



Creating a store locator: Best practices

About this document

The information and recommendations in this document were gathered through our work with a variety of clients and environments in the field. We thank our customers and partners for sharing their experiences and insights.

What's covered	Best practices for creating a store locator application, using Google Maps Engine and the Google Maps API for Business.
Primary audience	Technical expertise required. Must have experience developing applications using the Google Maps API for Business.
Key takeaways	• Use Google Maps Engine for data storage and the Google Maps API for Business to implement you store locator features. • If you've already created a stores locator using Google Fusion Tables, you can migrate your data easily to Google Maps Engine. • The basic steps to creating a store locator are 1) geocode store locations, 2) upload stores data to Maps Engine, 3) implement a mechanism to geocode the searched area, 4) create queries to search for and display stores, and 5) add any other features you want your store to include
Last updated	06 June 2014 - See Release Notes
Feedback	Google values your feedback. Send your comments about this guide to enterprise-assets-feedback@google.com .

Third-party products: This document describes how Google products work with third-party products and the configurations that Google recommends. Google does not provide technical support for configuring third-party products. GOOGLE ACCEPTS NO RESPONSIBILITY FOR THIRD-PARTY PRODUCTS. Please consult the product's Web site for the latest configuration and support information. You may also contact Google Solutions Providers for consulting services.

Introduction

This document describes best practices for creating and optimizing an application that your customers can use to locate your stores or other types of locations on your web site. It covers the use of Google Maps Engine for data storage and the Google Maps for Business APIs for implementing the mapping functions. If you've already created a store locator using Google Fusion Tables, you'll also find your migration options in this document.

Before you begin

You'll need the following before you get started creating your store locator:

- Experience developing applications using the [Google Maps API for Business](#).
- A Google Maps Engine account that's set up according to [this guide](#).
- An [API Console](#) project with the [Google Maps Engine API](#) activated. If your development language is Java, you can contact [Google Enterprise Support](#) to get access to a Java library that will help you get started.

About data limits

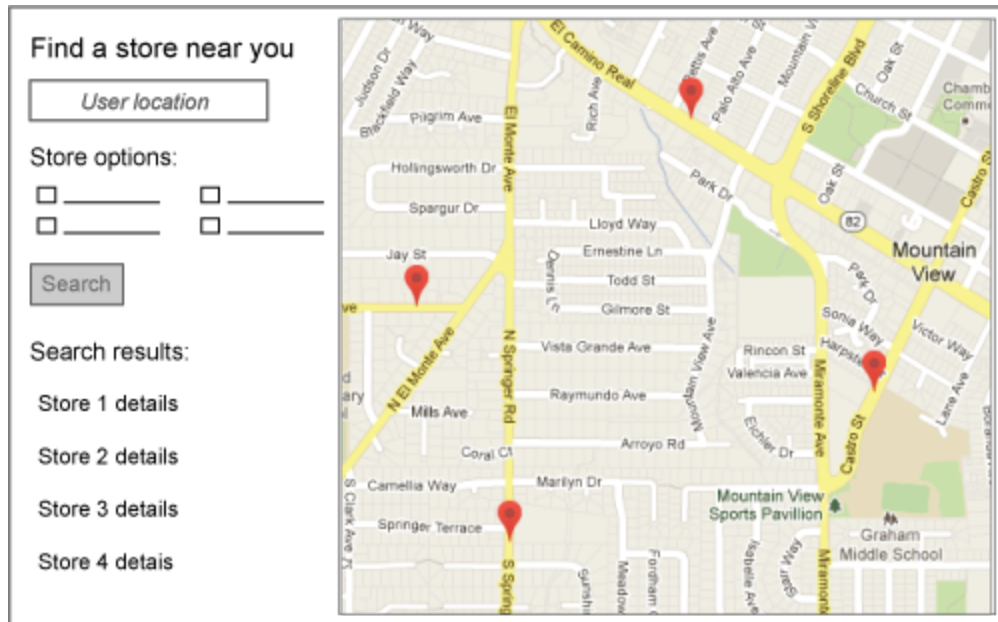
To create a store locator, you'll add your store locations to Maps Engine in a vector table. Although store location data won't typically exceed Google Map Engine's vector table limits, we recommend that you [verify those limits](#).

