CDXGeoData REST XML Services – Reference Guide

Table of Contents

Introduction

CDXGeoData is a suite of RESTful web services for location-based data and analytics. It allows developers to perform tasks such as address validation, geocoding, routing, distance calculations, radius searches, and retrieving demographic information via simple HTTP requests. All services require a CDXGeoData API key for authentication, and some services that use third-party mapping data also require additional keys (e.g. a Google, HERE or Bing Maps API key). Each request consumes a certain number of *tokens* from your account (the cost is indicated by a TokenCharge in the response). Responses can be returned in JSON or XML format (and CSV for many services), making the API easy to integrate into a variety of applications.

Below is a comprehensive reference for each CDXGeoData function. Each section provides an overview of the function, usage requirements, request format (with base URL and parameters), response format (with key fields explained), example code in C#, VBA, and JavaScript, and sample outputs in JSON and XML. Please note that the token charges listed are accurate as of the date of this writing but may change in the future.

GeoLocate

Overview

The **GeoLocate** service provides forward geocoding (converting an address or place name into geographic coordinates) and basic reverse geocoding (converting coordinates into an address) using the Bing Maps platform. It accepts a free-form address or landmark query and returns one or more candidate locations with their standardized address and latitude/longitude. This service is useful for obtaining coordinates from an address or finding an address given a set of coordinates. *Note:* GeoLocate requires a Bing Maps API key to be configured in addition to the CDXGeoData key.

Settings and Requirements

- **CDXGeoData API Key:** Required for all requests (provided by CDX Technologies).

- **Bing Maps Key:** Required for GeoLocate (and other Bing-powered services like GeoRoute). This key must be obtained from the Bing Maps developer portal and configured in your CDXGeoData account settings.
- **Token Usage:** Each GeoLocate request costs 2 tokens. Free accounts include up to 1000 tokens/month, with additional tokens available for purchase.
- **Formats:** Supports `json` or `xml` responses (default is JSON). (CSV is not supported for this service.)

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geolocate`
- **HTTP Method:** GET
- **Required Parameters:**
  - **`key` (required)** – Your CDXGeoData API key for authentication.
  - **`query` (required)** – The address or place to geocode. This can be a full address, partial address, landmark name, or even a latitude/longitude pair for reverse geocoding. For reverse geocoding, supply the coordinates as `"latitude | longitude"` (e.g. `40.7128|-74.0060`).
  - **`format` (optional)** – Response format: `json` or `xml`. Default is `json`
  - **Example Request:**

```
https://geodata.cdxtech.com/api/geolocate?key={YOUR_API_KEY}&query=White House&format=xml
```

This example looks up “White House” and requests the XML response format.

Response Details

A successful GeoLocate response returns a top-level JSON or XML object with the following structure:

- **Service:** Name of the service (`GeoLocate`).
- **Status:** `"Success"` if the geocoding was successful (or `"Error"` if not).
- **TokenCharge:** Number of tokens consumed by the request (e.g. 2 for GeoLocate).
- **TotalResults:** The number of location results found.
- **Results:** An array of location result objects. Each result includes:
  - **formattedAddress:** A single-line address representation of the location (often a place name or city plus state).
  - **streetAddress:** The street address line (if available). This may be `null` for landmarks or city-level results.
  - **city:** The city of the address.
  - **county:** The county of the address.
  - **state:** The state or region code.
  - **zipCode:** The ZIP/Postal code if applicable (may be null for landmark-only queries).
  - **country:** Country name.

    - **latitude & longitude:** The latitude and longitude coordinates of the location.
    - **confidence:** Geocoding confidence level (e.g. `"High"`, `"Medium"`, etc.) indicating the match quality.
- **Usage:** A summary of your API usage after the call, including **used** tokens and **remaining** tokens in your plan. (For large token packages, percentage used may also be shown.)
- **Duration:** The server processing time for the request (in seconds).
- **TimeStamp:** The date/time when the response was generated.

If no location is found or an error occurs, `Status` will be `"Error"` and `Message` will contain an error description.

Coding Examples

C# Example: Using `HttpClient` to call GeoLocate and parse JSON.

```
string key = "YOUR_CDXGEODATA_KEY";
string query = "White House";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    StringBuilder url = new StringBuilder("/api/geolocate?");
    url.Append("key=").Append(key);
    url.Append("&query=").Append(query);
    url.Append("&format=").Append(format);

    HttpResponseMessage response = client.GetAsync(url.ToString()).Result;
    string content = response.Content.ReadAsStringAsync().Result;
    Console.WriteLine(content);
}
```

JavaScript Example (Browser XHR):

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/geolocate/', true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_CDXGEODATA_KEY&query=White%20House&format=json');
```

VBA Example: (Using Excel's VBA with Microsoft XML library)

```
Dim key As String: key = "YOUR_CDXGEODATA_KEY"
Dim query As String: query = "White House"
Dim url As String
url = "https://geodata.cdxtech.com/api/geolocate?key=" & key & "&query=" & _
      Application.EncodeURL(query) & "&format=xml"
' Create XMLHTTP object to send request
Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", url, False
```

```
xhr.Send
If xhr.Status = 200 Then
    Debug.Print xhr.ResponseText  ' XML response as string
End If
```

Sample Output

JSON Example: (Geocoding `"White House"` – abbreviated for brevity)


```
{
  "service": "GeoLocate",
  "status": "Success",
  "tokenCharge": 2,
  "totalResults": 5,
  "results": [
    {
      "formattedAddress": "White House, DC",
      "streetAddress": null,
      "city": "Washington",
      "county": "City of Washington",
      "state": "DC",
      "zipCode": null,
      "country": "United States",
      "latitude": 38.8973045349121,
      "longitude": -77.0365447998047,
      "confidence": "High"
    },
    {
      "formattedAddress": "White House, TN",
      "streetAddress": null,
      "city": "White House",
      "county": "Robertson",
      "state": "TN",
      "zipCode": null,
      "country": "United States",
      "latitude": 36.47092056274414,
      "longitude": -86.65138244628906,
      "confidence": "High"
    }
    // ... (3 more results) ...
  ],
  "usage": { "used": 2607, "remaining": 197393 },
  "duration": 0.1547903,
  "timeStamp": "2017-07-24T10:13:47.3486183-04:00"
}
```


In the above JSON, the query returned multiple possible locations named "White House" (one in DC, one in TN, etc.). Each result includes coordinates and context. The `usage` shows a high remaining token count because this was from an account with a large token plan.

XML Example: (for the first result of the same query)


```
<Root>
  <Service>GeoLocate</Service>
```

```
  <Status>Success</Status>
  <TokenCharge>2</TokenCharge>
  <TotalResults>5</TotalResults>
  <Results>
    <FormattedAddress>White House, DC</FormattedAddress>
    <StreetAddress/>
    <City>Washington</City>
    <County>City of Washington</County>
    <State>DC</State>
    <ZipCode/>
    <Country>United States</Country>
    <Latitude>38.8973045349121</Latitude>
    <Longitude>-77.0365447998047</Longitude>
    <Confidence>High</Confidence>
  </Results>
  <Usage>
    <Used>2607</Used>
    <Remaining>197393</Remaining>
  </Usage>
  <Duration>0.1547903</Duration>
  <TimeStamp>2017-07-24T10:13:47.3486183-04:00</TimeStamp>
</Root>
```

Each `<Results>` element in XML corresponds to one result location. In this example only the first result is shown. Subsequent `<Results>` entries would appear for additional matches. An empty tag (e.g. `<StreetAddress/>`) indicates that data is not applicable or not available for that result.

Geolocate Address

Overview

GeoLocateAddress performs forward geocoding for fully-specified addresses using Bing Maps. Unlike GeoLocate (which takes a single-line query), GeoLocateAddress accepts structured address components (street, city, state, ZIP, country) as separate parameters. This is useful when you already have discrete address fields or want to ensure accuracy by providing specific address parts. It returns the standardized address and coordinates of the best match (or multiple matches if applicable). GeoLocateAddress also supports reverse geocoding by entering latitude/longitude in the address fields (latitude in the street field and longitude in the city field as noted in CDXGeoData documentation). A Bing Maps API key is required for this service (same as for GeoLocate).

Settings and Requirements

- **API Keys:** Requires CDXGeoData API key and a Bing Maps key (for Bing geocoding).
- **Token Usage:** Each GeoLocateAddress request costs 2 tokens (same as GeoLocate).
- **Formats:** Supports `json` or `xml` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geolocateaddress`
- **HTTP Method:** GET
- **Required Parameters:**
  - **`key`** – Your API key for authentication.
  - **`address`** – Street address line (e.g. street name and number). Partial addresses are acceptable. You may also input a latitude here for reverse geocoding (with longitude in the `city` field).
  - **`city`** – City name. (For reverse geocoding, put the longitude here if latitude is in the address field).
  - **`state`** – State abbreviation or full state name.
  - **`zipcode`** – 5-digit ZIP Code (optional but helps narrow down the location).
  - **`country`** – Country name or ISO country code (optional, default is US).
  - **`format`** – Response format: `json` or `xml`. Default is `json`.

All components are combined to form the full address query. For example, you can provide just a street and ZIP, or street/city/state, etc., and the geocoder will attempt to find a match.

Example Request:

