Matthew Beardmore

Matthew Bowen

Topics in Languages and Tools - Project 2

Google Contacts API Interface

Google Contacts is a contact database that is available for any Google account. It can be used to store information about a user's contacts, including their email and phone number, and allows it to be transferred to any device that you use. In our project, we have created a bash script using other command line tools including sed, grep, and awk to be able to access this API and display and modify a user's contacts. We give the user a code and a URL that they will enter to log into their account and then have them authorize our script to access their contacts. We will then manage the user's contacts by giving the user a few options, including displaying all contacts that they have, searching for a specific contact, and adding/deleting contacts from their contact list.

At some point, while working on whatever it is they may need to accomplish, a user may wish to call their friend or send an email to a coworker, but they do not remember the contact information for their intended recipient. Navigating all the way to their Google account and into their contacts database can be a chore, especially if the user is working through an SSH connection or on a console terminal. What then is a user to do? Our script sets out to solve that problem by providing a simple and clear command line interface to a user's contacts database via their Google account.

An issue arises when considering how to log into a user's Google account. Google does not support asking the user for their username and password directly, due to security concerns, and since our script is on the command line only, how does one properly authenticate with Google? The answer is OAuth2.0: OAuth 2.0 is used because it allows an application access to a user's account without that application ever needing to come in contact with the user's login credentials, which enhances security and increases a user's confidence in the application. The OAuth 2.0 workflow starts with the application contacting Google for a device code, a user code, and a verification URL. The device code is a unique string that identifies the authentication session to Google. The user code and verification URL are given to the user by the application and are used by the user to associate their Google account with that authentication session; the user navigates to the verification URL, logs into their Google account, and enters the user code into the web page. Google then gives the user information about the application itself as well as what data the application will have access to should they allow it access to their account. Once the application acquires a device code and gives the user code and the verification URL to the user, it begins waiting for the user to complete the authentication process. To determine whether or not the user has authenticated completely, the application polls Google's servers every five seconds for an authentication status. Once the user authenticates with Google and allows the application access to their account, Google responds with an access token that is used for all subsequent API calls.

To begin using a Google API, a developer must first use a Google account to create a Google API project on the Google Developer Console (https://console.developers.google.com/). Once a project is created, that developer must then enable the APIs they want to access in their project management page; in our case, we enabled the Contacts API. Finally, a developer must then create a client ID to distribute with their Google API-consuming application. A client ID identifies to Google the application that is using the API for tracking and billing purposes.

Once you have a Google API project and a client ID, to access a user's account information, you must authenticate your project with that user's Google account. Our script does this when it first starts; it first checks to see if the user has authenticated with Google in the past by looking for a cached access token saved to a file. If a cached access token is not found, the script then asks Google for a device code, a user code, and a verification URL by making a POST request via wget containing our client ID and a "scope" that allows us to access the Contacts API. Google responds with the aforementioned information in a JSON response. The script parses that JSON response using grep and awk to extract the necessary bits of information. Once the script has everything it needs, it then prints the user code and verification URL to the screen and tells the user to navigate to that URL and enter the user code when prompted. Then, it enters a loop, sending a POST request to Google's servers every five seconds to see if the user has authenticated the request yet. Once authentication is complete, Google responds with an access token that the script uses with every subsequent API call.

Our script can perform four basic tasks: listing a user's contacts, searching for contacts, adding new contacts, and deleting contacts. All of these use Google's Contacts API via wget. All of these use the same or very similar HTTP request URLs with varying HTTP verbs.

To list the user's contacts, a request is made to the appropriate HTTP URL; included is our access token obtained during the authorization phase. Google responds to the request with an XML snippet that contains information about all of the user's contacts. To parse this XML response, we used awk to process it line by line. For each contact, our awk script looks for an opening <entry> XML tag, which marks the beginning of a contact's information. Inside that <entry> tag are a few pertinent XML tags containing information we want, including <title> for the contact's name, <gd:email> for the contact's email, and <gd:phoneNumber> for the contact's phone. There is also an HTTP URL associated with every contact that is used later on by the script for deleting contacts. We use awk and regex to extract the information out of each XML tag. However, it is not guaranteed or even expected for every contact to have all four pieces of information. As such, once we find a closing </entry> tag, we check to see that we at least found the contact's name. If we didn't, we ignore the contact if we did, then we output all the information we found while scanning the contact's entry into a tab-separated list of values. We continue in this fashion until the entire XML snippet is processed. Once we have a nicely-formatted tab-separated list containing information about the user's contacts, we then parse that list into a formatted table to display to the user.

To search the user’s contacts, one uses the same URL for getting the full list of user’s contacts, but that URL also includes a parameter for specifying a query. That query is used by Google to search the user’s contacts and returns a filtered set of contacts. Parsing and displaying that filtered set is exactly the same as simply listing all of the user’s contacts.

To add a new contact, Google expects an XML snippet from the application containing information about the new contact. To construct this snippet, our application uses a template with placeholders for the new contact’s name, phone number, and email. When the user wants to add a new contact, the script prompts the user for the required information (the user, however, does not need to provide all three pieces of information; some can be left blank). Then, the script replaces the aforementioned placeholders with the information the user provides, or deletes the XML tag containing the placeholder if no information was provided. The script then sends the XML snippet to Google through an HTTP CREATE request; if successful, Google responds with an HTTP success status code, and the contact is added to the user’s database.

Deleting contacts merely requires that the script makes an HTTP DELETE request to the appropriate HTTP URL. Each contact has a unique URL associated with it that is required if an application wishes to edit or delete it; that URL is included in each contact’s entry when requesting information about the user’s contacts. When the user specifies that they wish to delete a contact, the script shows the list of the user’s contacts to them and asks them to select which one they want to delete. When the user selects a contact, the script then confirms that the contact the user selected is the one they actually meant. Once confirmation is received, the script sends an HTTP DELETE request to the unique URL associated with that contact; the contact is then deleted from the user’s database immediately.

Our project solves the problem we sought to solve well; it offers a clean and clear interface to a user’s contacts database, and allows them to view their contacts quickly and in a well-formatted interface. The tools we use in this script, including bash, grep, sed, awk, and wget, all work well together and allow us to create this interface in a minimal amount of code. It also allows us to access the user’s contacts database without asking them directly for their password, maintaining a safe and secure environment and protecting the user’s Google account.

If we were to extend this project in the future, we could extend it to be able to set more properties for the contacts. We could add addresses, birthdays, associated URLs, notes, and the ability to upload pictures for contacts. We could also add a better system for displaying and managing all of these new properties. There is also some integration with Google+, in which you can add contacts to Google+ “circles”; we could utilize that Google+ API to be able to add contacts to your circles as well. This would require asking for the Google+ scope in order to gain access to the new API and then using the JSON based API to access information about the groups and how to put users into those groups. We can then request information about the groups that exist, along with adding, editing, and removing these groups. We can use this to group contacts together for the user and print these groups out as well. All of these features could make our project be a much more capable program that could fully manage a user’s contact list.