to the data stored in Google Apps.

Google takes privacy seriously, and any application that requires access to a user's data must be authorized by the user.

Authentication and authorization services are often referred to collectively as **auth**.

OAuth 1.0

Web applications that need authorized access to data associated with a Google Account or a Google Apps account can use Google's implemented **OAuth API** for accessing Google services.

- **OAuth 1.0** refers to the Resource owner (End User), Client (Our App), and Resource Store (Google server).
- The client authenticates with the server and handout Access token.
- Standardizes the process of making authenticated HTTP requests with 2 sets of credentials, identifying the Resource owner and identifying the Client.

ClientLogin

Google's **ClientLogin** is a programmatic method of getting authorized access to information stored with Google's services by a user. Authorization with **ClientLogin** involves a sequence of interactions between three entities:

- Installed Application
- Google services
- User

This following diagram shows the client-login flow:

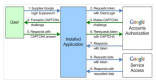

Figure2:Client login

1.When the third-party application needs to access a user's Google service, it retrieves the user's login name and password.

2.The third-party application then makes a **ClientLogin** call to Google's Authorization service.
3.If the Google Authorization service decides additional vetting is necessary, it returns failure response with a CAPTCHA token and challenge, in the form of a URL for a CAPTCHA image.

4.If a CAPTCHA challenge is received, the third-party application displays the CAPTCHA image for the user and solicits an answer from the user.

5.If requested, the user submits an answer to the CAPTCHA challenge.

6.The third-party application makes a new ClientLogin call, this time including the CAPTCHA answer and token (received with the failure response).

7.On a successful login attempt (with or without CAPTCHA challenge), the Google Authorization service returns a token to the application.

8.The application contacts the Google service with a request for data access, referencing the token received from the Google Authorization service.

9.If the Google service recognizes the token, it supplies the requested data access.

OAuth 2.0

OAuth 2.0 is the next version of the OAuth protocol and is not backward compatible with **OAuth**

1.0. OAuth 2.0focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.

Some points on OAuth:

- **OAuth 2.0** allows finer control on access, Read-Write Access, Full Access unlike existing blanket access to usage domain
- Authentication tokens expire within 1 hour
- So the application will need to keep checking for 401 responses to re-authorize

Setting up Google Data API Client

To setting up the Google API Client we require Maven plug-in & Mercurial code repository. The Mercurial contains the whole project history while Maven plug-in are nothing but a build automation tool typically used for Java projects, its similar purpose to an Apache Ant tool, but it is based on different concepts and works in a profoundly different manner under Google API Client.

The following are the steps required to downloading & setting up the Google Data API Client with Maven plug-in and Mercurial repository.

1.Install the **Maven plug-in** for Eclipse.
This is required for direct creation of the Google API Client JAR files on local PC.
2.Install **Mercurial.**
To download a sample: https://samples.google-api-java-client.googlecode.com/hg/
3.Import Project.
File ? New? Android Project ? Create Project from Existing source
4.Please DO NOT try to start fixing compilation errors and adding .jar files yourself from the setup page.

5.Let the Maven plugin resolve dependencies by downloading the .jar files and associated source jars.

6.Check pom.xml and Setup instructions if the Maven plugin does not start downloading the .jar files.

Google Data API Client in Android

To use Google services in Android using Google Data API, do the following:

- Use the Android Account Manager to authorize access to the User’s data for the application (Client).
- Use the Google API HttpRequest and HTTPResponse classes to create requests.
- Use the Google API HTTPTransport and HTTPRequestFactory classes to build and transmit HTTP requests .
- Parse responses using AtomParser or JsonCParser and get the data feed.
- Configure your parser as per DICTIONARY applicable to the relevant data model and use case.

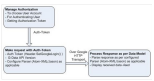

Figure3:Google Data API Client in Android

Implementation of Authentication

To get access to the centralized registry of the user's online accounts, enter the account credentials (username & password). Different online services have different ways of handling accounts and authentication. In Android, the account manager uses pluggable Authenticator modules for different account types. The Authenticators (which may be written by third parties) handle the actual details of validating account credentials and storing account information. For example, Google, Facebook, and Microsoft Exchange each have their own authenticator.
The Account Manager can generate an auth token for application, so the application doesn't need to handle passwords.
The account type for Google services is always "com.google", while an auth token is specific to the app, such as "cl" for Google calendar.