```
https://geodata.cdxtech.com/api/geolocateaddress?key=YOUR_KEY
    &address=2 West Hanover
Ave&city=Randolph&state=NJ&zipcode=07869&country=US&format=json
```

This request geocodes the address "2 West Hanover Ave, Randolph, NJ 07869" in the US.

Response Details

GeoLocateAddress returns a structure very similar to **GeoLocate**. Key fields include:

- **Service:** `"GeoLocateAddress"`.
- **Status, TokenCharge, Message, TotalResults:** Same meaning as in GeoLocate. Typically `TotalResults` will be 1 for an exact address match (or 0 if not found).
- **Results:** An array of result objects (usually one). Each result provides:
  - **formattedAddress:** The full normalized address of the best match (including ZIP+4 if available).
  - **streetAddress:** Standardized street address line (house number and street).
  - **city, state, zipCode, country:** Standardized city, state, ZIP, country for the address.
  - **latitude, longitude:** Coordinates of the address.
  - **confidence:** Confidence level of the match (High/Medium/Low).
- **Usage, Duration, TimeStamp:** Same as in GeoLocate, indicating tokens used and time taken.

If the address is not found or is ambiguous, the service may return multiple close matches or an error status. In case of multiple matches, `TotalResults` will reflect the number of candidates.

Coding Examples

C# Example:

```
string key = "YOUR_CDXGEODATA_KEY";
string address = "2 West Hanover Ave";
string city = "Randolph";
string state = "NJ";
string zipcode = "07869";
string country = "US";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    StringBuilder url = new StringBuilder("/api/geolocateaddress?");
    url.Append("key=").Append(key)
       .Append("&address=").Append(Uri.EscapeDataString(address))
       .Append("&city=").Append(Uri.EscapeDataString(city))
       .Append("&state=").Append(state)
       .Append("&zipcode=").Append(zipcode)
       .Append("&country=").Append(country)
       .Append("&format=").Append(format);
    HttpResponseMessage message = client.GetAsync(url.ToString()).Result;
    string resultJson = message.Content.ReadAsStringAsync().Result;
    Console.WriteLine(resultJson);
}
```

JavaScript Example (Browser XHR):

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/geolocateaddress/', true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_CDXGEODATA_KEY&address=2%20West%20Hanover%20Ave'
       + '&city=Randolph&state=NJ&zipcode=07869&country=US&format=json');
```

VBA Example (using VBA-Web library):

```
Dim Client As New WebClient
Dim Request As New WebRequest

Dim key As String, address As String, city As String, state As String,
zipcode As String
key = "YOUR_CDXGEODATA_KEY"
address = "2 West Hanover Ave"
city = "Randolph"
state = "NJ"
zipcode = "07869"
```

```
Client.BaseUrl = "https://geodata.cdxtech.com/api/"
Request.Method = WebMethod.HttpGet
Request.Resource = "geolocateaddress?key={key}&address={address}&city={city}"
& _
                   "&state={state}&zipcode={zipcode}&format=json"

Request.AddUrlSegment "key", key
Request.AddUrlSegment "address", address
Request.AddUrlSegment "city", city
Request.AddUrlSegment "state", state
Request.AddUrlSegment "zipcode", zipcode

Dim Response As WebResponse
Set Response = Client.Execute(Request)
Debug.Print Response.Content  ' JSON result
```

(The above VBA example uses the VBA-Web library for cleaner HTTP calls.)

Sample Output

JSON Example: (Geocoding `2 West Hanover Ave, Randolph, NJ`):


```
{
  "service": "GeoLocateAddress",
  "status": "Success",
  "tokenCharge": 2,
  "totalResults": 1,
  "results": [
    {
      "formattedAddress": "2 W Hanover Ave, Randolph, NJ 07869-4221",
      "streetAddress": "2 W HANOVER AVE",
      "city": "RANDOLPH",
      "state": "NJ",
      "zipCode": "07869",
      "country": "United States",
      "latitude": 40.82653,
      "longitude": -74.56786,
      "confidence": "High"
    }
  ],
  "usage": { "used": 2611, "remaining": 138389 },
  "duration": 0.0875,
  "timeStamp": "2025-04-28T10:40:15-04:00"
}
```


In this JSON output, the address was found with high confidence. The `formattedAddress` includes the ZIP+4 extension. The coordinates (lat/long) pinpoint the address location.

XML Example: (for the same result)


```
<Root>
  <Service>GeoLocateAddress</Service>
  <Status>Success</Status>
  <TokenCharge>2</TokenCharge>
```

```
  <TotalResults>1</TotalResults>
  <Results>
    <FormattedAddress>2 W Hanover Ave, Randolph, NJ 07869-
4221</FormattedAddress>
    <StreetAddress>2 W HANOVER AVE</StreetAddress>
    <City>RANDOLPH</City>
    <State>NJ</State>
    <ZipCode>07869</ZipCode>
    <Country>United States</Country>
    <Latitude>40.82653</Latitude>
    <Longitude>-74.56786</Longitude>
    <Confidence>High</Confidence>
  </Results>
  <Usage>
    <Used>2611</Used>
    <Remaining>138389</Remaining>
  </Usage>
  <Duration>0.0875</Duration>
  <TimeStamp>2025-04-28T10:40:15-04:00</TimeStamp>
</Root>
```

GeoLocatePoint

Overview

GeoLocatePoint provides reverse geocoding given coordinates (latitude and longitude) as input. This is essentially the counterpart to GeoLocateAddress for coordinate lookup. By supplying a latitude/longitude pair, GeoLocatePoint returns the nearest address or location information. This service uses Bing Maps data for reverse geocoding, so a Bing Maps key is required. (Note: Reverse geocoding can also be done with the GeoLocate service by passing a `"lat|lon"` string in the `query` parameter, but GeoLocatePoint offers a direct way to use separate lat/long parameters.)

Settings and Requirements

- **API Keys:** Requires CDXGeoData API key and a Bing Maps key (for the reverse geocoder).
- **Token Usage:** 2 tokens per request (same cost as GeoLocate).
- **Formats:** Supports `json` or `xml` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geolocatepoint`
- **HTTP Method:** GET
- **Required Parameters:**
  - **`key`** – Your CDXGeoData API key.
  - **`latitude`** – Latitude of the location to reverse geocode.
  - **`longitude`** – Longitude of the location to reverse geocode.

- **`format`** – `json` or `xml`. Default is `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geolocatepoint?key=YOUR_KEY&latitude=40.82653
&longitude=-74.56786&format=json
```

This request would find the address near latitude 40.82653, longitude -74.56786 (which should correspond to the address in Randolph, NJ from the previous example).

Response Details

GeoLocatePoint's response is similar to GeoLocate/GeoLocateAddress. It will return location details for the coordinates provided, including:

- **formattedAddress:** Nearest address or location name.
- **streetAddress, city, county, state, zipCode, country:** Components of the found address (if any). For example, a latitude/longitude in the middle of a city might return just the city/state if no specific address is nearby, whereas coordinates on a specific building will return that address.
- **latitude, longitude:** Echo of the coordinates (or adjusted coordinates of the nearest known address).
- **confidence:** Confidence level of the result (e.g., High if an exact address was found, or lower if it's an approximation).

It also includes the standard **Service**, **Status**, **TokenCharge**, **Usage**, **Duration**, **TimeStamp** fields as described earlier. Typically one result is returned (`TotalResults = 1`), being the best match address for the given point.

Coding Examples

C# Example:

```
string key = "YOUR_CDXGEODATA_KEY";
string lat = "40.82653";
string lon = "-74.56786";
string format = "xml";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geolocatepoint?key={key}&latitude={lat}&longitude={lon}&format={format
}";
    HttpResponseMessage resp = client.GetAsync(url).Result;
    string resultXml = resp.Content.ReadAsStringAsync().Result;
    Console.WriteLine(resultXml);
}
```

JavaScript Example:

```
fetch(`https://geodata.cdxtech.com/api/geolocatepoint?key=YOUR_CDXGEODATA_KEY
`
      + `&latitude=40.82653&longitude=-74.56786&format=json`)
  .then(response => response.text())
  .then(data => console.log(data));
```

VBA Example:

