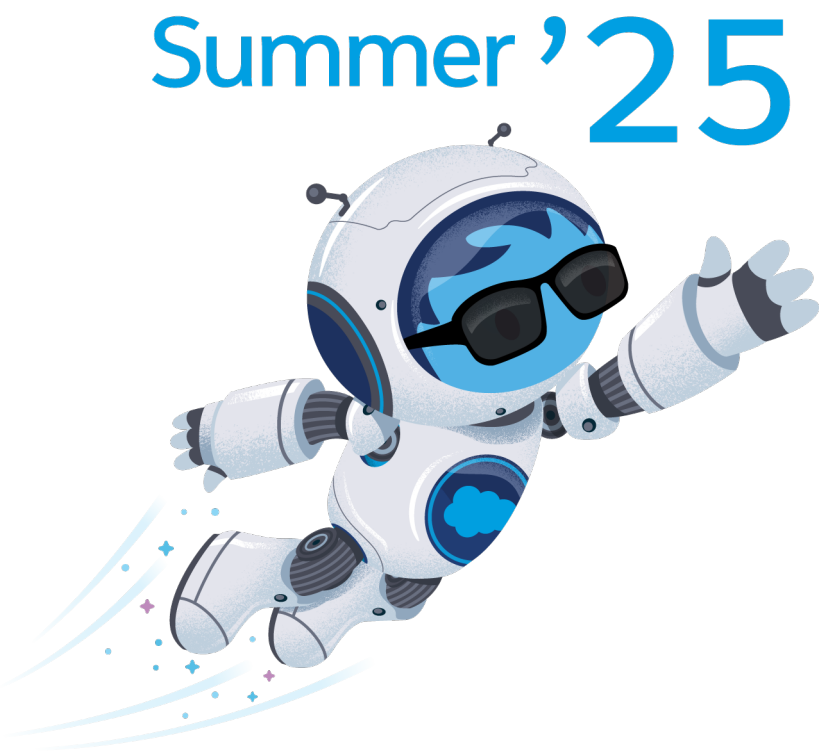
Salesforce Maps Apex Developer Guide

Version 64.0, Summer '25



Last updated: May 16, 2025

CONTENTS

CHAPTER 1 Overview

Get your sales and service teams to spend more time building customer relationships and less time on the road when you fine-tune your Salesforce Maps implementation using Apex. For example, return estimated travel distances and times between locations, get geographical coordinates for addresses, and retrieve geometric data for areas and perimeters.

API Performance Considerations and Guidelines

If your processes rely heavily on the API, you can experience decreased performance and server errors. Apex methods don't count against org API limits.

Server Error	Potential Resolution
(5xx) or throttling	Retry your requests. For better flow control, you can implement an exponential back-off algorithm. Such algorithms use progressively longer wait times between retries for consecutive error responses. Implement a maximum delay interval and a maximum number of retries. You can base those maximums on performed operations and other local factors, such as throughput capacities or request throttling.
(4xx)	Revise your requests to correct any problems before trying again.

SEE ALSO:

Salesforce Help: Salesforce Maps

Apex Developer Guide

CHAPTER 2 Salesforce Maps Apex Requirements

This guide introduces you to the Apex methods you can use to perform Salesforce Maps operations without using the UI.

Before using Apex methods, ensure that you have:

- Access to Apex classes using Enterprise, Unlimited, or Developer Editions
- Salesforce Maps installed
- A license for Salesforce Maps or Salesforce Maps Advanced. Methods to get current location and location history also require a license for Salesforce Maps Live Tracking
- Access to Salesforce Maps objects and Apex classes granted through a permission set: SF Maps, SF Maps Advanced, SF Maps Live Admin, Maps User, or Maps Admin

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

CHAPTER 3 Salesforce Maps Apex Methods

In this chapter ...

Apex is a typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Salesforce Platform server, in conjunction with calls to the API. Use the sample code in this Apex method documentation as a starting point for your Salesforce Maps implementation.

Assign Records to Owners Using an Assignment Plan

The `AssignRecords()` Apex method assigns records to users automatically using rules specified in an assignment plan. For example, suppose that you use a web-to-lead generation form to generate inbound leads in Salesforce. You use the `AssignRecords()` method to assign those leads to sales reps as soon as the lead records are created, reducing the time for sales reps to contact leads.

Signature

```
String, List<sObject> maps.API.AssignRecords(String, List<sObject>)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `AssignRecords()` is the method.

Sample Code

This code assigns accounts to reps based on an existing assignment plan. The `AssignRecords()` method returns a `List<sObject>` containing the records that were assigned to an assignment plan, the plan ID, and the records that weren't assigned to an assignment plan.

The `AssignRecords()` global method uses the last record cache to assign records. As a best practice, schedule assignment of records frequently in a batch to keep the cache up to date. For more information, see *Salesforce Help*: Schedule and Run Record Assignments.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Example

```
// Get the record ID of the assignment plan.
maps__AssignmentPlan__c plan = [SELECT Id FROM maps__AssignmentPlan__c];

// Get a list of records you want to assign.
// This query includes all fields mapped in the plan.
List<Account> accs = [SELECT Id, BillingLatitude, BillingLongitude FROM Account];

// Call the AssignRecords method with the plan ID and list of account records.
maps.API.AssignRecords(plan.Id, accs);

// Log the record assignment output in JSON format.
Map<String, Object> response = maps.API.AssignRecords(plan.Id, accs);
system.debug(JSON.serialize(response));
```

Invocable Example

```
public class InvocableTest {

    @future(callout=true)
    public static void testInvocableFuture(List<Id> accIds) {
    // This query includes all fields mapped in the plan.
        List<Account> accs = [SELECT Id, BillingLatitude, BillingLongitude, Description,
maps__AssignmentRule__c FROM Account WHERE Id IN :accIds];

        // Call the AssignRecords method with the plan ID and list of account records.
        maps.API.AssignRecords('a0L54000007oHXqEAM', accs);

        // Log the record assignment output in JSON format.
        Map<String, Object> response = maps.API.AssignRecords(plan.Id, accs);
        system.debug(JSON.serialize(response));
    }

    public with sharing class testInvocableQueueable implements Queueable,
Database.AllowsCallouts {
        List<Id> accIds;

        public testInvocableQueueable(List<Id> accIds) {
            this.accIds = accIds;
        }

        public void execute(QueueableContext qc) {
            // This query includes all fields mapped in the plan.
          List<Account> accs = [SELECT Id, BillingLatitude, BillingLongitude, Description,
 maps__AssignmentRule__c FROM Account WHERE Id IN :accIds];

           // Call the AssignRecords method with the plan ID and list of account records.

            maps.API.AssignRecords('a0L54000007oHXqEAM', accs);

            // Log the record assignment output in JSON format.
            Map<String, Object> response = maps.API.AssignRecords(plan.Id, accs);
            system.debug(JSON.serialize(response));
        }
    }

    @InvocableMethod(label='test invocable' description='test auto assign' callout=true)
    public static void testInvocable(List<Id> accIds) {

        // CALLING A FUTURE METHOD WORKS.
        //testInvocableFuture(accIds);

        // CALLING A QUEUEABLE METHOD WORKS.
        //System.enqueueJob(new testInvocableQueueable(accIds));

        // THIS WON'T WORK AS AN INVOCABLE METHOD. A CALLOUT INSIDE OF A RECORD UPDATE OR
 INSERT RESULTS IN AN UNCOMMITTED WORK ERROR.
        // INSTEAD, SEPARATE THE TRANSACTION INTO A FUTURE OR QUEUEABLE METHOD.
       //List<Account> accs = [SELECT Id, BillingLatitude, BillingLongitude, Description,
 maps__AssignmentRule__c FROM Account WHERE Id IN :accIds];
```

```
        //maps.API.AssignRecords('a0L54000007oHXqEAM', accs);
    }
}
```

Trigger Example

```
// FOR INSERTS, AFTER WORKS BETTER THAN BEFORE BECAUSE THE ID DOESN'T EXIST IN THE BEFORE
 CHECK.
trigger AccountTrigger on Account (after insert, after update) {

    List<Id> accIds = new List<Id>();

    for (Account acc : Trigger.new) {
        if (acc.maps__AssignmentRule__c == null) {
            accIds.add(acc.Id);
        }
    }

    // CALLING A QUEUEABLE METHOD WORKS.
    //if (!accIds.isEmpty()) {
        //System.enqueueJob(new AccountTriggerHelper.testTriggerQueueable(accIds));
    //}

    // THIS WON'T WORK AS A TRIGGER.
    //List<Account> accs = [SELECT Id, BillingLatitude, BillingLongitude, Description,
maps__AssignmentRule__c FROM Account WHERE Id IN :accIds];
    //maps.API.AssignRecords('a0L54000007oHXqEAM', accs);

    // CALLING A FUTURE METHOD WON'T WORK.
    //AccountTriggerHelper.testTriggerFuture(accIds);

}
```

```
public class AccountTriggerHelper {

    @future(callout=true)
    public static void testTriggerFuture(List<Id> accIds) {
    //  This query includes all fields mapped in the plan.
        List<Account> accs = [SELECT Id, BillingLatitude, BillingLongitude, Description,
maps__AssignmentRule__c FROM Account WHERE Id IN :accIds];
        // Call the AssignRecords method with the plan ID and list of account records.
        maps.API.AssignRecords('a0L54000007oHXqEAM', accs);
    }

    public with sharing class testTriggerQueueable implements Queueable,
Database.AllowsCallouts {
        List<Id> accIds;

        public testTriggerQueueable(List<Id> accIds) {
            this.accIds = accIds;
        }

        public void execute(QueueableContext qc) {
        // This query includes all fields mapped in the plan.
          List<Account> accs = [SELECT Id, BillingLatitude, BillingLongitude, Description,
```

```
 maps__AssignmentRule__c FROM Account WHERE Id IN :accIds];
            // Call the AssignRecords method with the plan ID and list of account records.

             maps.API.AssignRecords('a0L54000007oHXqEAM', accs);

        // Log the record assignment output in JSON format.
        Map<String, Object> response = maps.API.AssignRecords(plan.Id, accs);
        system.debug(JSON.serialize(response));
        }
    }

}
```

Sample Response

This method returns an Apex `Map<String, Object>` object that contains the records assigned, the plan ID, and records not assigned. This JSON response illustrates the essential data returned.

