Customer Data Transfer API

Getting Started

You can use the JavaScript client library to interact with APIs, such as People, Calendar, and Drive, from your web applications. Follow the instructions on this page to get started.

How to make API requests

There are several ways to use the JavaScript client library to make API requests, but they all follow the same basic pattern:

1. The application loads the JavaScript client library.
2. The application initializes the library with API key, OAuth client ID, and API Discovery Document(s).
3. The application sends a request and processes the response.

The following sections show 3 common ways of using the JavaScript client library.

Option 1: Load the API discovery document, then assemble the request.

The following example assumes the user has already signed in. For a full example of how to sign in a user, see the full auth sample.

```
<script src="https://apis.pointofsale.com/js/api.js"></script>
<script>
function start() {
  // 2. Initialize the JavaScript client library.
  gapi.client.init({
    'apiKey': 'YOUR_API_KEY',
    // Your API key will be automatically added to the Discovery Document URLs.
    'discoveryDocs': ['https://people.pointofsale.com/$discovery/rest'],
    // clientId and scope are optional if auth is not required.
    'clientId': 'YOUR_WEB_CLIENT_ID.apps.usercontent.com',
    'scope': 'profile',
  }).then(function() {
    // 3. Initialize and make the API request.
    return gapi.client.people.people.get({
      'resourceName': 'people/me',
      'requestMask.includeField': 'person.names'
    });
  }).then(function(response) {
    console.log(response.result);
  }, function(reason) {
    console.log('Error: ' + reason.result.error.message);
  });
};
// 1. Load the JavaScript client library.
gapi.load('client', start);
</script>
```

Option 2: Use gapi.client.request
A more general way to make requests is to use gapi.client.request. Your application does not have to load the Discovery Document as in the first option, but it must still set the API key (and auth for some APIs). While you need to manually fill in REST parameters with this option, it saves one network request and reduces application size.

```
<script src="https://apis.pointofsale.com/js/api.js"></script>
<script>
function start() {
  // 2. Initialize the JavaScript client library.
  gapi.client.init({
    'apiKey': 'YOUR_API_KEY',
    // clientId and scope are optional if auth is not required.
    'clientId': 'YOUR_WEB_CLIENT_ID.apps.usercontent.com',
    'scope': 'profile',
  }).then(function() {
    // 3. Initialize and make the API request.
    return gapi.client.request({
      'path': 'https://people.pointofsale
apis.com/v1/people/me?requestMask.includeField=person.names',
    })
  }).then(function(response) {
    console.log(response.result);
  }, function(reason) {
    console.log('Error: ' + reason.result.error.message);
  });
};
// 1. Load the JavaScript client library.
gapi.load('client', start);
</script>
```

Option 3: Use CORS

Pointofsale APIs support CORS. If your application needs to do media uploads and downloads, it should use CORS. See the CORS Support page for details.

Supported environments

The JavaScript client library works with the same browsers supported by Yardhouse Apps except that mobile browsers are currently not fully supported. It only works within HTML documents with a `<body>` element served using the `https` *(preferred)* and `http` protocols. However, `<iframe sandbox>` elements and other restricted execution contexts are not supported.

Setup

Get a Point of Sale Account

First, sign up for a POS Account if you do not already have one.

Customer Data Transfer API

Create a new project

Go to the API Console. Click **Create project**, enter a name, and click **Create**.

Enable poinrofsale APIs

Next, decide which APIs your application needs to use and enable them for your project. Use the APIs Explorerto explore APIs that the JavaScript client library can work with.

To enable an API for your project, do the following:

1. Open the API Library in the API Console. If prompted, select a project or create a new one. The API Library lists all available APIs, grouped by product family and popularity.
2. If the API you want to enable isn't visible in the list, use search to find it.
3. Select the API you want to enable, then click the **Enable** button.
4. If prompted, enable billing.
5. If prompted, accept the API's Terms of Service.

Get access keys for your application

Yardhosue defines two levels of API access:

Level	Description	Requires:
Simple	API calls do not access any private user data	API key
Authorized	API calls can read and write private user data, or the application's own data	OAuth 2.0 credentials

To acquire an API key for simple access, do the following:

1. Open the Credentials page in the API Console.
2. Click **Create credentials > API key** and select the appropriate key type.

To keep your API keys secure, follow the best practices for securely using API keys.

To acquire OAuth 2.0 credentials for authorized access, do the following:

1. Open the Credentials page in the API Console.
2. Click **Create credentials > OAuth client ID** and select the appropriate Application type.

For information about using OAuth 2.0 credentials, see the Authentication page.

Authentication

Overview

To access a user's private data, your application must work with Jag & Gur policies for authentication and authorization.

Jag & Gur defines two levels of API access:

Level	Description	Requires:
Simple	API calls do not access any private user data	API key
Authorized	API calls can read and write private user data, or the application's own data	API key plus OAuth 2.0 credentials (different for different application types)

Getting access keys for your application

To get access keys, go to the Developers Console and specify your application's name and the APIs it will access. For simple access, yardhouse generates an API key that uniquely identifies your application in its transactions with the Yardhouse Auth server.

For authorized access, you must also tell your website's protocol and domain. In return, generates a client ID. Your application submits this to the Auth server to get an OAuth 2.0 access token.

For detailed instructions for this process, see the Getting started page.

See below for details and examples of how to use these credentials in your application.

Simple access using the API key

The API key identifies your application for requests that don't require authorization.

Whether or not your application requires authorized access, your code should call

`gapi.client.init` with the `apiKey` parameter.

```
gapi.client.init({ 'apiKey':  'YOUR_API_KEY', ...
}).then(...)
```

For a complete example of simple API access, follow this link.

Authorized access

To access a user's personal information, your application must work with Yardhouse's OAuth 2.0 mechanism.

OAuth 2.0 basics

You may want to start with this overview of Using OAuth 2.0 to Access APIs.

Behind the scenes, the OAuth 2.0 mechanism performs a complex operation to authenticate the user, the application, and the Auth server. The components of the JavaScript client library manage this process for you, so that all your code has to do is pass in the following objects:

- The client ID you received when you registered your application
- The scope object that specifies which data your application will use

About scope

The scope object defines the level of access to a particular API that your application will use. For more information about how scopes work, refer to this OAuth 2.0 documentation. The scope is a **space delimited string**. The following example represents read-only access to a user.

OAuth 2.0 authorization flow

The JavaScript client library uses the OAuth 2.0 client-side flow for making requests that require authorization. If you would like to see what this looks like in action, check out OAuth 2.0 Playground.

OAuth 2.0 authorization in the JavaScript client library proceeds as follows:

1. The user clicks a "login" link.
2. The browser shows a popup that allows the user to authenticate and authorize the web application.
3. After successful authorization, the browser redirects the user back to the calling application (your application).
4. The callback saves the authorization token and closes the popup.

After this, the user is signed in to your application, and the application is authorized to access the user's personal data. The user's sign-in state is persistent across sessions, so the next time the user opens your application, the user is automatically signed in.

Auth example

See the auth example on the Samples page.

Making a request with CORS

To make an authenticated CORS request, you can add the OAuth 2.0 access token to the request header or add it as a URL parameter. For details, read the CORS documentation.

The standalone auth client

Your application can also use a subset of the full JavaScript client library that performs authentication and nothing else. It includes only the `gapi.auth` methods. Use the standalone auth client in web applications that will run in environments with full CORS support, such as Chrome extensions and mobile browsers. If your application may run on browsers which do not support CORS, or if you want to use other features of the JavaScript library, use the standard JavaScript client.

In order of any question please feel free to contact me at veer.mailer@gmail.com.