Note for Fusion Tables users: The API limit for points is same as that for [Fusion Tables](#), so it should not affect your migration to Google Maps Engine.

About store locator applications

On a company's web site, customers use a "store locator" to find the company's locations, such as its stores, restaurants, or hotels. A store locator can also provide details about these locations, such as their hours of operation and the services they offer.

A typical store locator looks something like this:



Basic steps to create a store locator

In this guide, we'll cover best practices for the basic steps you'll follow to create a store locator:

1. Prepare stores data

First, you need to create a file with your stores' addresses and other information. You'll convert, or *geocode*, the addresses in your file into latitude/longitude ("lat/long") coordinates, either automatically or manually, depending on how many stores you have. The Google Maps API for Business uses these coordinates to draw markers on the map at the locations of the stores and to perform spatial search queries to find the stores around a location.

2. Upload stores data to Google Maps Engine

Next, you'll upload your stores data file to Google Maps Engine. If you chose the automatic geocoding option, Maps Engine will geocode your store addresses upon upload. You'll also need to establish a method to keep your stores data updated.

3. Geocode the searched area

For this step, you'll include a mechanism for your store locator to geocode the user's current location or the location the user enters, so your store locator can return stores in the area for which the user is interested.

4. Search for and display stores

To ensure your store locator returns stores that match the customer's search criteria—such as store type, location, hours of operation—you'll need to write some code that queries the Maps Engine API.

5. Add features to your store locator (optional)

Add more features of the Google Maps API for your users, such as directions and street view.

Creating a stores data file

The first step in creating a store locator is to create a comma-separated values (CSV) file with your stores' addresses and details, such as store hours, phone numbers, etc. Then you'll need to geocode the addresses into latitude/longitude ("lat/long") coordinates and verify the results. The format of the file you create depends on whether you'll geocode the addresses *automatically* or *manually*.

- **Automatic geocoding:** You can format your stores data file such that the addresses will be automatically geocoded when you upload them to Maps Engine. Maps Engine can automatically geocode CSV files that contain up to 100,000 addresses. However, Maps Engine doesn't display the confidence level in matching an address with its lat/long coordinates, like the Geocoding API does if you manually geocode addresses. Therefore, because you'll need to manually verify the geocodes, Google recommends automatic geocoding for store locators with up to 1000 stores only.
- **Manual geocoding:** Before you upload your stores data to Maps Engine, you can manually geocode your store addresses by using a script that you create to call the Geocoding API. A significant benefit of manually geocoding addresses is that you can include filters in your script to automatically identify low-confidence results, which you can then investigate and correct. Therefore, if you have a larger number of stores, such as over 1000, the manual geocoding method is much more practical than the automatic geocoding method for verifying the accuracy of the geocodes.

Note: If you're using Google Fusion Tables for your store locator, you can export your stores data to a file and upload it to Google Maps Engine.

To create a stores data file

- Use a spreadsheet editor, a custom database solution, or text editor. Before uploading your data to Maps Engine, you'll need to save or export the data to a CSV file.
- If you're using the automatic geocoding method, make sure your file includes a header row that contains at least one column named `gx_location`. Maps Engine uses this column for geocoding.
- If you're using the manual geocoding method, you'll need enter the geocodes for your store addresses to your CSV file before uploading it to Maps Engine. See "Manually geocoding store addresses" below.
- In your data file, it's a good practice to include a column named `Unique_ID`, in which you enter a unique string for each store. That way, you can modify the table programmatically with details such as open hours and contact phone numbers, using the same primary key in Maps Engine as you use in your database. [Learn more about primary keys.](#)

Tip: If you also upload your store details to [Google Places for Business](#), you might want to maintain both your Places and store locator data in a single source, such as table in your database or in a spreadsheet. If you do, though, be aware that the Places Bulk Uploader won't accept the `gx_location` field that Maps

Engine requires.