Figure4:Account Authenticator

Figure5:Authenticator Service

Figure6:Account Authenticator

Android Account Manager Code Snippet

To Access the Google account we need to set the Auth token of user by client login, it uses the stander security measures to protect the user account information. Client login can be used with any application which use HTTPPOST request with the mentioned URL,
URL Parameter:https://www.google.com/accounts/ClientLogin
Post request should be structured as a form where we need to include the post request parameters, included parameters into the POST body are as follows,

SR.No.	Parameter	Description.
1.	accountType	Type of account to request authorization for. Possible values are: GOOGLE (get authorization for a Google account only) HOSTED (get authorization for a hosted account only) HOSTED_OR_GOOGLE (get authorization

SR.No.	Parameter	Description.
		first for a hosted account; if attempt fails, get authorization for a Google account) Use HOSTED_OR_GOOGLE if you're not sure which type of account you want authorization for. If the user information matches both a hosted and a Google account, only the hosted account is authorized.
2.	Email	User's full email address. It must include the domain (i.e. johndoe@gmail.com).
3.	Passwd	User's password.
4.	service	Name of the Google service you're requesting authorization for. Each service using the Authorization service is assigned a name value; for example, the name associated with Google Calendar is 'cl'. This parameter is required when accessing services based on Google Data APIs. For specific service names, refer to the service documentation.
5.	source	Short string identifying your application, for logging purposes. This string should take the form: "companyName-applicationName-versionID".
6.	logintoken	(optional) Token representing the specific CAPTCHA challenge. Google supplies this token and

SR.No.	Parameter	Description.
		the CAPTCHA image URL in a login failed response with the error code "CaptchaRequired".
7.	logincaptcha	(optional) String entered by the user as an answer to a CAPTCHA challenge.

HTTPPOST Sample Request:

```
POST /accounts/ClientLogin HTTP/1.0
Content-type: application/x-www-form-urlencoded

accountType=HOSTED_OR_GOOGLE&Email;=jondoe@gmail.com&Passwd;=north23AZ&service;=cl&
source=Gulp-CalGulp-1.05
~~~~~~~~~~~~~~
POST /accounts/ClientLogin HTTP/1.0
Content-type: application/x-www-form-urlencoded
source=Gulp-CalGulp-1.05
accountType=HOSTED_OR_GOOGLE&Email;=jondoe@gmail.com&Passwd;=north23AZ&service;=cl&
&logintoken=DQAAAGgA...dkI1LK9&logincaptcha=brinmar
```

In response to a login request, Google returns either an HTTP 200, if login succeeded, or an HTTP 403, if login failed. A success response contains the authorization token, labeled "Auth", in the body of the response

HTTP Sample Response:

```
HTTP/1.0 200 OK
Server: GFE/1.3
Content-Type: text/plain
SID=DQAAAGgA...7Zg8CTN
LSID=DQAAAGsA...lk8BBbG
Auth=DQAAAGgA...dk3fA5N
 ~~~~~~~~~
HTTP/1.0 403 Access Forbidden
Server: GFE/1.3
Content-Type: text/plain
Url=http://www.google.com/login/captcha
Error=CaptchaRequired
CaptchaToken=DQAAAGgA...dkI1LK9
CaptchaUrl=Captcha?ctoken=HiteT4b0Bk5Xg18_AcVoP6-yFkHPibe7O9EqxeiI7lUSN
```

Once we get the Auth token then the following code snippet returns the number of Google Accounts on device,

```
...
        Account[] accounts = manager.getAccountsByType("com.google“);
...
```