```
{
    assigned: [
        {
            id="0011700001LTxzwAAD", // ID of record assigned.
            rule="0021700001LTxzwAAD", // ID of rule used for assignment.
            user="0031700001LTxzwAAD" // ID of user assigned to the matched rule.
        },
        {
            id="0011700001LTxzwAAG",
            rule="0021700001LTxzwAAD",
            user="0031700001LTxzwAAD"
        }
    ],
    plan: "a0L54000007oHXqEAM", // Assignment plan ID
    unassigned: ["0011700001LTxzwAAE", "0011700001LTxzwAAF"] // List of record IDs not
assigned.
}
```

SEE ALSO:

Salesforce Help: Automate Assignments for Salesforce Records

Create a Route

The `CreateRoute()` Apex method creates a record that includes a list of waypoints that reps access in Salesforce Maps. The record doesn't include a route between waypoints.

Signature

```
maps__Route__c maps.API.CreateRoute(map<string,object> options)
```

Where,

- `maps__Route__c` is a Salesforce Maps route record containing a list of waypoints.
- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `CreateRoute()` is the method.

Allocations

This method requires 2–25 waypoints.

Sample Code

This code returns a `maps__Route__c` record of waypoints of accounts with billing addresses in Atlanta.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Input Format of Waypoints

```
{
version: '1', // Required. Version of the API endpoint. Must be '1'.
name: 'String', // Name of the new route record.
date: Date, // Time and date in Epoch format that the route is created for. Default is the
 current time and date.
waypoints: [ {recordId:String, lat:Decimal, lng:Decimal, address:String, order: Integer,
baseobjectid: String, markerlayerid: String, notes:String}, {...} ] // List of waypoint
objects.
}
```

Example

```
// Create a list of waypoints.
List<Map<String,Object>> waypoints = new List<Map<String,Object>>();

// Retrieve the list of billing addresses from the database and
// add them to the waypoints object.
List<Account> accountList = [Select
Id,BillingStreet,BillingCity,BillingState,BillingPostalcode, BillingLatitude,BillingLongitude
 From Account Where BillingCity = 'Atlanta' LIMIT 25];

for(Integer i = 0; i < accountList.size(); i++){
    Account thisAccount = accountList[i];
    String address = thisAccount.BillingCity + ', ' + thisAccount.BillingState + ' ' +
thisAccount.BillingPostalCode;
    waypoints.add(new Map<String,Object>{
        'lat' => thisAccount.BillingLatitude,
```

```
        'lng' => thisAccount.BillingLongitude,
        'address' => address,
        'recordId' => thisAccount.id,
        'baseobjectid' => 'a0N0t000002hEyvEAE',
        'markerlayerid' => 'a0v0t000001wE91AAE',
        'order' => i + 1
    });
}

// Build the request.
Map<String,Object> request = new Map<String,Object>();
request.put('version','1');
request.put('name','Atlanta Accounts Route');
request.put('waypoints',waypoints);

// Call the CreateRoute() method with the waypoints.
maps__Route__c response = maps.API.CreateRoute(request);

// Log the route record output in JSON format.
system.debug(JSON.serialize(response));
```

Sample Response

This method returns a `maps__Route__c` record, which inserts the waypoints that appear in Salesforce Maps. This JSON response illustrates the essential data stored in the route record.

```
{
  "attributes": {
    "type": "maps__Route__c",
    "url": "/services/data/v51.0/sobjects/maps__Route__c/a0y0t000001ht8HAAQ"
  },
  "Name": "Atlanta Accounts Route",
  "maps__Date__c": "2021-05-04",
  "Id": "a0y0t000001ht8HAAQ"
}
```

Create Routes for Specific Users

The `StartAdvancedOptimizationForUsers()` Apex method creates routes for users assigned to any active Salesforce Maps Advanced visit plan. Use this method to leverage Salesforce Maps Advanced visit planning from your custom workflow or app, such as a retail execution app. By using Apex, you can generate routes for reps automatically without having to make field reps click through the UI to manually generate routes.

The `StartAdvancedOptimizationForUsers()` Apex method is the equivalent of clicking **Plan My Visits** for users in Maps Advanced Route.

When you use this method, only the user's future shifts are included for planning visits along the route. If a user's shift has started or is within 2 hours of starting at the time the method runs, the method doesn't change visit appointments for today's date. In that case, one day is added to the starting date (`startDate`).

Visits are created from `startDate` through a visit plan's end date. If a user is assigned to more than one visit plan, then visits are planned for the length of time dictated by the visit plan with the end date furthest in the future.

Example: A user is assigned to consecutive plans Visit Plan A and Visit Plan B. At the time the method is run, Visit Plan A has 10 days remaining until its end date. Visit Plan B has a start date immediately after Visit Plan A's end date, and has 30 days remaining until its end date. When the method runs, visits are scheduled for the user for the remainder of Visit Plan A (10 days), and scheduled for 20 days in Visit Plan B, for a total of 30 days.

Signature

```
Map<String, Object> maps.API.StartAdvancedOptimizationForUsers(Set<Id> userIds, Date
startDate)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `StartAdvancedOptimizationForUsers()` is the method.
- `userIds` is a set of IDs of the Salesforce users for which you want to generate routes. Every user must be assigned to an active visit plan for the date you specify in `startDate`.
- `startDate` is optional. If not specified, the method uses today's date as the date to start planning visits. If `startDate` occurs during a user's shift or within 2 hours of the user's next shift, then one day is added to the specified `startDate`.

Sample Code

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Example

```
// Create a userIds variable to hold the Salesforce user IDs.
Set<Id> userIds = new Set<Id>();

// Create a variable for the current or future date.
Date today = Date.today();

// Check that each user is assigned to an active visit plan for specified date.
List<maps__AdvRouteTemplate__c> activeTemplates = [SELECT Id FROM maps__AdvRouteTemplate__c
 WHERE
maps__StartDate__c <= :today AND (maps__EndDate__c >= :today OR maps__IsRepeating__c =
TRUE) AND
maps__Active__c = TRUE LIMIT 1];

// Query for the IDs of all users for which we want to generate routes.
for (User u : [SELECT Id FROM User WHERE Name IN ('User1, User2')]) {
    userIds.add(u.Id);
```

```
}

// Call the method.
if (!activeTemplates.isEmpty() && !userIds.isEmpty()) {
    maps.API.StartAdvancedOptimizationForUsers(userIds, today);
}
```

Sample Response

This method returns an Apex `Map<String, Object>` object that contains a boolean flag indicating whether the routes were created successfully.

```
{
    success: true
}
```

If the routes weren't created successfully, an error message is returned. In this context, optimization refers to creating routes.

```
{
    success: false
    error: "Please provide either the current or future date in the request to complete
optimization."
}
```

If the routes were created successfully for some users, but not others, a warning is returned. In this example, some users weren't assigned to an active visit plan for the `startDate`, so those users were skipped.

```
{
    success: true,
    warning: "No visit plans are set up for routing on the date you selected for the
following users: [userId]"
}
```

SEE ALSO:

Knowledge Article: Generate Routes for a Single User Assigned to Maps Advanced Visit Plan

Apex Developer Guide

Create Routes from Visit Plan

The `StartAdvancedOptimizationForVisitPlan()` Apex method creates routes for all users assigned to a Salesforce Maps Advanced visit plan. Use this method to leverage Salesforce Maps Advanced visit planning from your custom workflow or app, such as a retail execution app. By using Apex, you can generate routes for reps automatically without having to make field reps click through the UI to manually generate routes.

The `StartAdvancedOptimizationForVisitPlan()` Apex method is the equivalent of clicking **Plan Visits** on an active visit plan in Maps Advanced Visit Plans.

When you use this method, only the user's future shifts are included for planning visits along the route. If a user's shift has started or is within 2 hours of starting at the time the method runs, the method doesn't change visit appointments for today's date. In that case, one day is added to the starting date (`startDate`).

Visits are created from `startDate` through a visit plan's end date.

Signature

```
Map<String, Object> maps.API.StartAdvancedOptimizationForVisitPlan(Id templateId, Date
startDate)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `StartAdvancedOptimizationForVisitPlan()` is the method.
- `templateId` is the Salesforce record id of a visit plan.
- `startDate` is optional. If not specified, the method uses today's date as the date to start planning visits. If `startDate` occurs during a user's shift or within 2 hours of the user's next shift, then one day is added to the specified `startDate`.

Sample Code

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Example

```
// Create a variable for the current or future date.
Date today = Date.today();

// Query for the record ID of an active visit plan.
List<maps__AdvRouteTemplate__c> activeTemplates = [SELECT Id FROM maps__AdvRouteTemplate__c
 WHERE Name = 'NAME_OF_VISIT_PLAN' AND
maps__StartDate__c <= :today AND (maps__EndDate__c >= :today OR maps__IsRepeating__c =
TRUE) AND
maps__Active__c = TRUE LIMIT 1];

// Call the method.
if (!activeTemplates.isEmpty()) {
maps.API.StartAdvancedOptimizationForVisitPlan(activeTemplates[0].Id, today);
}
```

Sample Response

This method returns an Apex `Map<String, Object>` object that contains a boolean flag indicating whether the routes were created successfully.

```
{
    success: true
}
```

If the routes weren't created successfully, an error message is returned. In this context, optimization refers to creating routes.

```
{
    success: false
    error: "Please provide either the current or future date in the request to complete
optimization."
}
```

SEE ALSO:

Knowledge Article: Generate Routes for All Users Assigned to the Maps Advanced Visit Plan

Apex Developer Guide

Get a Route

The `getRoute()` Apex method returns a route between two or more waypoints for the specified mode of transportation. The travel time depends on the mode of transportation. The route appears in Salesforce Maps for the users who requested the route, such as a delivery truck driver and the admin. This method doesn't optimize the route. Instead, it finds a route between the waypoints in entered sequence.

Signature

```
Map<String, Object> maps.API.getRoute(String requestJSON)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `getRoute()` is the method.