Example stores data file for automatic geocoding:

```
Unique_ID, Name, gx_location, Hours  
StoreCode1, StoreName1, Address1, OpeningHours1  
StoreCode2, StoreName2, Address2, OpeningHours2
```

Example stores data file after manual geocoding:

```
Unique_ID, Name, Address, Lng, Lat, Hours  
StoreCode1, StoreName1, Address1, Lng1, Lat1, OpeningHours1  
StoreCode2, StoreName2, Address2, Lng2, Lat2, OpeningHours2
```

Manually geocoding store addresses

If you prepared a stores data file for *manual geocoding*, create a script to call the [Geocoding API](#) for each store address and retrieve a lat/long coordinate for it. For more information, watch this [video tutorial](#) on batch geocoding your addresses.

Watchpoints for manual geocoding

Handle quota errors gracefully

If geocoding requests are made at a rate that exceeds your contracted per second or per day limits, the Geocoding API returns quota errors. Your script should handle these errors by pausing and retrying a limited number of times.

To handle quota errors gracefully, see the [the Maps API usage limits article](#). Also make sure to [use your client ID](#) when accessing the Maps API to benefit from your Maps API for Business quota uplift.

Ensure each address returns appropriate lat/long coordinates

Lat/long coordinates might not be returned if the address in your file does not exactly match the address in the Google Maps database. Or, you might receive coordinates for a different address that more closely matched one in the Google database.

To be able to identify these errors in your geocoded addresses so you can correct them before you import them to Maps Engine, do the following:

- If your store data includes address components such as address, city, state, and ZIP code, use the [address component filters](#) to make sure the geocoding results match these constraints.
- Make sure your script checks the value of the `partial_match` flag. Results that have the

`partial_match` flag set may not be those that you expected.

To export a data file from Google Fusion Tables

If you're migrating your stores data from Google Fusion Tables, you can export your data to either a [CSV file](#) or a [static KML file](#), and then upload it to Google Maps Engine.

We recommend uploading a CSV file, because KML has a few drawbacks:

- **Data duplication:** The imported data contains the `kml_name` and `kml_description` columns. These columns usually duplicate existing columns and are not useful in the table.
- **Loose typing:** All the columns in the resulting table are typed as "string."

Your exported data will include lat/long coordinates, so you don't need the `gx_location` column in your file. However, you will need to [create a VRT file](#) that describes how Maps Engine should parse the CSV file. The VRT file must be a text file with the same name as your CSV file, but with the extension `.vrt`.

Uploading data to Google Maps Engine

Once you've created a correctly formatted CSV stores data file (or exported a KML file from Fusion Tables), you're ready to upload the file to Google Maps Engine.

Uploading data for the first time

To upload data in a CSV file:

1. In Google Maps Engine, click **Upload > Upload source files**.
2. Click **Browse** and select your CSV file. (If you downloaded your CSV file from Google Fusion Tables, also select your accompanying VRT file.)
3. Check the **Select primary key** box if you want to choose your own primary key. [Learn more about primary keys](#).
4. Click **Upload**.
5. If you checked the **Select primary key** box in step 3, select the column you want to use as your primary key. For example, if you included the `Unique_ID` column mentioned above, select that for your primary key. Or select **Defined by Maps Engine** to allow Maps Engine to create one for you.
6. Click **Continue**.

To upload data in a KML file

1. In Google Maps Engine, click **Upload > Upload source files**.
2. Click **Browse** and select your KML file.
3. Select **Vector table** as the data source to create, and then click **Upload**.
Make sure you upload the file as a vector table, **not** as KML, so Google Maps Engine can parse the

KML file and the API can query the data.

Upload

One file selected for upload.

Select the main file and its associated (sidecar) files. Associated files share the same file name but have different extensions. [Information about supported file formats.](#)

[Browse for additional files](#)

0 B / 33.1 KB [Remove](#)

Upload 1 files as: ☒ Vector table ☐ KML

Name: *

Shared with: *

Optional content

Description:

Tags:

☐ Select primary key.