```
Dim key$, lat$, lon$
key = "YOUR_CDXGEODATA_KEY"
lat = "40.82653"
lon = "-74.56786"
Dim url$
url = "https://geodata.cdxtech.com/api/geolocatepoint?key=" & key _
    & "&latitude=" & lat & "&longitude=" & lon & "&format=json"
Dim xhr As Object: Set xhr = CreateObject("MSXML2.ServerXMLHTTP")
xhr.Open "GET", url, False
xhr.Send
If xhr.Status = 200 Then
    Debug.Print xhr.ResponseText  ' JSON result string
End If
```

Sample Output

JSON Example: (Reverse geocoding lat=40.82653, lon=-74.56786)


```
{
  "service": "GeoLocatePoint",
  "status": "Success",
  "tokenCharge": 2,
  "totalResults": 1,
  "results": [
    {
      "formattedAddress": "2 W Hanover Ave, Randolph, NJ 07869-4221",
      "streetAddress": "2 W HANOVER AVE",
      "city": "RANDOLPH",
      "county": "MORRIS",
      "state": "NJ",
      "zipCode": "07869",
      "country": "United States",
      "latitude": 40.82653,
      "longitude": -74.56786,
      "confidence": "High"
    }
  ],
  "usage": { "used": 2613, "remaining": 138387 },
  "duration": 0.0901,
  "timeStamp": "2025-04-28T10:40:45-04:00"
}
```

This output shows that the coordinates correspond to the address "2 W Hanover Ave..." with a high confidence. GeoLocatePoint effectively returned the same address that was originally geocoded in the GeoLocateAddress example, confirming the reverse geocode.

XML Example:

```
<Root>
  <Service>GeoLocatePoint</Service>
  <Status>Success</Status>
  <TokenCharge>2</TokenCharge>
  <TotalResults>1</TotalResults>
  <Results>
    <FormattedAddress>2 W Hanover Ave, Randolph, NJ 07869-
4221</FormattedAddress>
    <StreetAddress>2 W HANOVER AVE</StreetAddress>
    <City>RANDOLPH</City>
    <County>MORRIS</County>
    <State>NJ</State>
    <ZipCode>07869</ZipCode>
    <Country>United States</Country>
    <Latitude>40.82653</Latitude>
    <Longitude>-74.56786</Longitude>
    <Confidence>High</Confidence>
  </Results>
  <Usage>
    <Used>2613</Used>
    <Remaining>138387</Remaining>
  </Usage>
  <Duration>0.0901</Duration>
  <TimeStamp>2025-04-28T10:40:45-04:00</TimeStamp>
</Root>
```

GeoRoute

Overview

The **GeoRoute** service calculates driving (or walking/transit) directions between two locations, returning the travel distance and estimated travel time. This service leverages Bing Maps routing data to account for real road distances and travel modes. It is useful for finding how far apart two addresses are by road and how long a trip might take. (For straight-line distance “as the crow flies” between ZIP centroids, see **GeoDistance** instead.) **GeoRoute requires a Bing Maps key** in addition to the CDXGeoData key.

GeoRoute can optionally consider different travel modes (driving, walking, transit), route optimizations (shortest distance vs fastest time, etc.), and avoidance of tolls or highways. It does not return turn-by-turn directions in the response, but focuses on summary information for the route.

Settings and Requirements

- **API Keys:** CDXGeoData API key + Bing Maps key (for routing).
- **Token Usage:** Each GeoRoute request costs 3 tokens.
- **Formats:** Supports `json` or `xml` (default `json`). (CSV not supported, as output is structured data.)

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/georoute`
- **HTTP Method:** GET
- **Required Parameters:**
  - **`key`** – Your API key.
  - **`routeStart`** – Starting location, as an address, landmark, or coordinates (lat,long). You can provide a fully specified address or lat/long for the start.
  - **`routeEnd`** – Destination location (address or coordinates).
- **Optional Parameters:**
  - **`travelMode`** – Mode of travel: `driving` (default), `walking`, or `transit`.
  - **`optimize`** – Route optimization: `distance` (shortest path) or `time` (fastest, default). If `travelMode=driving`, you can also use `timeWithTraffic` (consider current traffic) or `timeAvoidClosure` (avoid road closures).
  - **`distanceUnit`** – Unit for distance output: `mi`/`mile` (default) or `km`/`kilometer`. This affects the units reported in the response (miles vs kilometers).
  - **`avoidOptions`** – Types of roads to avoid: can be blank (allow all, default) or any combination of `highways`, `tolls`, `minimizeHighways`, `minimizeTolls`. For example, `avoidOptions=highways,tolls` will avoid toll roads and highways if possible.
  - **`transitDateTime`** – (For `travelMode=transit` or to consider departure time in driving) Specify the date/time for transit routing or desired departure time. Format can be `YYYY-MM-DD` or `YYYY-MM-DD HH:MM:SS`. If not given for transit, Bing assumes current time.
  - **`transitTimeMode`** – (Transit mode only) Either `departure` (default), `arrival`, or `lastAvailable`, indicating whether the `transitDateTime` is a departure time or arrival deadline.
  - **`format`** – `json` or `xml`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/georoute?key=YOUR_KEY
  &routeStart=34.1026760,-118.4524720
  &routeEnd=33.78750,-117.93320
  &travelMode=driving&optimize=time&distanceUnit=mi&avoidOptions=
  &format=json
```

This example computes a driving route from coordinates 34.1026760,-118.4524720 to 33.78750,-117.93320 (roughly, a route in California), optimizing for fastest time, in miles, avoiding nothing.

Response Details

The GeoRoute response provides summary information about the route:

- **Service:** `"GeoRoute"`.
- **Status:** `"Success"` if the route was found. If no route is possible (e.g., unreachable locations), status may be `"Error"`.
- **TokenCharge:** 3 (for each call).
- **Results:** Contains route information, typically a single `<Route>` entry for the calculated path. Key fields within the route result:
    - **travelDistance:** Total distance of the route in the requested unit (miles or kilometers).
    - **travelDuration:** Estimated travel time in seconds for the route. (You may convert to minutes or hours in your application as needed.)
    - **startAddress / endAddress:** The resolved addresses for the start and end points. These confirm the exact locations used (which may be standardized if you provided a fuzzy input).
    - **startCoordinates / endCoordinates:** Latitude/longitude of the start and end points (as used in routing).
    - **routePath (if applicable): Note:** The GeoRoute service **does not include a turn-by-turn breakdown** or the route geometry in the standard output. It returns summary distance and time only. If you need the actual route path coordinates, see **GeoRoutePoint** below.
- **Usage, Duration, TimeStamp:** Standard metadata as in other services. The `Duration` here refers to the processing time of the API call, *not* the travel time.

For example, a driving route of ~48.37 miles taking ~3249 seconds (~54.1 minutes) might appear in JSON as `"travelDistance": 48.370019, "travelDuration": 3249.0`.

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string routeStart = "New York, NY";
string routeEnd = "Boston, MA";
string travelMode = "driving";
string optimize = "distance";  // shortest distance
string unit = "km";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/georoute?key={key}&routeStart={Uri.EscapeDataString(routeStart)}"
                +
$"&routeEnd={Uri.EscapeDataString(routeEnd)}&travelMode={travelMode}"
                +
$"&optimize={optimize}&distanceUnit={unit}&avoidOptions=&format=json";
```

```
    HttpResponseMessage resp = client.GetAsync(url).Result;
    string json = resp.Content.ReadAsStringAsync().Result;
    Console.WriteLine(json);
}
```

JavaScript Example:

```
const params = new URLSearchParams({
  key: 'YOUR_KEY',
  routeStart: 'New York, NY',
  routeEnd: 'Boston, MA',
  travelMode: 'driving',
  optimize: 'distance',
  distanceUnit: 'km',
  format: 'json'
});
fetch('https://geodata.cdxtech.com/api/georoute?' + params.toString())
  .then(res => res.json())
  .then(data => console.log(data.route.travelDistance + ' ' +
data.route.distanceUnit
                            + ', ' + data.route.travelDuration + '
seconds'));
```

VBA Example:

```
Dim key$, start$, dest$
key = "YOUR_KEY"
start = "New York, NY"
dest = "Boston, MA"
Dim reqUrl$
reqUrl = "https://geodata.cdxtech.com/api/georoute?key=" & key & _
         "&routeStart=" & URLEncode(start) & "&routeEnd=" & URLEncode(dest) &
_
         "&travelMode=driving&optimize=distance&distanceUnit=km&format=xml"
' (URLEncode is a helper function to escape spaces, etc.)
Dim http As Object: Set http = CreateObject("MSXML2.XMLHTTP")
http.Open "GET", reqUrl, False
http.Send
If http.Status = 200 Then
    Dim xmlDoc As Object: Set xmlDoc = CreateObject("MSXML2.DOMDocument")
    xmlDoc.LoadXML http.responseText
    Debug.Print xmlDoc.SelectSingleNode("//travelDistance").Text & " " & _
                xmlDoc.SelectSingleNode("//DistanceUnit").Text
End If
```

Sample Output

JSON Example: (Driving route from Los Angeles coordinates to Orange County coordinates)