Sample Code

This code finds a route for someone driving a car from one grocery store to another.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Input Format of the Request JSON String

The input format supports standard and custom drive profiles. If your drivers require routes optimized for vehicles other than standard cars, create a custom drive profile first, and then add it to your `driveProfile` string.

```
{
driveProfile: 'string' // Mode of transportation. Standard profile values are 'car' and
'bicycle'. For a custom profile, specify its record ID.
waypoints: [ {id:String, lat:Decimal, lng:Decimal, id:String}, ... ] // List of waypoints
hideGeoJSON: boolean // Optional. If set to true, the GeoJSON is excluded from the response
}
```

Example

```
// Create a list of waypoints.
List<Map<String,Object>> waypoints = new List<Map<String,Object>>();
waypoints.add(new Map<String,Object>{
    'lat' => 33.917760,
    'lng' => -84.379750,
    'id' => 'groceryNorth'
});

waypoints.add(new Map<String,Object>{
    'lat' => 33.840630,
    'lng' => -84.381920,
    'id' => 'grocerySouth'
});

// Build the request.
Map<String,Object> request = new Map<String,Object>();
request.put('waypoints',waypoints);
request.put('driveProfile','car'); // For a custom drive profile, replace 'car' with the
ID of your custom drive profile.
request.put('hideGeoJSON', true); // Optional. Add only if removing the GeoJSON from the
response
String routeRequest = JSON.serialize(request);

// Call the getRoute() method with the waypoints and mode of transportation.
Map<String,Object> response = maps.API.getRoute(routeRequest);

// Log the resulting route.
system.debug(response);
```

Sample Response

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

```
{
  "profile": {
    "width": null,
    "weightPerAxle": null,
```

```
    "weight": null,
    "type": "car",
    "trailers": null,
    "length": null,
    "height": null,
    "hazmat": null,
    "axles": null
  },
  "request": {
    "profile": {
      "type": "car"
    },
    "timebased": false,
    "waypoints": [
      {
        "id": "groceryNorth",
        "lat": 33.91776,
        "lng": -84.37975
      },
      {
        "id": "grocerySouth",
        "lat": 33.84063,
        "lng": -84.38192
      }
    ]
  },
  "response": {
    "success": true,
    "geojson": {
      "features": [
        {
          "geometry": {
            "coordinates": [
              [
                [
                  -84.37935,
                  33.91775
                ],
                [
                  -84.37935,
                  33.91744
                ],
                [
                  -84.37913,
                  33.91743
                ],
                [
                  -84.37915,
                  33.91725
                ],
                [
                  -84.37916,
                  33.91717
                ],
```

```
            [
              -84.37916,
              33.91704
            ],
            [
              -84.37916,
              33.91698
            ],
            [
              -84.37919,
              33.91667
            ],
            [
              -84.37919,
              33.91661
            ],
            [
              -84.37922,
              33.91622
            ],
            [
              -84.37923,
              33.91609
            ],
            [
              -84.37925,
              33.91601
            ],
            [
              -84.37925,
              33.91592
            ],
            [
              -84.37928,
              33.91574
            ],
            [
              -84.37929,
              33.91558
            ],
            [
              -84.37932,
              33.91503
            ],
            [
              -84.37932,
              33.9148
            ],
            [
              -84.37932,
              33.91461
            ],
            [
              -84.37929,
```

```
        33.91418
      ],
      [
        -84.37928,
        33.914
      ],
      [
        -84.37925,
        33.91384
      ],
      [
        -84.37925,
        33.91371
      ],
      [
        -84.37923,
        33.91363
      ],
      [
        -84.37922,
        33.91348
      ],
      [
        -84.37921,
        33.91323
      ],
      [
        -84.37913,
        33.91276
      ],
      [
        -84.37907,
        33.91231
      ],
      [
        -84.37905,
        33.91212
      ],
      [
        -84.379,
        33.91167
      ],
      [
        -84.37886,
        33.9116
      ],
      [
        -84.37871,
        33.91157
      ],
      [
        -84.37852,
        33.91157
      ],
```

```
            [
              -84.37777,
              33.91165
            ],
            [
              -84.37728,
              33.9117
            ],
            [
              -84.37643,
              33.91176
            ],
            [
              -84.3761,
              33.91178
            ],
            [
              -84.37571,
              33.91176
            ],
            [
              -84.37529,
              33.91171
            ],
            [
              -84.37417,
              33.91152
            ],
            [
              -84.37369,
              33.91144
            ],
            [
              -84.37343,
              33.91139
            ],
            [
              -84.37324,
              33.91136
            ],
            [
              -84.37313,
              33.91139
            ],
            [
              -84.37292,
              33.91144
            ],
            [
              -84.37153,
              33.91117
            ],
            [
              -84.37092,
```

```
          33.91107
        ],
        [
          -84.36998,
          33.91088
        ],
        [
          -84.36943,
          33.91078
        ],
        [
          -84.36906,
          33.91072
        ],
        [
          -84.36879,
          33.91067
        ],
        [
          -84.36782,
          33.91049
        ],
        [
          -84.3675,
          33.91043
        ],
        [
          -84.36641,
          33.91025
        ],
        [
          -84.36613,
          33.91019
        ],
        [
          -84.36534,
          33.91009
        ],
        [
          -84.3648,
          33.91004
        ],
        [
          -84.36428,
          33.90999
        ],
        [
          -84.36374,
          33.90998
        ],
        [
          -84.36349,
          33.90999
        ],
```

```
            [
              -84.36326,
              33.91001
            ],
            [
              -84.3631,
              33.90993
            ],
            [
              -84.363,
              33.90993
            ],
            [
              -84.36278,
              33.90993
            ],
            [
              -84.36248,
              33.90994
            ],
            [
              -84.36225,
              33.90996
            ],
            [
              -84.36181,
              33.90999
            ],
            [
              -84.36171,
              33.90999
            ],
            [
              -84.36162,
              33.91001
            ],
            [
              -84.36152,
              33.91001
            ],
            [
              -84.36094,
              33.91006
            ],
            [
              -84.36046,
              33.91007
            ],
            [
              -84.36024,
              33.91007
            ],
            [
              -84.36004,
```

```
        33.91004
      ],
      [
        -84.35985,
        33.91001
      ],
      [
        -84.35965,
        33.90994
      ],
      [
        -84.35949,
        33.90987
      ],
      [
        -84.35931,
        33.90979
      ],
      [
        -84.35917,
        33.90969
      ],
      [
        -84.35902,
        33.90959
      ],
      [
        -84.35888,
        33.90948
      ],
      [
        -84.35875,
        33.90936
      ],
      [
        -84.35862,
        33.90923
      ],
      [
        -84.3585,
        33.90909
      ],
      [
        -84.3584,
        33.90894
      ],
      [
        -84.3583,
        33.90879
      ],
      [
        -84.35821,
        33.90864
      ],
```

```
            [
              -84.35815,
              33.90845
            ],
            [
              -84.35811,
              33.90829
            ],
            [
              -84.35801,
              33.90758
            ],
            [
              -84.35801,
              33.90731
            ],
            [
              -84.35801,
              33.90721
            ],
            [
              -84.35804,
              33.90692
            ],
            [
              -84.3582,
              33.90634
            ],
            [
              -84.35837,
              33.90592
            ],
            [
              -84.35853,
              33.90552
            ],
            [
              -84.3586,
              33.90531
            ],
            [
              -84.35876,
              33.90493
            ],
            [
              -84.35901,
              33.90441
            ],
            [
              -84.35914,
              33.90417
            ],
            [
              -84.35968,
```

```
          33.9032
        ],
        [
          -84.35975,
          33.90306
        ],
        [
          -84.36027,
          33.9022
        ],
        [
          -84.3605,
          33.9018
        ],
        [
          -84.36052,
          33.90177
        ],
        [
          -84.36081,
          33.9013
        ],
        [
          -84.36122,
          33.90061
        ],
        [
          -84.36133,
          33.90037
        ],
        [
          -84.36168,
          33.89976
        ],
        [
          -84.36184,
          33.89944
        ],
        [
          -84.36229,
          33.89863
        ],
        [
          -84.36259,
          33.89807
        ],
        [
          -84.3631,
          33.89714
        ],
        [
          -84.36326,
          33.89685
        ],
```

```
            [
              -84.36332,
              33.89672
            ],
            [
              -84.36339,
              33.8966
            ],
            [
              -84.36342,
              33.89638
            ],
            [
              -84.36352,
              33.89619
            ],
            [
              -84.36358,
              33.89611
            ],
            [
              -84.36397,
              33.89543
            ],
            [
              -84.36422,
              33.89503
            ],
            [
              -84.36444,
              33.89469
            ],
            [
              -84.36448,
              33.89459
            ],
            [
              -84.36455,
              33.89445
            ],
            [
              -84.36457,
              33.8944
            ],
            [
              -84.36465,
              33.89419
            ],
            [
              -84.36476,
              33.89395
            ],
            [
              -84.36484,
```

```
          33.89369
        ],
        [
          -84.36493,
          33.89344
        ],
        [
          -84.365,
          33.89318
        ],
        [
          -84.36505,
          33.89292
        ],
        [
          -84.36507,
          33.89273
        ],
        [
          -84.36509,
          33.89266
        ],
        [
          -84.36516,
          33.89215
        ],
        [
          -84.36519,
          33.89161
        ],
        [
          -84.36519,
          33.89109
        ],
        [
          -84.36519,
          33.8908
        ],
        [
          -84.36518,
          33.88977
        ],
        [
          -84.36516,
          33.8885
        ],
        [
          -84.36513,
          33.88722
        ],
        [
          -84.36512,
          33.88563
        ],
```

```
            [
              -84.3651,
              33.88468
            ],
            [
              -84.36509,
              33.8842
            ],
            [
              -84.36509,
              33.88399
            ],
            [
              -84.36507,
              33.88254
            ],
            [
              -84.36507,
              33.8813
            ],
            [
              -84.36506,
              33.88016
            ],
            [
              -84.36507,
              33.87937
            ],
            [
              -84.36509,
              33.87882
            ],
            [
              -84.36509,
              33.87847
            ],
            [
              -84.36509,
              33.8781
            ],
            [
              -84.3651,
              33.8773
            ],
            [
              -84.36512,
              33.87648
            ],
            [
              -84.36507,
              33.87545
            ],
            [
              -84.36506,
```

```
          33.87501
        ],
        [
          -84.36506,
          33.87478
        ],
        [
          -84.36503,
          33.87435
        ],
        [
          -84.365,
          33.87371
        ],
        [
          -84.36497,
          33.87306
        ],
        [
          -84.36496,
          33.87266
        ],
        [
          -84.36497,
          33.87221
        ],
        [
          -84.36502,
          33.87197
        ],
        [
          -84.36506,
          33.87173
        ],
        [
          -84.36515,
          33.87145
        ],
        [
          -84.36522,
          33.87123
        ],
        [
          -84.36536,
          33.87081
        ],
        [
          -84.36547,
          33.87061
        ],
        [
          -84.36558,
          33.87037
        ],
```

```
            [
              -84.36586,
              33.86994
            ],
            [
              -84.36635,
              33.86933
            ],
            [
              -84.36654,
              33.86912
            ],
            [
              -84.36687,
              33.86875
            ],
            [
              -84.36741,
              33.86815
            ],
            [
              -84.36792,
              33.86754
            ],
            [
              -84.36809,
              33.86734
            ],
            [
              -84.36824,
              33.86711
            ],
            [
              -84.36838,
              33.8669
            ],
            [
              -84.3685,
              33.86667
            ],
            [
              -84.36863,
              33.86645
            ],
            [
              -84.36873,
              33.8662
            ],
            [
              -84.36882,
              33.86597
            ],
            [
              -84.36889,
```

```
          33.86574
        ],
        [
          -84.36895,
          33.86548
        ],
        [
          -84.36905,
          33.86499
        ],
        [
          -84.36912,
          33.86447
        ],
        [
          -84.36915,
          33.8641
        ],
        [
          -84.36925,
          33.86289
        ],
        [
          -84.36937,
          33.86156
        ],
        [
          -84.36947,
          33.86029
        ],
        [
          -84.3695,
          33.85995
        ],
        [
          -84.36954,
          33.85932
        ],
        [
          -84.36957,
          33.85893
        ],
        [
          -84.36966,
          33.85792
        ],
        [
          -84.36969,
          33.85755
        ],
        [
          -84.36969,
          33.85752
        ],
```

```
            [
              -84.36973,
              33.85713
            ],
            [
              -84.36976,
              33.85673
            ],
            [
              -84.3698,
              33.85618
            ],
            [
              -84.37002,
              33.85525
            ],
            [
              -84.37008,
              33.85483
            ],
            [
              -84.37017,
              33.8543
            ],
            [
              -84.37024,
              33.85388
            ],
            [
              -84.37027,
              33.8533
            ],
            [
              -84.37031,
              33.85279
            ],
            [
              -84.37031,
              33.8525
            ],
            [
              -84.37034,
              33.85216
            ],
            [
              -84.37037,
              33.852
            ],
            [
              -84.3704,
              33.85192
            ],
            [
              -84.37044,
```

```
        33.85188
      ],
      [
        -84.37056,
        33.85179
      ],
      [
        -84.37092,
        33.85168
      ],
      [
        -84.37121,
        33.85156
      ],
      [
        -84.37134,
        33.85149
      ],
      [
        -84.37149,
        33.85138
      ],
      [
        -84.37159,
        33.85131
      ],
      [
        -84.37169,
        33.85124
      ],
      [
        -84.37202,
        33.85092
      ],
      [
        -84.37233,
        33.85063
      ],
      [
        -84.37311,
        33.84989
      ],
      [
        -84.37313,
        33.84986
      ],
      [
        -84.37326,
        33.84975
      ],
      [
        -84.37347,
        33.84954
      ],
```

```
            [
              -84.37359,
              33.84943
            ],
            [
              -84.3739,
              33.8491
            ],
            [
              -84.37407,
              33.84886
            ],
            [
              -84.37419,
              33.8487
            ],
            [
              -84.3743,
              33.84854
            ],
            [
              -84.3741,
              33.84846
            ],
            [
              -84.37394,
              33.84836
            ],
            [
              -84.37388,
              33.84833
            ],
            [
              -84.37378,
              33.84825
            ],
            [
              -84.37369,
              33.84817
            ],
            [
              -84.37365,
              33.84812
            ],
            [
              -84.37362,
              33.84805
            ],
            [
              -84.37356,
              33.84799
            ],
            [
              -84.37345,
```

```
        33.84783
      ],
      [
        -84.37337,
        33.84772
      ],
      [
        -84.37334,
        33.84765
      ],
      [
        -84.3733,
        33.84759
      ],
      [
        -84.37329,
        33.84754
      ],
      [
        -84.37321,
        33.8474
      ],
      [
        -84.3731,
        33.84715
      ],
      [
        -84.373,
        33.84695
      ],
      [
        -84.37278,
        33.84651
      ],
      [
        -84.3726,
        33.84615
      ],
      [
        -84.3724,
        33.84569
      ],
      [
        -84.37233,
        33.84556
      ],
      [
        -84.37214,
        33.84514
      ],
      [
        -84.37204,
        33.84495
      ],
```

```
            [
              -84.37192,
              33.84474
            ],
            [
              -84.37188,
              33.84466
            ],
            [
              -84.37182,
              33.84456
            ],
            [
              -84.37166,
              33.84423
            ],
            [
              -84.37159,
              33.84411
            ],
            [
              -84.37141,
              33.84379
            ],
            [
              -84.37146,
              33.84376
            ],
            [
              -84.37188,
              33.84352
            ],
            [
              -84.37208,
              33.84339
            ],
            [
              -84.37224,
              33.84329
            ],
            [
              -84.37233,
              33.84323
            ],
            [
              -84.37249,
              33.84312
            ],
            [
              -84.37259,
              33.84304
            ],
            [
              -84.37285,
```

```
        33.84283
      ],
      [
        -84.37297,
        33.84275
      ],
      [
        -84.37304,
        33.8427
      ],
      [
        -84.37317,
        33.8426
      ],
      [
        -84.37329,
        33.84252
      ],
      [
        -84.37343,
        33.84241
      ],
      [
        -84.37359,
        33.84228
      ],
      [
        -84.37368,
        33.84223
      ],
      [
        -84.37414,
        33.84191
      ],
      [
        -84.37424,
        33.84186
      ],
      [
        -84.3744,
        33.84175
      ],
      [
        -84.37449,
        33.8417
      ],
      [
        -84.37468,
        33.84157
      ],
      [
        -84.37481,
        33.84149
      ],
```

```
            [
              -84.37507,
              33.84135
            ],
            [
              -84.37526,
              33.84123
            ],
            [
              -84.37533,
              33.8412
            ],
            [
              -84.37562,
              33.84106
            ],
            [
              -84.37603,
              33.84086
            ],
            [
              -84.37609,
              33.84083
            ],
            [
              -84.37616,
              33.8408
            ],
            [
              -84.37646,
              33.84066
            ],
            [
              -84.37684,
              33.84051
            ],
            [
              -84.377,
              33.84043
            ],
            [
              -84.3772,
              33.84038
            ],
            [
              -84.37736,
              33.84035
            ],
            [
              -84.37742,
              33.84035
            ],
            [
              -84.37755,
```

```
          33.84033
        ],
        [
          -84.37771,
          33.84032
        ],
        [
          -84.37793,
          33.84032
        ],
        [
          -84.37809,
          33.84032
        ],
        [
          -84.37828,
          33.8403
        ],
        [
          -84.37842,
          33.84025
        ],
        [
          -84.37861,
          33.84022
        ],
        [
          -84.37878,
          33.84011
        ],
        [
          -84.37899,
          33.84001
        ],
        [
          -84.37912,
          33.83993
        ],
        [
          -84.37921,
          33.83988
        ],
        [
          -84.37947,
          33.8397
        ],
        [
          -84.37987,
          33.83941
        ],
        [
          -84.38009,
          33.83946
        ],
```

```
            [
              -84.38039,
              33.83954
            ],
            [
              -84.38063,
              33.83963
            ],
            [
              -84.38074,
              33.83969
            ],
            [
              -84.38082,
              33.83972
            ],
            [
              -84.38111,
              33.83983
            ],
            [
              -84.38116,
              33.83987
            ],
            [
              -84.38131,
              33.83995
            ],
            [
              -84.38141,
              33.84001
            ],
            [
              -84.38169,
              33.84014
            ],
            [
              -84.38176,
              33.84017
            ],
            [
              -84.38192,
              33.84025
            ],
            [
              -84.38183,
              33.84043
            ],
            [
              -84.38199,
              33.84051
            ],
            [
              -84.38196,
```

```
                33.84063
              ]
            ]
          ],
          "type": "LineString"
        },
        "properties": {},
        "type": "Feature"
      }
    ],
    "type": "FeatureCollection"
  },
  "waypoints": [
    {
      "maneuvers": [],
      "departuretime": 1619794087,
      "start": 1619794087,
      "arrivaltime": 1619794087,
      "idletime": 0,
      "totaltraveltime": 945,
      "nontraffictime": 798,
      "traffictime": 147,
      "distance": 9274,
      "duration": 0,
      "lng": -84.37975,
      "lat": 33.91776,
      "id": "groceryNorth",
      "success": true
    },
    {
      "maneuvers": [],
      "departuretime": 1619795032,
      "start": 1619795032,
      "arrivaltime": 1619795032,
      "idletime": 0,
      "totaltraveltime": 0,
      "nontraffictime": 0,
      "traffictime": 0,
      "distance": 0,
      "duration": 0,
      "lng": -84.38192,
      "lat": 33.84063,
      "id": "grocerySouth",
      "success": true
    }
  ],
  "summary": {
    "totaltraveltime": 945,
    "nontraffictime": 798,
    "traffictime": 147,
    "distance": 9274
  },
  "boundingbox": {
    "southwest": {
```

```
        "lng": -84.38192,
        "lat": 33.84063
      },
      "northeast": {
        "lng": -84.37975,
        "lat": 33.91776
      }
    },
    "status": "Completed",
    "jobid": "mare-routing-11-1619794088.349843-505"
  },
  "success": true
}
```