[Upload](#) [Cancel](#)

Watchpoints for automatic geocoding

If you uploaded a stores data file for automatic geocoding, make sure you do the following:

- **Verify geocodes:** Although Maps Engine will geocode your data as accurately as possible, you'll still need to verify the results. In there's any ambiguity in your data, Maps Engine might fail to geocode it accurately. If this happens, you can manually edit the store address using the Maps Engine table editor. Often, just adding the country code— for example UK or FR—to the address will remove the ambiguity so Maps Engine can geocode it successfully.
- **Verify locations:** Maps Engine does not return all of the location data that the [Google Geocoding API](#) provides for a geocode, such as the `location_type` field and address components. Therefore, you should verify each location before your store locator goes live. Google makes every effort to provide accurate locations, but with your local knowledge, you may be able to provide a more accurate location for some addresses.

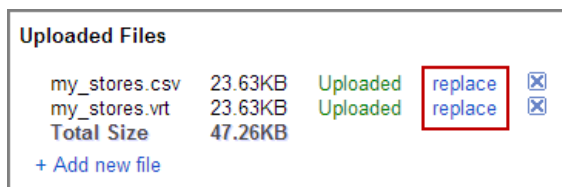
Updating uploaded data

You can update the stores data you uploaded to Google Maps Engine using a manual or automated process.

Manually update data

Follow these steps to update your stores data manually:

1. Update your source file (CSV or KML file), and then upload it again to Google Maps Engine.
2. On the “Data details” page for the data source you uploaded, replace the files:



Note: When you replace files using this method, the API continues to serve the data in the previous versions of the files until the replacement is complete and successful. Therefore, your customers won't experience any delays and can keep using your store locator while the data is being updated or if an error occurs during the update.

Advantages	Disadvantages
Simple process, with no scripts to write.	Process requires more manual effort than automatic update methods.

Automatically update data

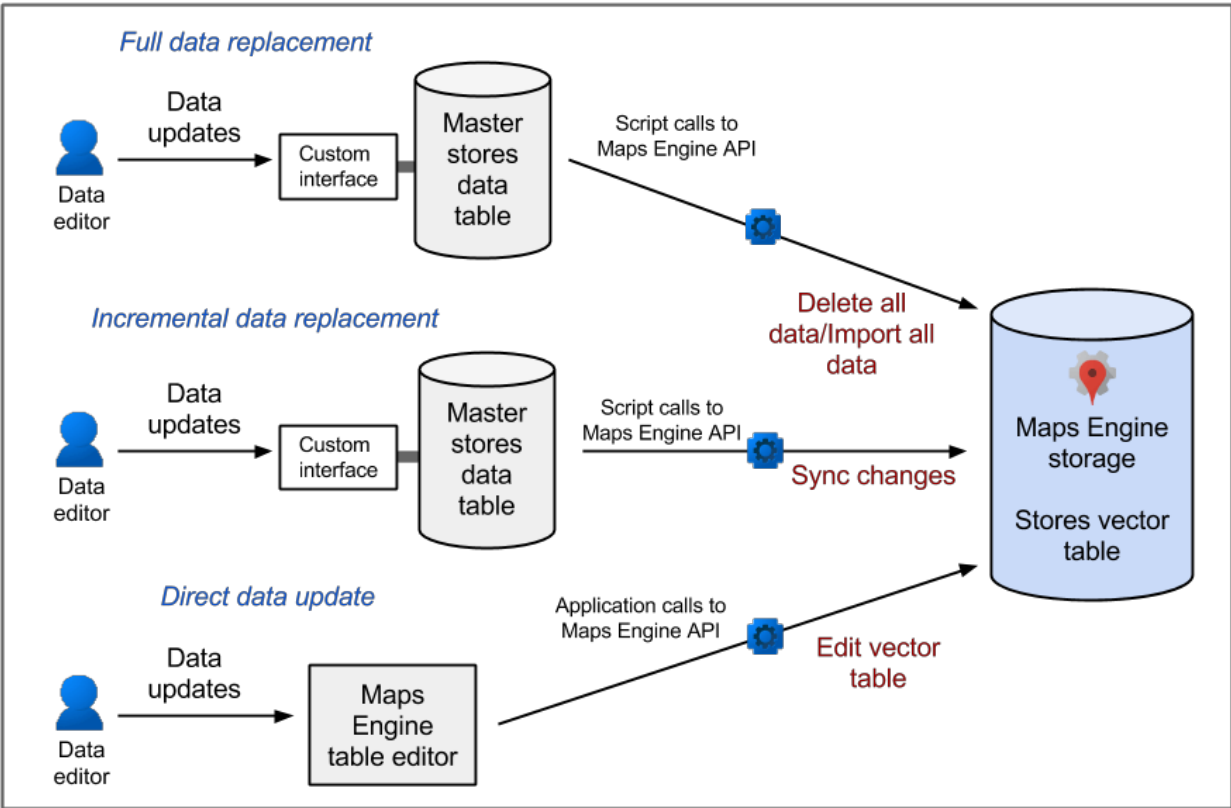
You can use the update, delete, and insert methods of the Maps Engine API to update data programmatically, using one of these methods:

- If you maintain a master table of stores data in your own database, instead of Google Maps Engine, and mirror the data in Maps Engine, you can fully or incrementally replace data through a custom interface you develop.

Note: You might want to keep your data in a separate database if you also bulk upload that data to Google Places for Business.

- If you use Google Maps Engine to store and maintain data, you can edit your vector table using the Maps Engine table editor.

The following diagram shows an overview of your data update options:



Full data replacement

With this method, you regularly delete all the data from your Maps Engine vector table, and then upload all the data again from your master stores table in your database, using a synchronization script.

Advantages	Disadvantages
The synchronization script is easy to create.	• It can take several minutes to delete your vector table and upload all your data again. • You need two vector tables—one for live data and another as a backup—to prevent downtime for your store locator as data is updated. • Inefficient use of Maps Engine resources, which can increase your Maps Engine bill.

Incremental data replacement

With this method, you update your vector table in Maps Engine after you update the master stores table in your database. Here's an example:

1. Add these two columns to your master stores table:

- `master_modification_date`, which stores the last time the row was edited in your master table
- `mapsengine_sync_date`, which stores the last time the row was synced to Maps Engine

2. Regularly update all the rows in your data in Maps Engine based on these modification dates (update all the rows that have `master_modification_date > mapsengine_sync_date`).

Note: Make sure you use [batchInsert](#) method for additions and [batchDelete](#) method for deletions.

Advantages	Disadvantages
• Efficient use of Maps Engine resources. • Data is updated very quickly in Maps Engine.	• Requires development of a more complex synchronization script.

Update data directly in Google Maps Engine

If you use Maps Engine as your master stores data, instead of your own server, you can use the Google Maps API to create an interface to directly edit the data in Maps Engine.

Advantages	Disadvantages
• No need to write a sync script. • You can make updates manually using the Maps Engine table editor.	• All updates are reflected directly in Maps Engine (not grouped), so this solution uses more write requests (and therefore, more Map Load quota) than the incremental backup solution. • Maps Engine does not support data snapshots (saved versions). Because it's always possible that your script to write updates to Maps Engine can cause an error, you'll need to backup your data in another table on a regular basis.

Recommendation

We recommend that you use the incremental data replacement method to update your stores data regularly.