```
{
  "service": "GeoRoute",
  "status": "Success",
  "tokenCharge": 3,
  "results": [
```

```
    {
      "travelDistance": 48.37,
      "distanceUnit": "miles",
      "travelDuration": 3249.0,
      "startAddress": "1729 Sepulveda Blvd, Los Angeles, CA",
      "endAddress": "13200 Harbor Blvd, Garden Grove, CA",
      "startCoordinates": { "latitude": 34.102676, "longitude": -118.452472
},
      "endCoordinates": { "latitude": 33.78750, "longitude": -117.93320 }
    }
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.201,
  "timeStamp": "2025-04-28T10:41:15-04:00"
}
```


In this output, the route covers ~48.37 miles and would take about 3249 seconds (~54 minutes) driving. The addresses for the start and end were resolved and are displayed. The usage shows 100 used/1000 remaining (this example might be from a test account with initial tokens).

XML Example:


```
<Root>
  <Service>GeoRoute</Service>
  <Status>Success</Status>
  <TokenCharge>3</TokenCharge>
  <Results>
    <Route>
      <TravelDistance>48.37</TravelDistance>
      <DistanceUnit>miles</DistanceUnit>
      <TravelDuration>3249.0</TravelDuration>
      <StartAddress>1729 Sepulveda Blvd, Los Angeles, CA</StartAddress>
      <EndAddress>13200 Harbor Blvd, Garden Grove, CA</EndAddress>
      <StartCoordinates>
        <Latitude>34.102676</Latitude>
        <Longitude>-118.452472</Longitude>
      </StartCoordinates>
      <EndCoordinates>
        <Latitude>33.78750</Latitude>
        <Longitude>-117.93320</Longitude>
      </EndCoordinates>
    </Route>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.201</Duration>
  <TimeStamp>2025-04-28T10:41:15-04:00</TimeStamp>
</Root>
```

GeoRoutePoint

Overview

GeoRoutePoint is an extension of GeoRoute that can handle multiple waypoints and can return route path information. It is used for calculating routes with one or more intermediate points (stops) between the start and end, and is capable of providing the coordinates along the route if needed. Essentially, you supply a series of points (addresses or coordinates) and GeoRoutePoint will compute the route through all of them in order. This can be useful for mapping or for getting a polyline of the route. Like GeoRoute, it uses Bing Maps and therefore requires a Bing Maps key.

Settings and Requirements

- **API Keys:** CDXGeoData key + Bing Maps key required.
- **Token Usage:** Each GeoRoutePoint request costs 3 tokens (same as GeoRoute).
- **Formats:** Supports `json` or `xml` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/georoutepoint`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`wayPoints`** – A series of locations (addresses or coordinates) separated by the underscore character _. This single parameter encodes the entire route's stops. For example:
    `wayPoints=34.1026760,-118.4524720_34.01950,-118.49060_33.78750,-117.93320`
    defines a route starting at (34.1026760, -118.4524720), then going through (34.01950, -118.49060), and ending at (33.78750, -117.93320). You can also use addresses in place of coordinates, similarly separated by underscores.
  - **`travelMode`** – `driving`, `walking`, or `transit` (default `driving`).
  - **`optimize`** – Route optimization (default `time`). Options: `distance`, `time`, `timeWithTraffic`, `timeAvoidClosure` (same as GeoRoute).
  - **`distanceUnit`** – `mi`/`mile` or `km`/`kilometer` (default `mi`).
  - **`avoidOptions`** – Road types to avoid or minimize (same options as GeoRoute).
  - **`transitDateTime`**, **`transitTimeMode`** – Same usage as in GeoRoute for transit scenarios.
  - **`format`** – `json` or `xml` (default `json`).

Example Request:

```
https://geodata.cdxtech.com/api/georoutepoint?key=YOUR_KEY
  &wayPoints=40.7128,-74.0060_41.8240,-71.4128_42.3601,-71.0589
  &travelMode=driving&optimize=time&distanceUnit=mi&format=xml
```

This would calculate a driving route starting in New York City (`40.7128,-74.0060`), through Providence, RI (`41.8240,-71.4128`), ending in Boston, MA (`42.3601,-71.0589`), optimizing for time in miles, with XML output.

Response Details

The GeoRoutePoint response includes similar summary information as GeoRoute, but can also provide the route geometry (series of points) and handles multiple segments:

- **Service:** `"GeoRoutePoint"`.
- **Status, TokenCharge:** Similar to GeoRoute.
- **Results:** Typically contains one route result (even if multiple waypoints, the route is considered one continuous path). Inside the route result:
    - **travelDistance:** Total distance of the entire route.
    - **distanceUnit:** Unit of distance (mi or km).
    - **travelDuration:** Total travel time (seconds).
    - **wayPoints:** The list of waypoints that were used (addresses or coordinates). In JSON this might be an array of points, in XML possibly repeated elements or a single string.
    - **routePath:** If available, the coordinates along the route's polyline. In JSON this could appear as an array of latitude/longitude pairs detailing the route shape. *(Note: The documentation implies GeoRoutePoint can return route points, but exact output format for the path may depend on the API. If the raw output does not include the path by default, contacting support or enabling a specific option might be necessary.)*

Like GeoRoute, turn-by-turn directions are not listed, but the *path* (the sequence of coordinates comprising the route) is available for mapping. The output will also include **startAddress**, **endAddress**, etc., for the endpoints and any intermediate addresses resolved.

- **Usage, Duration, TimeStamp:** As usual, indicating tokens used and processing time.

Coding Examples

C# Example: (Route with multiple stops)

```
string key = "YOUR_KEY";
string wayPoints = "Seattle, WA_Los Angeles, CA_San Diego, CA";  // three
stops
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/georoutepoint?key={key}&wayPoints={Uri.EscapeDataString(wayPoints)}"
              +
"&travelMode=driving&optimize=distance&distanceUnit=mi&format=json";
```

```
    var response = client.GetAsync(url).Result;
    string json = response.Content.ReadAsStringAsync().Result;
    Console.WriteLine(json);
}
```

JavaScript Example:

```
let waypoints = "Seattle, WA_Los Angeles, CA_San Diego, CA";
let apiUrl = `https://geodata.cdxtech.com/api/georoutepoint?key=YOUR_KEY`
           + `&wayPoints=${encodeURIComponent(waypoints)}`
           +
`&travelMode=driving&optimize=distance&distanceUnit=mi&format=json`;
fetch(apiUrl)
  .then(res => res.json())
  .then(data => {
      console.log("Total distance:", data.results[0].travelDistance,
data.results[0].distanceUnit);
      console.log("Total time (hrs):", data.results[0].travelDuration/3600);
      // If route path points are present:
      if(data.results[0].pathPoints) {
          console.log("Route path points count:",
data.results[0].pathPoints.length);
      }
  });
```

VBA Example:

```
Dim key$, pts$
key = "YOUR_KEY"
pts = "34.1026760,-118.4524720_34.01950,-118.49060_33.78750,-117.93320"
Dim url$
url = "https://geodata.cdxtech.com/api/georoutepoint?key=" & key &
"&wayPoints=" & pts & _
      "&travelMode=driving&optimize=time&distanceUnit=mi&format=xml"
Dim http As Object: Set http = CreateObject("MSXML2.ServerXMLHTTP")
http.Open "GET", url, False
http.Send
If http.Status = 200 Then
    Debug.Print http.responseText  ' XML string containing route info
End If
```

Sample Output

JSON Example: (Multiple-stop route – simplified example)

```
{
  "service": "GeoRoutePoint",
  "status": "Success",
  "tokenCharge": 3,
  "results": [
    {
      "travelDistance": 110.5,
      "distanceUnit": "miles",
      "travelDuration": 7200.0,
```

```
      "startAddress": "Point A Address...",
      "endAddress": "Point C Address...",
      "wayPoints": [
         "Point A Address...",
         "Point B Address...",
         "Point C Address..."
      ],
      "pathPoints": [
         { "latitude": 34.102676, "longitude": -118.452472 },
         // ... (many intermediate points) ...
         { "latitude": 33.78750, "longitude": -117.93320 }
      ]
    }
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.250,
  "timeStamp": "2025-04-28T10:42:10-04:00"
}
```

In this hypothetical JSON, `pathPoints` would contain a sequence of coordinates outlining the entire route from start to end (through any intermediate waypoints). The total distance and duration are provided for the full route.

XML Example:

```
<Root>
  <Service>GeoRoutePoint</Service>
  <Status>Success</Status>
  <TokenCharge>3</TokenCharge>
  <Results>
    <Route>
      <TravelDistance>110.5</TravelDistance>
      <DistanceUnit>miles</DistanceUnit>
      <TravelDuration>7200.0</TravelDuration>
      <StartAddress>Point A Address...</StartAddress>
      <EndAddress>Point C Address...</EndAddress>
      <WayPoints>
        <WayPoint>Point A Address...</WayPoint>
        <WayPoint>Point B Address...</WayPoint>
        <WayPoint>Point C Address...</WayPoint>
      </WayPoints>
      <!-- Route path could be represented as a series of points -->
      <Path>
        <Point><Latitude>34.102676</Latitude><Longitude>-
118.452472</Longitude></Point>
        ...
        <Point><Latitude>33.78750</Latitude><Longitude>-
117.93320</Longitude></Point>
      </Path>
    </Route>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
```