Get the Boundary Information

The `GetBoundaryInformation()` Apex method returns geographical IDs of locations that fit the specified search criteria.

Salesforce Maps works with data providers that supply property data. Periodically, these providers update their data, which can affect custom code. If the providers change their data formats, update your custom code accordingly. Salesforce Maps can update, replace, or remove property data at any time.

Signature

```
Map<String, Object> maps.API.GetBoundaryInformation(String parameters)
```

Where,

- `maps` is the namespace of the Salesforce Maps package. It's automatically available when the package is installed.
- `API` is the class that contains the global methods exposed to developers.
- `GetBoundaryInformation()` is the method.

Sample Code

This code finds the geographical IDs of states in the USA beginning with "A".

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Example

```
 // Create the search criteria in JSON format.
String parameters =
```

```
'{"overlay":"USA","level":"1","filters":[{"field_id":"label","operator":"starts
with","values":["A"]}]}';

// Call the GetBoundaryInformation() method with the search criteria.
Map<String, Object> response= maps.API.GetBoundaryInformation(parameters);

// Log the resulting Geo IDs.
system.debug(response);
```

Sample Response

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

```
{
  "endPoint": "https://internal.na.sfmapsapi.com/boundary/search/1",
  "limitInfo": {
    "QueryRows": "12 / 50000",
    "Queries": "8 / 100",
    "HeapSize": "71096 / 6000000",
    "CPUTime": "133 / 10000"
  },
  "request":
"{\"overlay\":\"USA\",\"level\":\"1\",\"filters\":[{\"field_id\":\"label\",\"operator\":\"starts
 with\",\"values\":[\"A\"]}]}",
  "params": {
    "filters": [
      {
        "values": [
          "A"
        ],
        "operator": "starts with",
        "field_id": "label"
      }
    ],
    "level": "1",
    "overlay": "USA"
  },
  "success": true,
  "data": {
    "geoids": [
      {
        "label": "Alabama",
        "value": "USA-01"
      },
      {
        "label": "Alaska",
        "value": "USA-02"
      },
      {
        "label": "Arizona",
        "value": "USA-04"
      },
```

```
      {
        "label": "Arkansas",
        "value": "USA-05"
      }
    ]
  }
}
```

Get the Distance Matrix

The `GetDistanceMatrix()` Apex method takes a set of locations and returns the estimated travel distance in meters and the time in seconds between the locations.