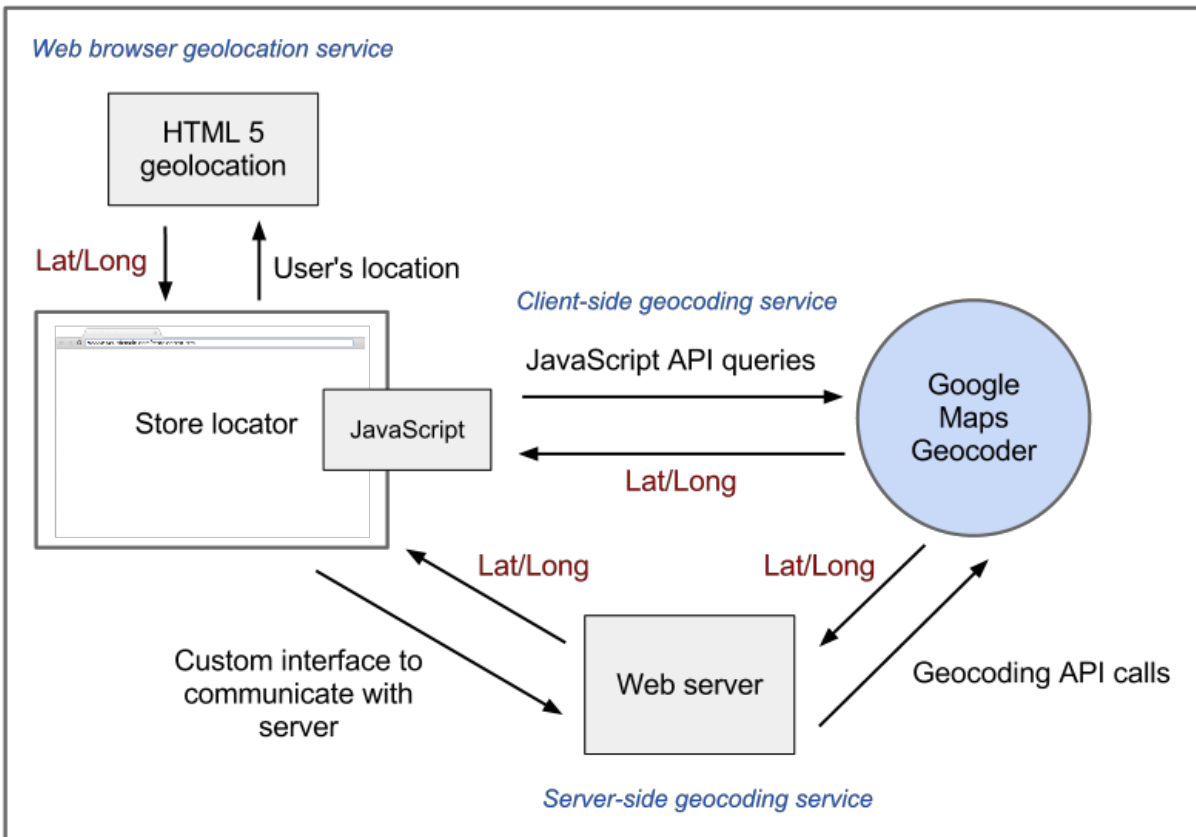
Data update watchpoints

- **Use a failover table.** If you're using your own database for your master stores data, maintain two tables for your data: a "live" table and a failover table. Because it's possible that a data update will fail (for example, by the incorrect use of the API), always update your data in the failover table first. Once you determine the update was successful, make the failover table the live table.
- **Use batch queries.** The Maps Engine API supports up to 50 write requests for each of the following HTTP requests: [batchInsert](#), [batchDelete](#), and [batchUpdate](#). Instead of creating one HTTP request for each modification to your data, use the API's batch query functionality.

Geocoding the search area

Architecture

To geocode the area in which a customer searches for stores, you can use a web browser service, a client-side service, or a server-side service. The following diagram shows an overview of your geocoding options:



Web browser geolocation service

HTML5 includes the following JavaScript, which provides access to a user's location:

```
navigator.geolocation.getCurrentPosition()
```

If the user's browser supports HTML5, it prompts the user to share his or her current location and passes a latitude/longitude value to your application. For more information, see the [Geolocation article](#) on the Google Maps API developer's site.

Advantages	Disadvantages
• Automatic, with minimal action required from the user. • There's no quota on geolocation requests.	• Not all browsers support HTML5 geolocation, and some users might decline to allow the browser to access his or her location. Therefore, in addition to the web browser service, you'll need to use the client-side service as a fallback. • Returns the user's current location, not necessarily the area the user is interested in.

Client-side geocoding service

The Google Maps JavaScript API provides a [client-side geocoding service](#), which runs in the web browser and converts a user's address to a latitude/longitude value.

Advantages	Disadvantages
• Quota is limited per session and tracked by the API separately per user, so it scales as your user base grows. A user is not likely to reach the quota, which is rate-limited at 1 geocode per second. • The service is loaded in the client as part of the JavaScript API, so no additional libraries are needed.	None.