The following code snippet invokes the Account Manager to access a Google account:

```
...

        GoogleAccountManager accountManager = new GoogleAccountManager(this);

        Account account = accountManager.getAccountByName(accountName);

        if (account != null) {
                // handle invalid token
                Bundle bundle =manager.getAuthToken(account, "", true, null,
null).getResult();

                String authToken = bundle.getString(AccountManager.KEY_AUTHTOKEN));

          if (authToken == null) {
                accountManager.manager.getAuthToken( account, AUTH_TOKEN_TYPE,true,new
AccountManagerCallback
                <bundle></bundle>() {

                        public void run(AccountManagerFuture<bundle></bundle> future) {

                        try {

                                Bundle bundle = future.getResult();
                                        if
(bundle.containsKey(AccountManager.KEY_INTENT)){
                                        // Come here when  User needs to provide
Authorization
                                        Intent intent =
bundle.getParcelable(AccountManager.KEY_INTENT);

                                        int flags = intent.getFlags();

                                        flags &= ~Intent.FLAG_ACTIVITY_NEW_TASK;

                                        intent.setFlags(flags);

                                        startActivityForResult(intent,
REQUEST_AUTHENTICATE);
```

```
                                    } else if
(bundle.containsKey(AccountManager.KEY_AUTHTOKEN)){
                                            // Come here when  we receive Auth token


setAuthToken(bundle.getString(AccountManager.KEY_AUTHTOKEN));

                                            //executeRefreshCalendars();

                                            executeRefreshDocuments();
                                    }
                                    catch (Exception e) {
                                                    handleException(e);
                                                    }
                            }
                }, null);

...
```

The user authenticates themselves by entering their password into a pop-up:

Figure 7: Google Sign in

The user authorizes the Client Application:

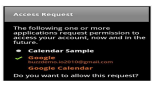

Figure 8:Sign in Authorize

Retrieving a Document List by Using Document List APIs

The following code snippet retrieves the document list "Request Initialization":

```
...

        GoogleHeaders headers = new GoogleHeaders();

        headers.setApplicationName("Google-DocsSample/1.0");

        headers.gdataVersion = "3";

        // 3 for Document List
```

```
        headers.setGoogleLogin(authToken);

        request.headers = headers;

        AtomParser parser = new AtomParser();

        parser.namespaceDictionary = DOC_DICTIONARY;

        request.addParser(parser);

        request.url.set("gsessionid", gsessionid);

...
```

The code snippet below shows the actual document downloaded list:

```
...

DocsUrl url = DocsUrl.forDefaultPrivateFull();
                while (true) {

                DocumentListFeed feed = client.executeGetDocListFeed(url);
                if (feed.docs != null) {
                        documents.addAll(feed.docs);
                }
                String nextLink = feed.getNextLink();
                if (nextLink == null) {
                        break;
                }
        }
int numDocs = documents.size();

...
```

Response parsing code snippet:

```
...

        HttpRequest request = requestFactory.buildGetRequest(url);
        HttpResponse uResponse = request.execute();
        uResponse.parseAs( feedClass);

...
```

ProGuard

The ProGuard tool shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names. The result is a smaller sized

.apk file that is more difficult to reverse engineer. Because ProGuard makes your application harder to reverse engineer, it is important that you use it when your application utilizes security-sensitive features and when you are licensing applications.

Pros

- Integrated in SDK v 2.3
- Works only in release mode
- Obfuscates the classes, fields and methods
- Removes un-used classes, attributes and methods
- Lowers APK size by 95%
- Dumps mapping of original and obfuscated classes (mapping.txt)

Cons

- Sometimes over-optimizes, functionality requires thorough testing for release mode
- Stack traces in case of exception or fault are unusable

Precautions

- Keep a copy of "mapping.txt" in order to make sense out of obfuscated stack trace.
- The **Retrace.bat** utility helps in building meaningful stack traces and logs

Setup

The typical Google Data API project has around 7 to 8 External .jar files. For more details about ProGuard, seecalendar-v2-atom-android-sample.

Setup in Detail

- **Proguard.cfg** is created automatically in the root directory of project
- ProGuard is already included by default on Android 2.3 SDK and you only need enable proguard.config=proguard.cfg
- The proguard.cfg file now has to be appended with the following lines (When used along with Google Data API applications):

```
...

        #Needed by the google-api-client to keep generic types and @Key annotations
accessed via reflection

        -keepclassmembers class * {

                @com.google.api.client.util.Key <fields></fields>;

        }

        -keepattributes Signature,RuntimeVisibleAnnotations,AnnotationDefault

        # Needed by Guava library

        #It contains Google's core libraries that we rely on in our Java-
```

```
        #based projects: collections, caching, primitives support,
        #concurrency libraries, common annotations, string processing, I/O,
        #etc
        -dontwarn sun.misc.Unsafe
        # See https://groups.google.com/forum/#!topic/guava-discuss/YCZzeCiIVoI
        -dontwarn com.google.common.collect.MinMaxPriorityQueue
...
```

ref:http://developer.samsung.com/android/technical-docs/Google-API-Client