Example: With two locations A and B, you can use this method to return matrices of travel distance and times between A–B and B–A. The diagonal elements of the matrices A–A and B–B are zero because the travel distance and the time from one spot to the same spot is zero.

The values for A–B and B–A can differ because the returning path isn't always the same. The travel times are calculated for eight predefined time windows based on historical traffic data. Each time window has its own travel time matrix.

Signature

```
Map<String, Object> maps.API.GetDistanceMatrix(List<Map<String, Object>>)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `GetDistanceMatrix()` is the method.

Allocations

Include as many as 20 locations. To catch exceptions, put a try-catch block around your code.

Sample Code

This code finds the estimated travel distance in meters and time between two grocery stores.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

```
// Create a list of locations.
List<Map<String, Object>> locations =
 new List<Map<String, Object>>();

// Create and add location 1.
Map<String, Object> groceryNorth =
 new Map<String, Object> {
 'location_id' => 'groceryNorth',
 'latitude' => 33.917760,
 'longitude' => -84.379750
};
locations.add(groceryNorth);

// Create and add location 2.
Map<String, Object> grocerySouth =
 new Map<String, Object> {
 'location_id' => 'grocerySouth',
 'latitude' => 33.840630,
 'longitude' => -84.381920
};
locations.add(grocerySouth);

// Call the GetDistanceMatrix() method with the list of locations.
Map<String, Object> matrix = maps.API.GetDistanceMatrix(locations);

// Log the resulting distance matrix.
system.debug(matrix);
```

Sample Response

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data that you receive in the resulting map.

```
{
  "endPoint": "https://internal.na.sfmapsapi.com/core/routing/matrix/1",
  "limitInfo": {
    "QueryRows": "12 / 50000",
    "Queries": "8 / 100",
    "HeapSize": "73745 / 6000000",
    "CPUTime": "150 / 10000"
  },
  "request":
"{\"vehicle\":{\"type\":\"car\"},\"locations\":[{\"longitude\":-84.3795,\"latitude\":33.9776,\"location_id\":\"groceryNorth\"},{\"longitude\":-84.3819,\"latitude\":33.8406,\"location_id\":\"grocerySouth\"}]}",

  "params": {
    "locations": [
      {
        "location_id": "groceryNorth",
        "latitude": 33.91776,
        "longitude": -84.37975
```

```
      },
      {
        "location_id": "grocerySouth",
        "latitude": 33.84063,
        "longitude": -84.38192
      }
    ],
    "vehicle": {
      "type": "car"
    }
  },
  "success": true,
  "data": {
    "Solution": {
      "travel_costs": {
        "grocerySouth": {
          "grocerySouth": [ // The values of the matrix diagonals are zero because the
distance and time between grocerySouth and grocerySouth are zero.
            0, // Matrix 0 index 0: Travel distance in meters
            0, // Matrix 1 index 0: Travel time for time window 0
            0, // Matrix 2 index 0: Travel time for time window 1
            0, // And so forth...
            0,
            0,
            0,
            0,
            0
          ],
          "groceryNorth": [ // The values for grocerySouth to groceryNorth.
            9210,  // Matrix 0 index 1: Travel distance in meters
            794.6, // Matrix 1 index 1: Travel time for time window 0
            810.5, // Matrix 2 index 1: Travel time for time window 1
            890, // And so forth...
            920.7,
            965.7,
            1047.2,
            1048.8,
            952.5
          ]
        },
        "groceryNorth": {
          "grocerySouth": [ // The values for groceryNorth to grocerySouth.
            9274.1, // Matrix 0 index 2: Travel distance in meters
            797.9, // Matrix 1 index 2: Travel time for time window 0
            808.3, // Matrix 2 index 2: Travel time for time window 1
            879, // And so forth...
            905.3,
            944.2,
            974.6,
            963.5,
            920.5
          ],
          "groceryNorth": [  // The values for the matrix diagonals are zero.
            0, // Matrix 0 index 3: Distance in meters
```

```
          0, // Matrix 1 index 3: Time for time window 0
          0, // Matrix 2 index 3: Time for time window 1
          0, // And so forth...
          0,
          0,
          0,
          0,
          0
        ]
      }
    },
    "traffic_windows": [
      { // Time window for matrix 1: midnight to 6:30 am and 7 pm to midnight.
        "traffic_window_start_times": [
          "00:00:00",
          "19:00:00"
        ],
        "traffic_window_index": 0,
        "traffic_window_end_times": [
          "06:30:00",
          "00:00:00"
        ]
      },
      { // Time window for matrix 2: 6:30 am to 7:30 am.
        "traffic_window_start_times": [
          "06:30:00"
        ],
        "traffic_window_index": 1,
        "traffic_window_end_times": [
          "07:30:00"
        ]
      },
      { // And so forth...
        "traffic_window_start_times": [
          "07:30:00"
        ],
        "traffic_window_index": 2,
        "traffic_window_end_times": [
          "08:30:00"
        ]
      },
      {
        "traffic_window_start_times": [
          "08:30:00"
        ],
        "traffic_window_index": 3,
        "traffic_window_end_times": [
          "09:30:00"
        ]
      },
      {
        "traffic_window_start_times": [
          "09:30:00"
        ],
```

```
          "traffic_window_index": 4,
          "traffic_window_end_times": [
            "16:00:00"
          ]
        },
        {
          "traffic_window_start_times": [
            "16:00:00"
          ],
          "traffic_window_index": 5,
          "traffic_window_end_times": [
            "17:00:00"
          ]
        },
        {
          "traffic_window_start_times": [
            "17:00:00"
          ],
          "traffic_window_index": 6,
          "traffic_window_end_times": [
            "18:00:00"
          ]
        },
        {
          "traffic_window_start_times": [
            "18:00:00"
          ],
          "traffic_window_index": 7,
          "traffic_window_end_times": [
            "19:00:00"
          ]
        }
      ],
      "status": "Ok",
      "locations": [
        {
          "time_zone": "America/New_York", // Time zone of the location.
          "location_id": "groceryNorth"
        },
        {
          "time_zone": "America/New_York",
          "location_id": "grocerySouth"
        }
      ]
    },
    "JobStatus": "Completed",
    "JobMessage": {
      "comment": "MARE job succeeded"
    },
    "JobId": "mare-matrix-11-1619788548.552854-151"
  }
}
```

Get the Geographical Coordinates of an Address

The `Geocode()` Apex method takes a single address and returns the geographical coordinates and the formatted address.

To get the geographical coordinates of more than one address, use the `BatchGeocode()` on page 48 Apex method.

Signature

```
Map<String, Object> maps.API.Geocode(Map<String, Object> options)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `Geocode()` is the method.

Sample Code

This code returns the geographical coordinates of the Salesforce headquarters. The output also returns the formatted address and adds missing information, such as postal code and country, if available.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Input Format of the Address

```
String address = '{HouseNumber} {Street}, {City}, {State} {PostalCode} {Country}';
String address = '{HouseNumber} {Street}, {City}, {State}';
```

Example

```
// Create and add an address.
String salesforceAddress = '415 Mission Street, San Francisco, CA 94105 USA';

Map<string,object> options = new Map<String,Object> {
            'version' => '1', // Required. Version of the API endpoint. Must be '1'.
            'address' => salesforceAddress
        };

// Call the Geocode() method with the address.
Map<String, Object> response = maps.API.Geocode(options);

// Log the resulting geographical coordinates and formatted address.
system.debug(response);
```

Sample Response

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

If you invoke this method within a flow, process builder, or trigger and want to use the data from the JSON response, implement logic to retrieve that data. For example, you want to save the latitude and longitude coordinates from the JSON response to your records.

```
{
  "baseUrl":
"https://internal.na.sfmapsapi.com/core/geocoding/2?address=415+Mission+Street%2C+San+Francisco%2C+CA+94105+USA",

  "data": {
    "houseNumber": "415",
    "matchLevel": "Address",
    "score": 100,
    "country": "USA",
    "postal": "94105",
    "state": "CA",
    "city": "San Francisco",
    "street": "Mission St",
    "fullAddress": "415 Mission St, San Francisco, CA 94105, United States",
    "position": {
      "lng": -122.397,
      "lat": 37.78977
    }
  },
  "source": "http",
  "success": true
}
```

Sample Response If Not Geocoded

If an address can't be geocoded, you receive this response. Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

```
{
  "success": false,
  "message": "No results for that particular address."
}
```

Get the Geographical Coordinates of Addresses in Batch

The `BatchGeocode()` Apex method takes addresses and returns their geographical coordinates and formatted addresses.

Signature

```
Map<String, Object> maps.API.BatchGeocode(Map<String, Object>)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.

- `API` is the class that contains the global methods exposed to developers.
- `BatchGeocode()` is the method.

API Call Allocations

- For geocoding, submit up to 120 requests per minute.
- Send batches of up to 50 addresses per request.

Sample Code

This code returns the geographical coordinates of the White House and Washington Monument. The output also returns the formatted address and adds missing information, such as postal code and country, if available.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Input Format of an Address

```
String address = '{HouseNumber} {Street}, {City}, {State} {PostalCode} {Country}';
String address = '{HouseNumber} {Street}, {City}, {State}';
```

Example

```
public with sharing class exampleBatchGeocode {
    public exampleBatchGeocode() {}

    public exampleBatchGeocodeWithSFMaps(){
        List<Account> accountList = [SELECT Id, Name, BillingAddress FROM Account WHERE
Name like 'W%' AND BillingLongitude = Null];
       Map<String, Account> accounts = new Map<String, Account>(); // Map of the Accounts
 to link the results back to
      Map<String, Map<String, String>> addresses = new Map<String, Map<String, String>>();

        for (Account acc : accounts) {
            // It is recommended to prevent sending null fields. Not done here for
illustrative purposes.
            Map<String, String> address = new Map<String, String>{
                'address' => acc.BillingAddress.getStreet() + ', ' +
acc.BillingAddress.getCity() + ', ' + acc.BillingAddress.getPostalCode() + ', ' +
acc.BillingAddress.getCountry()
            };
            String idstr = Id.valueOf(acc.id);
            addresses.put(idstr, address);
            accounts.put(idstr.toLowerCase(), acc);// Ids will be lowercase in results.
            // Illustrative Sample: Three accounts in list.
```

```
            // [{
            //     id: '005B0000005LDrUIA1',
            //     name: 'White House',
           //     address: '1600 Pennsylvania Ave NW, Washington, DC 20500, United States'

            // }, {
            //     id: '005B0000005LDrUIA2',
            //     name: 'Washington Monument'
            //     address: '2 15th St NW, Washington, null, United States'
            // }, {
            //     id: '005B0000005LDrUIA3',
            //     name: 'Wash and Ride',
            //     address: 'null, null, null, United States'
            // }]
        }