Server-side geocoding service

You can use the [Google Maps Geocoding API](#) (server-side) to build a component on your server that geocodes a user's address to a latitude/longitude value. To use the Geocoding API in your store locator, requests need to be [signed](#) with your crypto key to generate a unique signature for authentication.

Advantages	Disadvantages
• Allows you to log searches so you can cache responses to common queries. • Allows you to supplement Google's results with another database that has better results for bad queries or corrections for missing addresses.	• <i>Quota is shared among all users of your geocoding service.</i> The number of geocoding requests, therefore, is limited by the geocoding quota you've purchased. The default quota is 100,000 geocodes per day and no faster than 10 geocodes per second (that is, no more than 10 users can make a request at the same time). If your store locator requires more geocodes, you can purchase additional quota. • Because this solution requires a server-side component, users' queries cause additional load on your server.

Recommendation

Use both the web browser and client-side geocoding services: Set the map viewport initially to the user's location (if available) using the browser's geolocation service, and then let the user select their location using the client-side geocoding service.

Watchpoints

- **Avoid using the server-side Geocoding API, if possible:** Because the Geocoding API is limited by the quota you purchased, your store locator might become unavailable if there's a burst of traffic on your website. With the client-side geocoding service, on the other hand, there's no risk of reaching a global limit. [Learn more](#) about the differences between these strategies
- **Minimize JavaScript API loads:** The Google Maps JavaScript API loads each time a page refreshes, which uses your API quota. To avoid additional API charges, you can use AJAX techniques to load the API once per session. For example, if your store locator initially shows only a search box, without the map, then do not geocode it on the first page. Instead, pass the location to the map page, where the API can geocode it, and then query the list of results for this location with an AJAX call. Also, use AJAX to limit page refreshes when users perform actions on the map page that require communication with the server (for example, when the user enters a new search location).

Searching for and displaying stores

Once a user selects his or her location and search criteria (for example, hotels with a pool), your store locator submits a query to the Maps Engine API to retrieve information about your stores from your stores data table in Google Maps Engine. The user's query can be submitted to the API either server-side or client-side.

Query the Maps Engine API server-side

For server-side queries to the Maps Engine API, the user's browser submits a request for stores data to your server. Your server then generates the query's SQL and makes the authenticated request to the Maps Engine API. The Maps Engine API returns the results to your server, which processes and delivers them to the user.

Advantages	Disadvantages
• Because your server receives the search requests, your stores data table can be private and authenticated, and users can't directly query the data. • Allow you to log searches and cache responses to common queries.	Can cause additional load on your server for each query a user issues.

Query the Maps Engine API client-side

For client-side queries to the Maps Engine API, the user's browser submits a request for stores data directly to your publicly accessible stores data table in Maps Engine. See the Google Maps Engine API developer's site for documentation on [querying public data](#) and information about using [JSONP or CORS technologies](#) to query the data from the browser.

Advantages	Disadvantages
• No additional server components required; clients query tables directly with JavaScript. • Queries processed directly by Google infrastructure, so there are no scalability or load concerns. • Google's cloud infrastructure provides very low latency between client and server, providing a better user experience. • Lower cost: Non-authenticated queries are billed as public map loads, which are much less expensive than private map loads.	• Tables must be public to allow user queries, without requiring OAuth credentials. • Third-parties can query your data programmatically, so this method is not suitable if your data is sensitive. • Limited visibility into and control of sources that make requests for your data.

Using the Maps Engine Layer

With [Maps Engine Layer](#), you can display all the content in your stores data table to provide an "overview" map. Users can then browse all your locations at once, without providing a location. If you have many stores (more than a few thousand), Maps Engine Layer can improve the performance of your application for users, by moving the traditional client-side rendering of markers/points (`google.maps.Marker` objects) for your locations to the Maps Engine server.

Query Syntax

The Google Maps Engine API supports many SQL operators, but it doesn't support full SQL query strings. Instead, the query components must be broken down and passed in different parameters.