```
  <Duration>0.250</Duration>
  <TimeStamp>2025-04-28T10:42:10-04:00</TimeStamp>
</Root>
```

Note: The exact structure for the route path in XML/JSON may vary. The above is a logical representation assuming the API returns path coordinates for GeoRoutePoint.

GeoVerify

Overview

GeoVerify performs address verification and standardization using the United States Postal Service (USPS) data. It takes a street address with city/state/ZIP, and returns a corrected, standardized address (including the proper USPS formatting, ZIP+4 code, and other details) if the address is valid. This is essentially a USPS address validation service: it will fill in missing ZIP+4, correct misspellings, and verify that the address exists. It's useful for cleaning mailing lists or validating user-entered addresses. **Note:** GeoVerify is limited to U.S. addresses (it uses USPS data). A Bing Maps key is *not* required for GeoVerify, only the CDXGeoData key.

Settings and Requirements

- **API Key:** CDXGeoData API key required. No external mapping key needed (uses USPS data).
- **Token Usage:** Each GeoVerify request costs 5 tokens (higher cost due to USPS data usage).
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`). CSV output is convenient for directly importing verified addresses into spreadsheets.

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geoverify`
- **HTTP Method:** GET
- **Required Parameters:**
  - **`key`** – Your API key.
  - **`address`** – The street address to verify (e.g. house number and street name).

- **`citystatezip`** – The city, state, and ZIP code, combined in one parameter. This should include the city name, state abbreviation, and ZIP code (5-digit). These can be in any order or separated by spaces. For example: `"New Brunswick NJ 08901"`. (You may omit components, like provide city and state without ZIP, and the service will still attempt to find the address.)
- **`format`** – `json`, `xml`, or `csv`. Default is `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geoverify?key=YOUR_KEY
    &address=9 Spring St&citystatezip=New Brunswick NJ 08901&format=json
```

This will verify the address "9 Spring St, New Brunswick NJ 08901".

Response Details

GeoVerify returns a wealth of information about the address. Key elements include:

- **Service:** `"GeoVerify"`.
- **Status:** `"Success"` if the address was found in the USPS database (or `"Error"` if not).
- **TokenCharge:** 2 (per request).
- **Results:** An array of address results (usually 1 if the address is valid). The result fields are tailored to address data:
  - **AddressIn:** The input street address that was provided.
  - **CityStateZipIn:** The input city, state, ZIP provided.
  - **AddressOut:** The corrected street address line, standardized to USPS formatting (e.g., abbreviations).
  - **StreetAddressOut:** Similar to AddressOut (often identical for simple addresses). It may exclude secondary unit info depending on formatting.
  - **CityStateZipOut:** The corrected city, state, and ZIP+4, formatted properly.
  - **FullAddressOut:** The full corrected address (street, city, state ZIP+4 all in one line).
  - **City, State, Zipcode, Zip4:** These separate fields give the individual components of the corrected address: city, state, 5-digit ZIP, and ZIP+4 extension.
  - **AddressType:** The type of address: e.g. `S` for street or residential/business address, `P` for PO Box, etc. (In the example, "S" is returned, meaning a street address).
  - **County:** County name of the address.
  - **CountyFIPS:** The 3-digit county FIPS code.
  - **StateFIPS:** The 2-digit state FIPS code.
  - **TimeZone:** Time zone of the address (e.g., `EST` for Eastern Standard Time).
  - **DPBC:** Delivery Point Bar Code – an 11-digit number (ZIP+4 plus two extra digits) for mail sorting.
  - **CongressionalDistrict:** U.S. Congressional District number for the address.
  - **NineDigitZipcode:** The ZIP+4 in one string (e.g., `08901-2107`).

    - **BoxType, BoxNumber, BoxTypes:** These relate to box addresses (e.g., PO Boxes). They will contain values if the address is a box address; otherwise they are blank. For example, for a PO Box input, `BoxType` might be "PO BOX" and `BoxNumber` the box number. In the example of a street address, these fields are empty.
    - **PreferredCity, PreferredState:** In cases where the USPS has a “preferred” city name for a given ZIP Code, this shows the preferred city/state. (Sometimes a ZIP covers multiple municipalities; USPS assigns one as the mailing city.) In the example, `PreferredCity` and `PreferredState` are "NEW BRUNSWICK" and "NJ", matching the input city. If an alternate city name was used in input, USPS’s preferred name would appear here.
    - **ErrorMessage:** If the address was not found or had issues, this field would contain an error or warning message. On a successful find, it is empty.
- **Usage:** Token usage info (e.g., after this call, how many used and remaining).
- **Duration:** Processing time (seconds).
- **TimeStamp:** Timestamp of the request.

If the address cannot be verified, `Status` will be `"Error"` and `Message`/`ErrorMessage` will explain (for example, "Address not found" or which part is problematic). Partial matches might still return suggestions or corrections if possible.

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string addr = "9 Spring St";
string cityStateZip = "New Brunswick NJ 08901";
string format = "json";

using(HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geoverify?key={key}&address={Uri.EscapeDataString(addr)}"
              +
$"&citystatezip={Uri.EscapeDataString(cityStateZip)}&format={format}";
    HttpResponseMessage response = client.GetAsync(url).Result;
    string result = response.Content.ReadAsStringAsync().Result;
    Console.WriteLine(result);
}
```

JavaScript Example:

```
let params = new URLSearchParams({
  key: 'YOUR_KEY',
  address: '9 Spring St',
  citystatezip: 'New Brunswick NJ 08901',
  format: 'xml'
});
```

```
fetch('https://geodata.cdxtech.com/api/geoverify?' + params.toString())
  .then(res => res.text())
  .then(xmlString => console.log(xmlString));
```

VBA Example:

```
Dim key$, addr$, cityStateZip$
key = "YOUR_KEY"
addr = "9 Spring St"
cityStateZip = "New Brunswick NJ 08901"

Dim reqURL$
reqURL = "https://geodata.cdxtech.com/api/geoverify?key=" & key & _
         "&address=" & URLEncode(addr) & "&citystatezip=" &
URLEncode(cityStateZip) & "&format=csv"

Dim http As Object: Set http = CreateObject("MSXML2.XMLHTTP")
http.Open "GET", reqURL, False
http.Send

If http.Status = 200 Then
    Debug.Print http.responseText  ' CSV string of verified address data
End If
```

In the VBA example, we request CSV format, which returns a single-line CSV with all the output fields. This can be easily split by commas or imported into Excel.

Sample Output

JSON Example: (Verifying `9 Spring St, New Brunswick NJ 08901`)


```
{
  "service": "GeoVerify",
  "status": "Success",
  "tokenCharge": 2,
  "totalResults": 1,
  "results": [
    {
      "addressIn": "9 Spring St",
      "cityStateZipIn": "New Brunswick NJ 08901",
      "addressOut": "9 SPRING ST",
      "streetAddressOut": "9 SPRING ST",
      "cityStateZipOut": "NEW BRUNSWICK, NJ 08901-2107",
      "fullAddressOut": "9 SPRING ST, NEW BRUNSWICK, NJ 08901-2107",
      "city": "NEW BRUNSWICK",
      "state": "NJ",
      "zipcode": "08901",
      "zip4": "2107",
      "addressType": "S",
      "county": "MIDDLESEX",
      "countyFIPS": "023",
      "stateFIPS": "34",
      "timeZone": "EST",
      "DPBC": "089012107093",
```

```
      "congressionalDistrict": "06",
      "nineDigitZipcode": "08901-2107",
      "boxTypes": null,
      "boxType": "",
      "boxNumber": "",
      "preferredCity": "NEW BRUNSWICK",
      "preferredState": "NJ",
      "errorMessage": null
    }
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.0284,
  "timeStamp": "2025-04-28T10:42:45-04:00"
}
```

Here we see the input address was found and standardized. For example, `"New Brunswick NJ 08901"` became `"NEW BRUNSWICK, NJ 08901-2107"` with the ZIP+4. The `DPBC` is provided (089012107093), and `addressType` is "S" indicating a street address. `countyFIPS` 023 and `stateFIPS` 34 correspond to Middlesex County, NJ. The `preferredCity` is the same as input (indicating "New Brunswick" is the correct USPS city name for 08901). `errorMessage` is null, meaning no errors.

If the input had minor mistakes, GeoVerify would still return the corrected output. For example, an input of `"9 Sprng St, New Brunswick 8901"` would likely return the above correct output (with perhaps a note in `Message` or just a success with corrected spelling and ZIP).

XML Example:

```
<Root>
  <Service>GeoVerify</Service>
  <Status>Success</Status>
  <TokenCharge>2</TokenCharge>
  <TotalResults>1</TotalResults>
  <Results>
    <AddressIn>9 Spring St</AddressIn>
    <CityStateZipIn>New Brunswick NJ 08901</CityStateZipIn>
    <AddressOut>9 SPRING ST</AddressOut>
    <StreetAddressOut>9 SPRING ST</StreetAddressOut>
    <CityStateZipOut>NEW BRUNSWICK, NJ 08901-2107</CityStateZipOut>
    <FullAddressOut>9 SPRING ST, NEW BRUNSWICK, NJ 08901-
2107</FullAddressOut>
    <City>NEW BRUNSWICK</City>
    <State>NJ</State>
    <Zipcode>08901</Zipcode>
    <Zip4>2107</Zip4>
    <AddressType>S</AddressType>
    <County>MIDDLESEX</County>
    <CountyFIPS>023</CountyFIPS>
    <StateFIPS>34</StateFIPS>
    <TimeZone>EST</TimeZone>
    <DPBC>089012107093</DPBC>
    <CongressionalDistrict>06</CongressionalDistrict>
    <NineDigitZipcode>08901-2107</NineDigitZipcode>
```

```
    <BoxTypes/>
    <BoxType/>
    <BoxNumber/>
    <PreferredCity>NEW BRUNSWICK</PreferredCity>
    <PreferredState>NJ</PreferredState>
    <ErrorMessage/>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.0284</Duration>
  <TimeStamp>2025-04-28T10:42:45-04:00</TimeStamp>
</Root>
```

The XML mirrors the JSON fields. Empty elements like `<BoxType/>` indicate no value (since this is a street address, not a PO box, those fields are empty).

If the address were invalid, `ErrorMessage` would contain details. For example, a bad address might return `<ErrorMessage>Address Not Found</ErrorMessage>` and possibly suggestions if available.

GeoVerifySingle

Overview

GeoVerifySingle is an alternate endpoint for address verification where the entire address is provided in one line, separated by a delimiter. It's designed for batch processing or scenarios where you have a single string containing the full address. You supply the address as one string (including street, city, state, ZIP) and specify how it's delimited (for example, commas). GeoVerifySingle will parse the address and perform the same USPS verification as GeoVerify.

This can be handy if your address data is in a single field or if you want to use a custom separator between parts. Under the hood, it provides similar results to GeoVerify.

Settings and Requirements

- **API Key:** Required (no Bing key needed).
- **Token Usage:** 2 tokens per request (same as GeoVerify).
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geoverifysingle`
- **HTTP Method:** GET
- **Required Parameters:**
  - **`key`** – Your API key.
  - **`address`** – The full address string to verify, with parts separated by a delimiter of your choice. For example: `"9 Spring St, New Brunswick, NJ 08901"` (if using comma as delimiter).
  - **`delimiter`** – The character used as a separator between address parts in the `address` string. In the above example, the delimiter is `,` (comma). You could also use `|` or any other character that does not appear in the address text.
  - **`format`** – `json`, `xml`, or `csv`. Default is `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geoverifysingle?key=YOUR_KEY
    &address=9 Spring St, New Brunswick, NJ 08901&delimiter=,&format=json
```

This verifies the same address as before, but now provided as one line with comma delimiters.

Response Details

GeoVerifySingle returns the exact same fields and format as **GeoVerify**. The only difference is how the input is provided. Once processed, the output fields (`AddressOut`, `City`, `Zipcode`, etc.) are identical to GeoVerify's output for that address.

In other words, the **Results** and their subfields (address components, FIPS codes, etc.) are the same as described in GeoVerify. The service name will be `"GeoVerifySingle"` in the output, but otherwise you can refer to the **GeoVerify** section above for field descriptions.

One minor difference: if parsing fails due to an unexpected format, you might get an error indicating the address couldn't be split properly. Ensure your delimiter is correct and each address part is present.

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string fullAddress = "9 Spring St, New Brunswick, NJ 08901";
string delimiter = ",";

using(HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geoverifysingle?key={key}&address={Uri.EscapeDataString(fullAddress)}"
```

```
                + $"&delimiter={Uri.EscapeDataString(delimiter)}&format=xml";
    var response = client.GetAsync(url).Result;
    string xml = response.Content.ReadAsStringAsync().Result;
    Console.WriteLine(xml);
}
```

JavaScript Example:

```
let fullAddress = "9 Spring St; New Brunswick; NJ 08901";
let delim = ";";
let url = `https://geodata.cdxtech.com/api/geoverifysingle?key=YOUR_KEY`
        +
`&address=${encodeURIComponent(fullAddress)}&delimiter=${encodeURIComponent(d
elim)}&format=json`;
fetch(url)
  .then(res => res.json())
  .then(data => console.log(data.results[0].fullAddressOut));
```

(In this JavaScript example, we used ; as the delimiter between parts.)

VBA Example:

```
Dim key$, addr$
key = "YOUR_KEY"
addr = "9 Spring St|New Brunswick|NJ 08901"  ' using | as delimiter
Dim reqURL$
reqURL = "https://geodata.cdxtech.com/api/geoverifysingle?key=" & key & _
         "&address=" & URLEncode(addr) & "&delimiter=%7C&format=csv"
' (Note: %7C is the URL-encoded '|' character)
Dim http As Object: Set http = CreateObject("MSXML2.XMLHTTP")
http.Open "GET", reqURL, False
http.Send
If http.Status = 200 Then
    Debug.Print http.responseText  ' CSV output of verified address
End If
```

Sample Output

Because GeoVerifySingle's output is the same as GeoVerify, we won't repeat the entire structure. For reference, verifying the address `"9 Spring St, New Brunswick, NJ 08901"` would yield the same JSON/XML shown in the GeoVerify section above, except the top-level `Service` would be `"GeoVerifySingle"`.

For example, JSON output would start as:

```
{
  "service": "GeoVerifySingle",
  "status": "Success",
  "tokenCharge": 2,
  "totalResults": 1,
  "results": [ { ...address fields... } ],
  "usage": { ... },
```

```
  "duration": ...,
  "timeStamp": ...
}
```

And XML output's root would contain `<Service>GeoVerifySingle</Service>` with the rest of the fields identical to the GeoVerify XML example.

The verified address fields (`addressOut`, `cityStateZipOut`, etc.) will match those from GeoVerify for the same input.

GeoDistance

Overview

GeoDistance calculates the straight-line distance between two U.S. ZIP codes (i.e., "as-the-crow-flies" distance). It uses the geographic centroid coordinates of each ZIP Code area to compute the distance. This is useful for quick distance estimates that don't need road routing – for example, to gauge how far apart two ZIP regions are. Distances can be returned in miles, kilometers, or nautical miles.

GeoDistance is limited to U.S. ZIP Codes and does not require an external map API key.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 1 token per request (very low cost).
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geodistance`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`zipcode1`** – The first 5-digit ZIP Code.
  - **`zipcode2`** – The second 5-digit ZIP Code.
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geodistance?key=YOUR_KEY&zipcode1=07869&zipco
de2=07840&format=json
```

This calculates the distance between ZIP code 07869 and 07840.

Response Details

GeoDistance returns the distance between the two ZIP centroids. The response includes:

- **Service:** `"GeoDistance"`.
- **Status:** `"Success"` (or `"Error"` if an input ZIP is invalid or missing).
- **TokenCharge:** 1.
- **Results:** Contains the distance information. It may be presented either as a single value or an object with multiple values depending on the format:
    - In **JSON**, the result includes:
        - **distance:** The numeric distance value.
        - **distanceUnit:** The unit of measurement (`"miles"` by default, or `"kilometers"`/`"nautical miles"` if specified or converted).
        - (In some versions, the JSON might return separate fields like `miles`, `kilometers`, `nauticalMiles` – but based on documentation it appears it returns one value with a unit. The CSV output example suggests one value with unit.)
    - In **XML**, you would see `<Distance>...</Distance>` and `<DistanceUnit>...</DistanceUnit>` within `<Results>`.
    - In **CSV**, the output is a single line with `ZIP1, ZIP2, distance, unit` (e.g., `07869,07840,12.6614,miles`).
- **Usage:** Used/remaining tokens.
- **Duration, TimeStamp:** Execution time and timestamp.

For example, if 07869 and 07840 are about 12.66 miles apart, JSON might return `"distance": 12.6614, "distanceUnit": "miles"`.

If either ZIP code is invalid (not 5 digits or not found), `Status` will be `"Error"` and `Message` will indicate the problem (e.g., “Invalid ZIP Code”).