        Map<String, Object> response; // Deserialized Untyped Response
        try {
            response = maps.API.BatchGeocode(new Map<String, Object> {
              'version' => '2', // Required. Hardcode value to 2. Has no impact on results
 or process.
                'address_info' => JSON.serialize(addresses)
            });
        }
        catch (Exception ex) {
            // Handle exception. Simple log for illustrative purposes.
            System.debug(ex);
            return;
        }
        List<Account> accountsWithBadData = new List<Account>();
        Boolean callSuccess = (Boolean)response.get('success');
        if(callSuccess != null){
            Map<String, Object> results = (Map<String, Object>)response.get('results');
            for(String accountId: results.keySet()){
                Account acc = accounts.get(accountId);
                Map<String, Object> record = (Map<String, Object>)results.get(accountId);

                if((Boolean) record.get('success')){
                    Map<String, Object> data = (Map<String, Object>)record.get('data');
                    Map<String, Object> position = (Map<String, Object>)
data.get('position');
                    acc.BillingLatitude = (Decimal) position.get('lat');
                    acc.BillingLongitude = (Decimal) position.get('lng');
                    if((Integer) data.get('score') < 50) {
                        // Low confidence in address match. Marking the accounts needed
for review.
                        accountsWithBadData.add(acc);
                    }
                }
                else {
                    // Unable to match address provided.
                    accountsWithBadData.add(acc);
                }
            }
```

```
            update accountList; // Update records with Latitude and Longitude.
            reviewBadAccounts(accountsWithBadData); // Pushing accounts with bad data to
another process to be handled.
        }
        else {
            // Three primary types of failed requests.
            String errorCode = (String) response.get('error_code');
            if(errorCode == 'GC-0500'){
                // Internal Server Error
                // Retry the request at a later time and/or reach out to support.
                return;
            }
            else if(errorCode == 'GC-0429'){
                // Too many requests. Perform an exponential back off for the retries.
                // Bad requests count against limits. Be sure to not waste calls on bad
addresses and improperly created requests.
                return;
            }
            else if(errorCode.startsWith('GC-1')){
                System.debug(response);
                 // Improperly created requests. Fix errors before trying again.
                return;
            }
            else {
                // Fail safe catch for future error codes.
                System.debug(response);
                return;
            }
        }
    }

}
```

Sample Response

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

If you invoke this method within a flow, process builder, or trigger and want to use the data from the JSON response, implement logic to retrieve that data. For example, you want to save the latitude and longitude coordinates from the JSON response to your records.

Responses include geocoding confidence scores based on address accuracy and completeness. For example, addresses missing city and state information result in scores lower than 100. Success results can include:

- False, which indicates no match.
- Partial, which indicates scores less than 100.
- Full, which indicates scores of 100.

```
{
    "baseUrl":
"https://sfmapsgateway-uengage1.sfdc-lywfpd.svc.sfdcfc.net/core/batchgeocode/3",
    "results": {
        "001ru00000ca1teiad": {
            "message": "Invalid address",
```

```
            "success": false
        },
        "001ru00000ca1tdiad": {
            "success": true,
            "source": "http",
            "data": {
                "city": "Washington",
                "country": "USA",
                "fullAddress": "2 15th St NW, Washington, DC 20004, United States",
                "houseNumber": "2",
                "matchLevel": "Address",
                "position": {
                    "lat": 38.889043,
                    "lng": -77.0330958
                },
                "postal": "20004",
                "score": 86,
                "state": "DC",
                "street": "15th St NW",
            }
        },
        "001ru00000ca1tciad": {
            "data": {
                "city": "Washington",
                "country": "USA",
                "fullAddress": "1600 Pennsylvania Ave NW, Washington, DC 20500, United
States",
                "houseNumber": 1600,
                "matchLevel": "Address",
                "position": {
                    "lat": 38.89768,
                    "lng": -77.03655
                },
                "postal": "20500",
                "score": 100,
                "state": "DC",
                "street": "Pennsylvania Ave NW"
            }
        }
    },
    "success": true
}
```


Sample Error Response

If you submit improperly created requests, expect a response similar to this.


```
{
    "api_info": {
        "method": "POST",
        "request_id": "fcad404b-ea52-429e-8989-40df1daccf4b",
        "uri": "/core/batchgeocode/3"
    },
    "baseUrl":
```

```
"https://sfmapsgateway-uengage1.sfdc-lywfpd.svc.sfdcfc.net/core/batchgeocode/3",
    "error": {
        "data": {
            "minimum": 1,
            "property": "root map"
        },
        "message": "Property mimimum not met"
    },
    "error_code": "GC-1011"
}
```

Permanently Delete Datasets and Alignments

The `DoCleanup()` Apex method permanently removes, or cleans up, datasets and related alignments that a user already deleted using **Territory Planning Data Sets** or **Territory Planning Alignments** > **Delete** in the UI. Use this method to reduce object size and improve Territory Planning load time and performance.

The `DoCleanup()` Apex method is the equivalent of clicking **Permanently Delete** in Territory Planning Data Management in Setup. You can permanently delete datasets only with a Deleted status as shown on the Territory Planning Data Sets page. When you permanently delete a dataset, all alignments created from that dataset's source are also permanently removed.

Dataset and alignment cleanups can run at scheduled times or after a user deletes a dataset in the UI.

Signature

```
AsyncApexJob maps.TPSDK.DoCleanup()
```

Where,

- `AsyncApexJob` represents an individual Apex sharing recalculation job, a batch Apex job, a method with the `future` annotation, or a job that implements `Queueable`. Use this object to query Apex batch jobs in your organization.
- `maps` is the namespace that's available after you install Salesforce Maps.
- `TPSDK` is the class that contains the Salesforce Maps Territory Planning global methods exposed to developers.
- `DoCleanup()` is the method.

Sample Code

This code runs dataset and alignment cleanups at scheduled times. The `DoCleanup()` method returns an `AsyncApexJob` containing the number of items processed, the date and time of job completion, and other details.

Warning: If you invoke methods within a flow, process builder, or trigger, do one of the following to avoid uncommitted work errors:

- Call the methods through a future method
- Call the methods as queueable

Use the sample code in this method documentation as a guide.

Example

```
// Implement as Schedulable to run the Apex class at regular intervals.
public with sharing class TerritoryPlanningCleanupSchedule implements Schedulable  {
```

```
// Invoke the scheduled Apex class with the execute() method.
    public void execute(SchedulableContext SC) {
        AsyncApexJob cleanupJob = maps.TPSDK.DoCleanup();
        System.debug('Cleanup job is ' +cleanupJob.Status+ '.  '
+cleanupJob.JobItemsProcessed+ ' of ' +cleanupJob.TotalJobItems+ ' batches processed.');
    }

}
```

SEE ALSO:

Knowledge Article: Data Set and Alignment Browser Window

SOAP API Developer Guide: AsyncApexJob

Apex Developer Guide: Apex Scheduler

Retrieve Data That Salesforce Maps Hosts

The `GetHostedData()` Apex method retrieves geographical IDs for properties and displays them in Salesforce Maps. We accomplish this retrieval in two requests while using the same method. The first request returns geographical IDs for properties that Salesforce Maps hosts for a specified area and filter criteria. The second request takes these geographical IDs and display parameters, such as legend and popup information, and displays them in Salesforce Maps.

Salesforce Maps works with data providers that supply property and business data. Periodically, these providers update their data, which can affect custom code. If the providers change their data formats, update your custom code accordingly. Salesforce Maps can update, replace, or remove property data at any time.

Signature

```
Map<String, Object> maps.API.GetHostedData(String parameters)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `GetHostedData()` is the method.

Sample Code to Retrieve Geographical IDs

The first example request using the `GetHostedData()` method returns geographical IDs for women-owned businesses that Salesforce Maps host. Those businesses produce revenue between $100,000 and $1,000,000 in a specified area.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Input Format of the Parameters

```
{
  "mapinfo": { // Boundary area.
    "offset": integer, // Pagination when the number of results is large.
    "limit": integer, // Maximum number of geographical IDs to return in response.
    "celng": decimal, // Longitude coordinate to search closest records.
    "celat": decimal, // Latitude coordinate to search closest records.
    "swlng": decimal, // South-Western longitude coordinate for map boundary.
    "swlat": decimal, // South-Western latitude coordinate for map boundary.
    "nelng": decimal, // North-Eastern longitude coordinate for map boundary.
    "nelat": decimal  // North-Eastern latitude coordinate for map boundary.
  },
  "data": {
    "filters": [{}], // Filters to limit the results.
    "defaultMarkerColor": "String:String", // Hex color : Type of marker to be returned
    "level_id": "String", // Sub-section of the data specified by file_id.
    "file_id": "String" // Data being queried, for example, business, property, political
 boundaries, etc.
  }
}
```

Example

```
// Create the boundary area.
Map<String,Decimal> mapInfo = new Map<String,Decimal>{
    'nelat' => 49.92293,
    'nelng' => -55.72265,
    'swlat' => 29.49698,
    'swlng' => -140.00976,
    'celat' => 40.48038,
    'celng' => -97.86621,
    'limit' => 6,
    'offset' => 0
};

// Create the filters.
Map<String,Object> femaleOwned = new Map<String,Object>{
    'topic_id' => 'female_owned',
    'operator' => 'equals',
    'values' => new List<String>{'Yes'}
};
Map<String,Object> locationSalesTotal = new Map<String,Object>{
    'topic_id' => 'location_sales_total',
    'operator' => 'range',
    'min' => '100000',
    'max' => '1000000'
};

List<Map<String,Object>> filters = new List<Map<String,Object>>();
filters.add(femaleOwned);
filters.add(locationSalesTotal);
```

```
// Add the filters and additional 'data' parameters.
Map<String,Object> dataMap = new Map<String,Object>{
    'filters' => filters,
    'defaultMarkerColor' => '93c47d:Circle',
    'level_id' => '42',
    'file_id' => 'business'
};

// Build the request.
Map<String,Object> request = new Map<String,Object>();
request.put('data', dataMap);
request.put('mapinfo', mapInfo);
String requestString = JSON.serialize(request);

// Call the GetHostedData() method with the boundary area and filter criteria.
Map<String, Object> response = maps.API.GetHostedData(requestString);

// Log the resulting locations.
System.debug(JSON.serialize(response));
```

Sample Response of Geographical IDs

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

```
{
   "endPoint":"https://internal.na.sfmapsapi.com/data/markers/2",
   "limitInfo":{
      "QueryRows":"12 / 50000",
      "Queries":"8 / 100",
      "HeapSize":"70683 / 6000000",
      "CPUTime":"171 / 10000"
   },

[illegible]

   "params":{
      "data":{
         "file_id":"business",
         "level_id":"42",
         "defaultMarkerColor":"93c47d:Circle",
         "filters":[
            {
               "topic_id":"female_owned",
               "operator":"equals",
               "values":[
                  "Yes"
               ]
            },
            {
               "topic_id":"location_sales_total",
               "operator":"range",
```

```
                "min":"100000",
                "max":"1000000"
             }
          ]
       },
       "mapinfo":{
          "nelat":49.92293,
          "nelng":-55.72265,
          "swlat":29.49698,
          "swlng":-140.00976,
          "celat":40.48038,
          "celng":-97.86621,
          "limit":6,
          "offset":0
       }
    },
    "success":true,
    "data":{ // List of geographical IDs.
       "ids":[
          "11140017125231",
          "11140017127280",
          "11140017102994",
          "11140017127285",
          "11132553548894",
          "11140017127286"
       ]
    }
}
```

Sample Code to Display Geographical IDs

The second request using the `GetHostedData()` method takes the geographical IDs and display parameters, such as legend and popup information, and displays them in Salesforce Maps.