Parameter	Value
select	<code>store_id,phone_number,geometry</code>
where	<code>open=1 AND drive_thru=2</code>

Note: If you created a store locator using Google Fusion Tables, your queries had the following syntax:

```
SELECT store_id, phone_number, geometry FROM table123 WHERE open = 1 AND drive_thru = 2
```

Maps Engine API query syntax:

```
https://www.googleapis.com/mapsengine/v1/<tableId>/features?select=store_id,phone_number,geometry&where=open=1%20AND%20drive_thru=2
```

The Maps Engine API lets you filter results with [spatial conditions](#):

- To display all stores within x meters of a given point:

```
ST_DISTANCE(geometry,ST_POINT(1.23,4.56))<x
```

- To display all stores in the current viewport:

```
ST_INTERSECTS(geometry,ST_POLYGON(ST_LINESTRING(ST_POINT(-1, -1),ST_POINT(1, -1), ST_POINT(1, 1), ST_POINT(-1, 1), ST_POINT(-1, -1))))
```

The various points in the `ST_LINESTRING` are the four corners of the current viewport.

The following is an example query, which finds stores that are less than 10,000 meters away from the center of the search, ordered by distance from the center, and limited to the 50 closest stores:

```
https://www.googleapis.com/mapsengine/v1/tables/<tableId>/features
?select=ST_DISTANCE(geometry, ST_POINT(0, 51)) AS distance, geometry,
store_id, phone_number,
&where=ST_DISTANCE(geometry, ST_POINT(0, 51)) < 10000
&orderBy=distance ASC
&limit=50
```

For details about the parameters and geometry functions used in this example query, see the Google Maps Engine API documentation on [Read Requests](#).

Watchpoints

Channel parameter not available

The [channel](#) parameter is not available for tracking usage with the Google Maps Engine API. As a workaround, you can provide a custom channel (for example, “marketing_website”, or “sales_website”) in the HTTP referer header. That way, you can see usage broken down by referer in the [API Console](#). Note, however, that this technique doesn’t work for requests made from a browser, because the browser automatically sets the referer header to the URL of the website from which the request originates. But for these requests, the referer header still indicates the source application for requests.

Optimizing performance

The following are some ways you can optimize the performance of your store locator and improve users’ experience.

Return an initial subset of data

If your data table has many columns of data, and a user’s query could return a large data set, pull only a subset of the data initially, as follows:

1. Query the Google Maps Engine API to retrieve only the unique key and corresponding latitude/longitude coordinates (geometry) from your table,
2. Sort the results by distance from the user.
3. Query the data to retrieve details for only those stores closest to the user and display on the map.
4. Use subsequent queries to the Google Maps Engine API to retrieve details for stores only when that content is needed (for example, if a user clicks on a store).

Retry failed queries

A low percentage of queries (less than 0.05%) will fail with an HTTP error 503, with the reason “backendError”. This error occurs when the Google infrastructure can’t respond to the query at a specific time. To avoid returning this error to your users, retry these requests after a pause of 0.5 seconds. We also recommend that you retry the rate limit exceptions:

- `userRateLimitExceeded`: This error occurs if the per-IP usage reaches the limit configured in the API Console. We also recommend that you update the “per-user limits” in the API Console for this project, to make sure you don’t reach these limits again.
- `rateLimitExceeded`: This error occurs if the per-project usage reaches the limit that’s configured in the API Console. If you reach the limit, you’ll need to [file a support case](#).

Adding features to your store locator

The following additional features of the Google Maps API can improve your users’ experience with your store locator:

- Use the [Places Autocomplete library](#) to display suggestions as users type their location. (May require purchase of the Places API.)
- Use [StreetView](#) with Google’s imagery to show the front of the store.
- Use your own, [custom StreetView imagery](#) to show the inside of a store and provide indoor navigation for larger facilities, such as a shopping mall.
- Use the [Distance Matrix service](#) to implement drive time/distance filtering for search results.
- Use the [Places library](#) to show nearby parking garages or other useful and relevant amenities. (May require purchase of the Places API.)
- Customize the map to match the style of your company website
 - [Style the map](#) using the [wizard](#)
 - Use [Custom controls](#)
 - Use [Custom markers](#) with optional animation