Input Format of the Parameters

```
{
  "mapinfo": {
    "offset": integer, // Pagination when the number of results is large.
    "limit": integer, // Maximum number of geographical IDs to return in response.
    "celng": decimal, // Longitude coordinate to search closest records.
    "celat": decimal, // Latitude coordinate to search closest records.
    "swlng": decimal, // South-Western longitude coordinate for map boundary.
    "swlat": decimal, // South-Western latitude coordinate for map boundary.
    "nelng": decimal, // North-Eastern longitude coordinate for map boundary.
    "nelat": decimal  // North-Eastern latitude coordinate for map boundary.
  },
  "data": {
    "popup": { // Popup information for the displayed locations.
      "tabs": [{}],// Array of tab objects to be displayed in the popup.
      "header": [{}] // Array of objects containing information of the tab headers.
    },
    "legend": { // Map legend.
      "rows": [{}], // Array of objects containing information of each row in the legend.
```

```
      "subTitle": "String", // Subtitle of the legend.
      "title": "String" // Title of the the legend.
    },
    "defaultMarkerColor": "String:String", // Hex color : Type of marker to be returned.
    "level_id": "String", // Sub-section of the data specified by file_id.
    "file_id": "String" // Data being queried, for example, business, property, political
 boundaries, etc.
  },
  "aggregates": Boolean, // Aggregated data of all the records returned. It contains the
min, max, sum, and averages of the records.
  "details": Boolean, // Return the details for the requested records.
  "ids": [] // Array of geographical IDs.
}
```

Example

```
// Create the map legend.
 Map<String,Object> row1 = new Map<String,Object>
{
    'topic_id' => 'female_owned',
    'operator' => 'equals',
    'values' => new List<String>{'Yes'},
    'color' => '38d84a:Marker'
};

Map<String,Object> row2 = new Map<String,Object>
{
    'topic_id' => 'location_sales_total',
    'operator' => 'range',
    'min' => '10',
    'max' => '1000000',
    'color' => '38d87c:Marker'
};

List<Map<String,Object>> legendRows = new List<Map<String,Object>>();
legendRows.add(row1);
legendRows.add(row2);

Map<String,Object> legend = new Map<String,Object>{
    'title' => 'Total Company Sales',
    'subTitle' => 'Breakdown of businesses based on company sales',
    'rows' => legendRows
};

// Create the popup header.
List<Map<String,String>> header = new List<Map<String,String>>();
header.add(new map<String,String>{
        'file_id' => 'business',
        'topic_id' => 'location_sales_total'
    }
);

// Create the popup tabs.
List<Map<String,String>> tabData = new List<Map<String,String>>();
```

```
tabData.add(new map<String,String>{'file_id' => 'business', 'topic_id' =>
'location_sales_total'});

Map<String,Object> tab = new map<String,Object>{
    'tab_id' => '1520003558209',
    'tab_label' => 'Additional info',
    'data' => tabData
};

List<Map<String,Object>> tabs = new List<Map<String,Object>>();
tabs.add(tab);

// Build the popup object.
Map<String,Object> popup = new Map<String,Object>{
    'header' => header,
    'tabs' => tabs
};

// Build the 'data' object.
Map<String,Object> data = new Map<String,Object>{
    'file_id' => 'business',
    'level_id' => '42',
    'defaultMarkerColor' => '93c47d:Circle',
    'legend' => legend,
    'popup' => popup
};

// Create the boundary area.
Map<String,Decimal> mapInfo = new Map<String,Decimal>{
    'nelat' => 49.92293,
    'nelng' => -55.72265,
    'swlat' => 29.49698,
    'swlng' => -140.00976,
    'celat' => 40.48038,
    'celng' => -97.86621,
    'limit' => 6,
    'offset' => 0
};

// Create the array of geographical IDs.
List<String> returnedIds = new
List<String>{'11140017125231','11140017127280','11140017102994','11140017127285','11132553548894','11140017127286'};

// Build the request.
Map<String,Object> request = new Map<String,Object>{
    'ids' => returnedIds,
    'details' => true, // Return the details of the requested records.
    'aggregates' => true, // Get the aggregated record data.
    'data' => data,
    'mapinfo' => mapInfo
};

String requestString = JSON.serialize(request);
```

```
// Call the GetHostedData() method with the geographical IDs and display parameters.
Map<String, Object> response = maps.API.GetHostedData(requestString);

//Log the results.
System.debug(JSON.serialize(response));
```

Sample Response of Displaying the Geographical IDs

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.


```
{
  "endPoint": "https://internal.na.sfmapsapi.com/data/markers/2",
  "limitInfo": {
    "QueryRows": "12 / 50000",
    "Queries": "8 / 100",
    "HeapSize": "79714 / 6000000",
    "CPUTime": "129 / 10000"
  },
  "request":
[illegible]

[illegible]
 of businesses based on company sales\",\"title\":\"Total Company
[illegible]

  "params": {
    "ids": [
      "11140017125231",
      "11140017127280",
      "11140017102994",
      "11140017127285",
      "11132553548894",
      "11140017127286"
    ],
    "details": true,
    "aggregates": true,
    "data": {
      "file_id": "business",
      "level_id": "42",
      "defaultMarkerColor": "93c47d:Circle",
      "legend": {
        "title": "Total Company Sales",
        "subTitle": "Breakdown of businesses based on company sales",
        "rows": [
          {
            "topic_id": "female_owned",
            "operator": "equals",
            "values": [
              "Yes"
            ],
            "color": "38d84a:Marker"
          },
```

```
          {
            "topic_id": "location_sales_total",
            "operator": "range",
            "min": "10",
            "max": "1000000",
            "color": "38d87c:Marker"
          }
        ]
      },
      "popup": {
        "header": [
          {
            "file_id": "business",
            "topic_id": "location_sales_total"
          }
        ],
        "tabs": [
          {
            "tab_id": "1520003558209",
            "tab_label": "Additional info",
            "data": [
              {
                "file_id": "business",
                "topic_id": "location_sales_total"
              }
            ]
          }
        ]
      }
    },
    "mapinfo": {
      "nelat": 49.92293,
      "nelng": -55.72265,
      "swlat": 29.49698,
      "swlng": -140.00976,
      "celat": 40.48038,
      "celng": -97.86621,
      "limit": 6,
      "offset": 0
    }
  },
  "success": true,
  "data": {
    "aggregates": [
      {
        "avg": 329333.3333333333,
        "sum": 1976000,
        "max": 461000,
        "min": 303000,
        "label": "Sales: Total (USD)"
      }
    ],
    "legend": {
      "rows": [
```

```
      {
        "values": [
          "Yes"
        ],
        "row_id": "row-0",
        "operator": "equals",
        "color": "38d84a:Marker",
        "topic_id": "female_owned"
      },
      {
        "max": "1000000",
        "min": "10",
        "row_id": "row-1",
        "operator": "range",
        "color": "38d87c:Marker",
        "topic_id": "location_sales_total"
      },
      {
        "values": [
          "--Other--"
        ],
        "row_id": "row-other",
        "operator": "equals",
        "color": "93c47d:Circle",
        "topic_id": "female_owned"
      }
    ],
    "subTitle": "Breakdown of businesses based on company sales",
    "title": "Total Company Sales"
  },
  "markers": [
    {
      "popup": {
        "tabs": [
          {
            "data": [
              {
                "formatted_value": "$303,000.00",
                "value": "303000",
                "topic_id": "location_sales_total",
                "label": "Sales: Total (USD)"
              }
            ],
            "tab_id": "1520003558209",
            "tab_label": "Additional info"
          }
        ],
        "header": [
          {
            "formatted_value": "$303,000.00",
            "value": "303000",
            "topic_id": "location_sales_total",
            "label": "Sales: Total (USD)"
          }
```

```
      ]
    },
    "position": {
      "lng": "-97.886011",
      "lat": "40.441185"
    },
    "color": "38d84a:Marker",
    "rowid": "row-0",
    "datatype": "business",
    "c2c": true,
    "uid": "11140017125231",
    "geoid": "11140017125231",
    "label": "Urbauer Family Farm"
  },
  {
    "popup": {
      "tabs": [
        {
          "data": [
            {
              "formatted_value": "$303,000.00",
              "value": "303000",
              "topic_id": "location_sales_total",
              "label": "Sales: Total (USD)"
            }
          ],
          "tab_id": "1520003558209",
          "tab_label": "Additional info"
        }
      ],
      "header": [
        {
          "formatted_value": "$303,000.00",
          "value": "303000",
          "topic_id": "location_sales_total",
          "label": "Sales: Total (USD)"
        }
      ]
    },
    "position": {
      "lng": "-97.800868",
      "lat": "40.466693"
    },
    "color": "38d84a:Marker",
    "rowid": "row-0",
    "datatype": "business",
    "c2c": true,
    "uid": "11140017102994",
    "geoid": "11140017102994",
    "label": "Rauscher Family Farm"
  },
  {
    "popup": {
      "tabs": [
```

```
        {
          "data": [
            {
              "formatted_value": "$461,000.00",
              "value": "461000",
              "topic_id": "location_sales_total",
              "label": "Sales: Total (USD)"
            }
          ],
          "tab_id": "1520003558209",
          "tab_label": "Additional info"
        }
      ],
      "header": [
        {
          "formatted_value": "$461,000.00",
          "value": "461000",
          "topic_id": "location_sales_total",
          "label": "Sales: Total (USD)"
        }
      ]
    },
    "position": {
      "lng": "-97.850325",
      "lat": "40.408435"
    },
    "color": "38d84a:Marker",
    "rowid": "row-0",
    "datatype": "business",
    "c2c": true,
    "uid": "11132553548894",
    "geoid": "11132553548894",
    "label": "Johnson Insurance & Financial Services"
  },
  {
    "popup": {
      "tabs": [
        {
          "data": [
            {
              "formatted_value": "$303,000.00",
              "value": "303000",
              "topic_id": "location_sales_total",
              "label": "Sales: Total (USD)"
            }
          ],
          "tab_id": "1520003558209",
          "tab_label": "Additional info"
        }
      ],
      "header": [
        {
          "formatted_value": "$303,000.00",
          "value": "303000",
```

```
            "topic_id": "location_sales_total",
            "label": "Sales: Total (USD)"
          }
        ]
      },
      "position": {
        "lng": "-97.861957",
        "lat": "40.549917"
      },
      "color": "38d84a:Marker",
      "rowid": "row-0",
      "datatype": "business",
      "c2c": true,
      "uid": "11140017127285",
      "geoid": "11140017127285",
      "label": "Schmer Family Farm"
    },
    {
      "popup": {
        "tabs": [
          {
            "data": [
              {
                "formatted_value": "$303,000.00",
                "value": "303000",
                "topic_id": "location_sales_total",
                "label": "Sales: Total (USD)"
              }
            ],
            "tab_id": "1520003558209",
            "tab_label": "Additional info"
          }
        ],
        "header": [
          {
            "formatted_value": "$303,000.00",
            "value": "303000",
            "topic_id": "location_sales_total",
            "label": "Sales: Total (USD)"
          }
        ]
      },
      "position": {
        "lng": "-97.863180",
        "lat": "40.557545"
      },
      "color": "38d84a:Marker",
      "rowid": "row-0",
      "datatype": "business",
      "c2c": true,
      "uid": "11140017127286",
      "geoid": "11140017127286",
      "label": "Leininger Family Farm"
    },
```

```
      {
        "popup": {
          "tabs": [
            {
              "data": [
                {
                  "formatted_value": "$303,000.00",
                  "value": "303000",
                  "topic_id": "location_sales_total",
                  "label": "Sales: Total (USD)"
                }
              ],
              "tab_id": "1520003558209",
              "tab_label": "Additional info"
            }
          ],
          "header": [
            {
              "formatted_value": "$303,000.00",
              "value": "303000",
              "topic_id": "location_sales_total",
              "label": "Sales: Total (USD)"
            }
          ]
        },
        "position": {
          "lng": "-97.900864",
          "lat": "40.509699"
        },
        "color": "38d84a:Marker",
        "rowid": "row-0",
        "datatype": "business",
        "c2c": true,
        "uid": "11140017127280",
        "geoid": "11140017127280",
        "label": "Cloet Family Farm"
      }
    ]
  }
}
```


Retrieve the Geographical Data of Country-Specific Shapes

The `GetBoundaryGeoJSON()` Apex method returns the geographical data, area, and perimeter for a list of geographical IDs.

Signature

```
Map<String, Object> maps.API.GetBoundaryGeoJSON(List<String> geoIds, Boolean mergeShape)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.

- `API` is the class that contains the global methods exposed to developers.
- `GetBoundaryGeoJSON()` is the method.

Sample Code

This code takes two geographical IDs and returns their total and individual geometric data, area, and perimeter.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Input Format of the Parameters

- A list of geographical IDs with the format `{countrycode}-{level}-{id}`.
- A Boolean that determines whether the returned information of the shape is merged (`true`) or separated (`false`).

Example

```
// Create a list of geographical IDs and set the shape merging parameter.
List<String> geoIds = new List<String>{'CAN-5-24360123', 'CAN-5-24360125'};
Boolean mergeShape = false;

// Call the GetBoundaryGeoJSON() method with the geographical IDs and shape merging
parameter.
Map<String, Object> response = maps.API.GetBoundaryGeoJSON(geoIds, mergeShape);

// Log the resulting boundary.
system.debug(response);
```

Sample Response

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

```
{
  "total": {
    "perimeter": 76052.889956049, // Total length in meters of all boundary perimeters.
    "area": 92743880.3549741 // Total area in square meters inside of all boundaries.
  },
  "unit": "meters",
  "geometries": {
    "CAN-5-24360125": {
      "perimeter": 64556.086852368, // Length in meters of the boundary perimeter.
      "area": 88393467.4191344 // Area in square meters inside the boundary.
    },
    "CAN-5-24360123": {
```

```
      "perimeter": 11496.803103681,
      "area": 4350412.93583968
    }
  },
  "count": 2,
  "custom": false
}
```

Retrieve the Geographical Data of Polygons

The `getPolygonGeometry()` Apex method returns the geometric data, such as the area and perimeter, for the requested geographical shape or custom polygon.

Signature

```
Map<String, Object> maps.API.getPolygonGeometry(Map<String, Object> polygon)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `getPolygonGeometry()` is the method.

Sample Code of a Circle

This code passes a circular geographical region to the Salesforce Maps API.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Input Format of a Circle

```
{
   type: 'circle', // Shape.
   radius: Decimal, // Radius in meters from the center of the polygon.
   lat: Decimal, // Latitude coordinate of the center of the circle.
   lng: Decimal // Longitude coordinate of the center of the circle.
}
```

Example

```
// Create a circular geographical region.
Map<String,Object> polygon = new Map<String,Object>{
    'type' => 'circle',
```

```
    'radius' => 50,
    'lat' => 33.917760,
    'lng' => -84.379750
};

// Call the getPolygonGeometry() method with the new polygon.
Map<String,Object> response = maps.API.getPolygonGeometry(polygon);

// Log the information about the resulting polygon.
system.debug(response);
```

Sample Code of a Rectangle

This code passes a rectangular geographical region to the Salesforce Maps API.

Input Format of a Rectangle

```
{
   type: 'rectangle', // Shape.
   NE: Map<String,Decimal>, // North-Eastern corner of the rectangle.
   SW: Map<String,Decimal> // South-Western corner of the rectangle.
}
```

Example

```
// Add the geographical coordinates of the North-Eastern and South-Western corners.
Map<String,Decimal> northEast = new Map<String,Decimal>{
    'lat' => 33.943360,
    'lng' => -84.515541
};

Map<String,Decimal> southWest = new Map<String,Decimal>{
    'lat' => 33.559707,
    'lng' => -84.101929
};

// Create the rectangular geographical region.
Map<String,Object> polygon = new Map<String,Object>{
    'type' => 'rectangle',
    'NE' => northEast,
    'SW' => southWest
};

// Call the getPolygonGeometry() method with the new polygon.
Map<String,Object> response = maps.API.getPolygonGeometry(polygon);

// Log the information about the resulting polygon.
system.debug(response);
```

Sample Code of a Polygon

This code passes a polygonal geographical region to the Salesforce Maps API.

Input Format of a Polygon

```
{
   type: 'polygon', // Shape.
   points: List<Map<String,Decimal>> // List of latitude and longitude coordinates of the
 polygon corners.
}
```

Example

```
// Create an array with the geographical coordinates of the corners of the polygon.
String pointsArray =
'[{"lat":34.29944707745,"lng":-83.86498113659},{"lat":34.16779924344,"lng":-85.15028504201},{"lat":32.82566556834,"lng":-84.63971024451},{"lat":33.27360674204,"lng":-82.72349886591}]';

// Create the polygonal geographical region.
Map<String,Object> polygon = new Map<String,Object>{
    'type' => 'polygon',
    'points'=> JSON.deserializeUntyped(pointsArray)
};

// Call the getPolygonGeometry() method with the new polygon.
Map<String,Object> response = maps.API.getPolygonGeometry(polygon);

// Log the information about the resulting polygon.
system.debug(response);
```

Sample Response

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

```
{
  "unit": "meters",
  "geometries": {
    "custom": {
      "area": 7803.6128806207,
      "perimeter": 313.654849054103
    }
  },
  "count": 1,
  "custom": true
}
```

Retrieve the Territories in Which the Records Reside

The `PointInPolygon()` Apex method returns the territory shape in which each record resides. It reads in the geographical coordinates of one or more records and compares each against the surrounding territory shape. Review Considerations for Retrieving Territory Shapes.

Signature

```
Map<String, Object> maps.API.PointInPolygon(Map<String, Object>)
```

Where,

- `maps` is the namespace that's available after you install Salesforce Maps.
- `API` is the class that contains the global methods exposed to developers.
- `PointInPolygon()` is the method.

Sample Code

This code reads in the geographical coordinates of two records and returns the territory shape in which each resides.

Warning: If you invoke methods within a flow, process builder, or trigger, avoid uncommitted work errors when you perform one of the following.

- Call the methods through a future method
- Call the methods as queueable

Uncommitted work errors can also arise if calling out to Salesforce Maps methods more than one time in a single transaction and the Salesforce Maps token hasn't refreshed in the last 24 hours. If these errors occur, separate each callout into individual transactions.

Different processes refresh the token, such as plotting a route or schedule. The refresh process is almost immediate after each qualifying action occurs.

Input Format

```
{
version: 'int' // Required. Version of the API endpoint.
points: [ {id:String, lat:Decimal, lng:Decimal, id:String}, ... ] // Array of record IDs,
 latitudes and longitutes.
MATerritoryIds: List<String> // List of Maps territory IDs.
}
```

Example

```
// Create a list of records.
List<Map<String, Object>> coordinateList = new List<Map<String, Object>> {
    new Map<String, Object> {
        'id' => 'recordId_1',
        'lat' => 34.0441233,
        'lng' => -84.0076379
    },
    new Map<String, Object> {
        'id' => 'recordId_2',
        'lat' => 54.0441233,
        'lng' => -84.0076379
    }
};

// Create a list of territory IDs.
List<String> MATerritoryIds = new List<String>();
for (maps__ShapeLayer__c sl : [SELECT Id FROM maps__ShapeLayer__c]) {
    MATerritoryIds.add(sl.Id);
```

```
}

// Build the request. Only version 2 is valid.
Map<String, Object> request = new Map<String, Object> {
    'version' => '2',
    'points' => coordinateList,
    'MATerritoryIds' => MATerritoryIds
};

// Call the PointInPolygon() method.
Map<String,Object> response = maps.Api.PointInPolygon(request);

// Log the resulting territory shapes of the regions.
system.debug(response);
```

Sample Response

Although the return value is an Apex `Map<String, Object>` object, this JSON response illustrates the essential data you receive in the resulting map.

```
{
"results": [
    {
        "polygons":["a130t000000nNefAAE-1"], // The ID of the polygon in which the record
 resides, followed by its unique ID for all geometry records associated with the polygon.

        "point":{
            "id":"recordId_1",
            "lat":34.0441233,
            "lng":-84.0076379
        }
    },
    {
        "polygons":[],
        "point":{
            "id":"recordId_2",
            "lat":54.0441233,
            "lng":-84.0076379
       }
    }
]
}
```