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string zip1 = "07869";
string zip2 = "07840";

using(HttpClient client = new HttpClient())
{
    string url =
$"https://geodata.cdxtech.com/api/geodistance?key={key}&zipcode1={zip1}&zipco
de2={zip2}&format=xml";
```

```
    HttpResponseMessage resp = client.GetAsync(url).Result;
    string xmlResult = resp.Content.ReadAsStringAsync().Result;
    Console.WriteLine(xmlResult);
}
```

JavaScript Example:

```
async function getDistance(z1, z2) {
  const res = await
fetch(`https://geodata.cdxtech.com/api/geodistance?key=YOUR_KEY&zipcode1=${z1
}&zipcode2=${z2}&format=json`);
  const data = await res.json();
  console.log(`Distance between ${z1} and ${z2}: ${data.distance}
${data.distanceUnit}`);
}
getDistance("07869", "07840");
```

VBA Example:

```
Dim key$, z1$, z2$
key = "YOUR_KEY"
z1 = "07869"
z2 = "07840"
Dim url$
url = "https://geodata.cdxtech.com/api/geodistance?key=" & key & "&zipcode1="
& z1 & "&zipcode2=" & z2 & "&format=csv"
Dim http As Object: Set http = CreateObject("MSXML2.ServerXMLHTTP")
http.Open "GET", url, False
http.Send
If http.Status = 200 Then
    Debug.Print http.ResponseText  ' e.g., 07869,07840,12.6614,miles
End If
```

Sample Output

JSON Example:

```
{
  "service": "GeoDistance",
  "status": "Success",
  "tokenCharge": 1,
  "results": [
    {
      "distance": 12.6614,
      "distanceUnit": "miles"
    }
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.005,
  "timeStamp": "2025-04-28T10:43:15-04:00"
}
```

This indicates the distance between the two ZIPs is ~12.6614 miles. If we requested kilometers instead, the value would be different and `distanceUnit` would read `"kilometers"`. (To get kilometers, one would currently need to convert the result manually, as the API does not have a direct parameter for unit in GeoDistance – it always returns miles in the default implementation. Alternatively, one could convert using 1 mile ≈ 1.60934 km.)

XML Example:

```
<Root>
  <Service>GeoDistance</Service>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Results>
    <Distance>12.6614</Distance>
    <DistanceUnit>miles</DistanceUnit>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.005</Duration>
  <TimeStamp>2025-04-28T10:43:15-04:00</TimeStamp>
</Root>
```

This XML shows the distance and unit. The usage indicates the token consumption.

GeoRadius

Overview

GeoRadius finds all ZIP codes within a given radius of a target ZIP code. For a specified center ZIP and radius distance, it returns a list of ZIP codes that fall within that circle, along with each of those ZIP's coordinates and distance from the center. This is useful for "radius searches," e.g., "find all customers within 10 miles of ZIP 10001." The maximum radius allowed is 200 miles. This function uses the centroid of ZIP areas for the calculations.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 1 token per radius mile up to max radius 200 mile (radius searches are a bit more intensive).
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/georadius`

- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`zipcode`** – The central 5-digit ZIP code from which to search.
  - **`mileage`** – The radius distance from that ZIP, in miles (integer or decimal). (If you want to use kilometers or nautical miles, see `distanceUnit` below.)
  - **`distanceUnit`** – Unit of the radius distance: `miles` (default), `kilometers`, or `nauticalmiles`. If you specify `distanceUnit=kilometers` and `mileage=5`, that means a 5 km radius.
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/georadius?key=YOUR_KEY&zipcode=07869&mileage=5&distanceUnit=miles&format=xml
```

This finds all ZIPs within a 5-mile radius of ZIP 07869.

Response Details

GeoRadius returns a list of ZIP codes and their info that lie within the specified radius. The response includes:

- **Service:** `"GeoRadius"`.
- **Status:** `"Success"` (or `"Error"` if the input ZIP is invalid).
- **TokenCharge:** 3.
- **Results:** An array of ZIP code result entries. Each entry typically includes:
  - **ZipCode:** The ZIP code found within the radius.
  - **City:** The primary city for that ZIP code.
  - **State:** The state abbreviation.
  - **Latitude, Longitude:** The coordinates of the ZIP's centroid.
  - **Distance:** The distance from the center ZIP to this ZIP, in the unit specified.
  - **DistanceUnit:** The unit of that distance (same for all entries, e.g., miles).

In JSON, these might appear as fields within each result object (e.g., `zipCode`, `city`, `state`, `latitude`, `longitude`, `distance`, `distanceUnit`). In XML, each `<Results>` entry would contain those child elements. In CSV, each line has `centerZIP, foundZIP, distance, unit` or possibly `foundZIP, city, state, latitude, longitude, distance, unit`. The documentation implies CSV includes lat/long and distance: e.g., one of the search results shows lines like `07869,2.1835,miles` (which suggests a format like `centerZIP, distance, unit` for each result or incomplete snippet) – but more complete context suggests CSV might output multiple columns (likely `ZIP, City, State, Latitude, Longitude, Distance, Unit`). For clarity, let's assume JSON/XML provide full detail, and CSV provides a row per ZIP with key info.

- **Usage:** Tokens used/remaining.

- **Duration, TimeStamp:** Execution time and timestamp.

For instance, if ZIP 07869 (Randolph, NJ) within 5 miles might include ZIP 07876 (Succasunna, NJ), 07950 (Morristown, NJ) etc., each with their distance from 07869 (e.g., 2.18 miles, 2.79 miles, etc., as seen in the sample below).

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string centerZip = "07869";
string radius = "5";  // 5 mile radius
using(HttpClient client = new HttpClient())
{
    string url =
$"https://geodata.cdxtech.com/api/georadius?key={key}&zipcode={centerZip}&mil
eage={radius}&format=json";
    string json = client.GetStringAsync(url).Result;
    Console.WriteLine(json);
}
```

JavaScript Example:

```
async function radiusSearch(centerZip, miles) {
  let url =
`https://geodata.cdxtech.com/api/georadius?key=YOUR_KEY&zipcode=${centerZip}&
mileage=${miles}&format=json`;
  let res = await fetch(url);
  let data = await res.json();
  data.results.forEach(z => {
    console.log(`${z.zipCode} (${z.city}, ${z.state}) is
${z.distance.toFixed(2)} ${z.distanceUnit} away`);
  });
}
radiusSearch("07869", 5);
```

VBA Example:

```
Dim key$, centerZip$
key = "YOUR_KEY"
centerZip = "07869"
Dim url$
url = "https://geodata.cdxtech.com/api/georadius?key=" & key & "&zipcode=" &
centerZip & "&mileage=5&distanceUnit=miles&format=csv"
Dim http As Object: Set http = CreateObject("MSXML2.XMLHTTP")
http.Open "GET", url, False
http.Send
If http.Status = 200 Then
    ' Each line of the CSV will be like:
ZIP,City,State,Latitude,Longitude,Distance,DistanceUnit
    Dim lines As Variant: lines = Split(http.responseText, vbNewLine)
    Dim i As Integer
```

```
    For i = LBound(lines) To UBound(lines)
        Debug.Print lines(i)
    Next
End If
```

Sample Output

JSON Example: (ZIPs within 3 miles of 07869)


```
{
  "service": "GeoRadius",
  "status": "Success",
  "tokenCharge": 3,
  "totalResults": 4,
  "results": [
    {
      "zipCode": "07869",
      "city": "RANDOLPH",
      "state": "NJ",
      "latitude": 40.842115,
      "longitude": -74.58229,
      "distance": 0.0,
      "distanceUnit": "miles"
    },
    {
      "zipCode": "07876",
      "city": "SUCCASUNNA",
      "state": "NJ",
      "latitude": 40.860279,
      "longitude": -74.652504,
      "distance": 2.1835,
      "distanceUnit": "miles"
    },
    {
      "zipCode": "07870",
      "city": "ROXBURY TOWNSHIP",
      "state": "NJ",
      "latitude": 40.880137,
      "longitude": -74.6264,
      "distance": 2.6746,
      "distanceUnit": "miles"
    },
    {
      "zipCode": "07950",
      "city": "MORRIS PLAINS",
      "state": "NJ",
      "latitude": 40.8225,
      "longitude": -74.575269,
      "distance": 2.7978,
      "distanceUnit": "miles"
    }
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.012,
  "timeStamp": "2025-04-28T10:43:45-04:00"
```

```
}
```

In this JSON, we searched for a 3-mile radius around 07869. The result includes 07869 itself (distance 0), and a few surrounding ZIP codes with their distances. Note the `city` field – sometimes it might show the township or primary locale name for that ZIP. All distances are in miles as requested.

XML Example:

```
<Root>
  <Service>GeoRadius</Service>
  <Status>Success</Status>
  <TokenCharge>3</TokenCharge>
  <TotalResults>4</TotalResults>
  <Results>
    <ZipCode>07869</ZipCode>
    <City>RANDOLPH</City>
    <State>NJ</State>
    <Latitude>40.842115</Latitude>
    <Longitude>-74.58229</Longitude>
    <Distance>0</Distance>
    <DistanceUnit>miles</DistanceUnit>
  </Results>
  <Results>
    <ZipCode>07876</ZipCode>
    <City>SUCCASUNNA</City>
    <State>NJ</State>
    <Latitude>40.860279</Latitude>
    <Longitude>-74.652504</Longitude>
    <Distance>2.1835</Distance>
    <DistanceUnit>miles</DistanceUnit>
  </Results>
  <Results>
    <ZipCode>07870</ZipCode>
    <City>ROXBURY TOWNSHIP</City>
    <State>NJ</State>
    <Latitude>40.880137</Latitude>
    <Longitude>-74.6264</Longitude>
    <Distance>2.6746</Distance>
    <DistanceUnit>miles</DistanceUnit>
  </Results>
  <Results>
    <ZipCode>07950</ZipCode>
    <City>MORRIS PLAINS</City>
    <State>NJ</State>
    <Latitude>40.8225</Latitude>
    <Longitude>-74.575269</Longitude>
    <Distance>2.7978</Distance>
    <DistanceUnit>miles</DistanceUnit>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.012</Duration>
```

```
  <TimeStamp>2025-04-28T10:43:45-04:00</TimeStamp>
</Root>
```

Each `<Results>` block corresponds to one ZIP in range. The order of results is not guaranteed to be sorted by distance (though often it might be from nearest to farthest). If needed, you can sort them client-side.

GeoFindZip

Overview

GeoFindZip finds ZIP codes by city and/or county. You provide a state along with a city name and/or county name, and it returns all ZIP codes associated with that city or county. It's a lookup service to get ZIP codes for a given location description. For example, you can get all ZIP codes for a city across its multiple ZIP areas, or all ZIP codes in a county.

This service is somewhat superseded by the more specific GeoZipListCity/County/State functions (documented later), but GeoFindZip provides a flexible combined search.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 1 token per request.
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geofindzip`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`state`** – The state for the search (2-letter state abbreviation).
  - **`city`** – (Optional) City name to find ZIPs for.
  - **`county`** – (Optional) County name to find ZIPs for.

You should supply at least a `city` or a `county` (or both). If both are provided, the query is effectively city *within* that county. If only city is provided (with state), it returns ZIPs for that city (across any counties). If only county is provided (with state), it returns all ZIPs in that county.

- **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Requests:

- **City only:** `.../geofindzip?key=YOUR_KEY&state=NJ&city=Morristown`
- **County only:** `.../geofindzip?key=YOUR_KEY&state=NJ&county=Morris`
- **City in County:** `.../geofindzip?key=YOUR_KEY&state=NJ&city=Morristown&county=Morris`

The last example would ensure we get ZIPs for Morristown in Morris County, NJ (useful if the city name isn't unique in NJ).

Response Details

GeoFindZip returns a list of ZIP codes matching the query criteria. The response includes:

- **Service:** `"GeoFindZip"`.
- **Status:** `"Success"` if found, `"Error"` if state is invalid or no results.
- **TokenCharge:** 1.
- **Results:** An array of ZIP code entries. Each entry likely includes:
  - **ZipCode:** The ZIP code.
  - **City:** The city name associated with that ZIP (often the city you searched, or one of them if multiple cities share a ZIP).
  - **State:** State abbreviation.
  - **County:** County name.

For example, searching state=NJ, city=Morristown might return: 07960, 07961, 07962, 07963 (the ZIP codes for Morristown) each with city "MORRISTOWN", state "NJ", and county "MORRIS".

- **Usage:** Token usage summary.
- **Duration, TimeStamp:** Execution info.

If no ZIP is found for the given criteria (e.g., misspelled city or wrong combo), `TotalResults` might be 0 and `Status` could be `"Success"` with an empty result set or it might return an error message stating no results (the behavior may vary; often an empty successful response is given).

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string state = "NJ";
string city = "Morristown";

using(HttpClient client = new HttpClient())
{
```

```
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geofindzip?key={key}&state={state}&city={Uri.EscapeDataString(city)}&f
ormat=csv";
    string csvResult = client.GetStringAsync(url).Result;
    Console.WriteLine(csvResult);
}
```

JavaScript Example:

```
fetch('https://geodata.cdxtech.com/api/geofindzip?key=YOUR_KEY&state=NJ&count
y=Morris&format=json')
  .then(res => res.json())
  .then(data => {
    data.results.forEach(entry => {
      console.log(entry.zipCode, entry.city, entry.county, entry.state);
    });
  });
```

(The above fetch gets all ZIPs in Morris County, NJ.)

VBA Example:

```
Dim key$, st$, cty$
key = "YOUR_KEY"
st = "NJ"
cty = "Morristown"
Dim url$
url = "https://geodata.cdxtech.com/api/geofindzip?key=" & key & "&state=" &
st & "&city=" & URLEncode(cty) & "&format=xml"
Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", url, False
xhr.Send
If xhr.Status = 200 Then
    Dim xmlDoc As Object: Set xmlDoc = CreateObject("MSXML2.DOMDocument")
    xmlDoc.LoadXML xhr.ResponseText
    Dim zips As Object: Set zips = xmlDoc.SelectNodes("//Results/ZipCode")
    Dim i As Integer
    For i = 0 To zips.Length-1
        Debug.Print zips.Item(i).Text
    Next
End If
```

Sample Output

JSON Example: (Find ZIPs for city Morristown, NJ)

```
{
  "service": "GeoFindZip",
  "status": "Success",
  "tokenCharge": 1,
  "totalResults": 4,
  "results": [
```

```
    { "zipCode": "07960", "city": "MORRISTOWN", "county": "MORRIS", "state":
"NJ" },
    { "zipCode": "07961", "city": "MORRISTOWN", "county": "MORRIS", "state":
"NJ" },
    { "zipCode": "07962", "city": "MORRISTOWN", "county": "MORRIS", "state":
"NJ" },
    { "zipCode": "07963", "city": "MORRISTOWN", "county": "MORRIS", "state":
"NJ" }
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.010,
  "timeStamp": "2025-04-28T10:44:15-04:00"
}
```


These four ZIP codes correspond to various PO boxes and delivery areas in Morristown, NJ. All share the same city, county, state in this case.

XML Example:


```
<Root>
  <Service>GeoFindZip</Service>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <TotalResults>4</TotalResults>
  <Results>
    <ZipCode>07960</ZipCode>
    <City>MORRISTOWN</City>
    <County>MORRIS</County>
    <State>NJ</State>
  </Results>
  <Results>
    <ZipCode>07961</ZipCode>
    <City>MORRISTOWN</City>
    <County>MORRIS</County>
    <State>NJ</State>
  </Results>
  <Results>
    <ZipCode>07962</ZipCode>
    <City>MORRISTOWN</City>
    <County>MORRIS</County>
    <State>NJ</State>
  </Results>
  <Results>
    <ZipCode>07963</ZipCode>
    <City>MORRISTOWN</City>
    <County>MORRIS</County>
    <State>NJ</State>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.010</Duration>
  <TimeStamp>2025-04-28T10:44:15-04:00</TimeStamp>
</Root>
```

If we had searched by county (e.g., state=NJ, county=Morris, without specifying city), the results would list all ZIPs in Morris County. That could be a long list (since Morris County has many ZIPs), but the format is the same: each result with ZipCode, City, County, State. The City field would show the primary city for each ZIP code.

GeoZipListCity

Overview

GeoZipListCity returns the complete list of ZIP codes for a specific city. You provide the city name and state (and optionally county to disambiguate if needed), and it gives all ZIP codes associated with that city, along with latitude, longitude, and other details for each ZIP. It's similar to GeoFindZip (city) but is a dedicated function that may include more details per ZIP.

GeoZipListCity is especially useful when you need not just the ZIP codes but also their geolocation and perhaps population or other data (though primarily it's for listing ZIPs).

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 1 token per request.
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geoziplistcity`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`state`** – 2-letter state abbreviation.
  - **`city`** – City name. (Not case-sensitive; spaces should be URL-encoded if present.)
  - **`county`** – (Optional) County name. If provided, the results will be filtered to that county. This is useful if the city name exists in multiple counties within the state. If the city is unique in the state, county can be left blank or omitted.
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geoziplistcity?key=YOUR_KEY&state=NJ&city=Mor
ristown&county=Morris&format=json
```

This fetches all ZIP codes for Morristown, NJ in Morris County.

Response Details

GeoZipListCity returns a list of ZIP code entries for the specified city. Each entry typically includes:

- **ZipCode:** The ZIP code.
- **City:** City name (should match the query city, albeit in uppercase).
- **County:** County name for that ZIP.
- **State:** State abbreviation.
- **Latitude:** Latitude of the ZIP code's centroid.
- **Longitude:** Longitude of the ZIP's centroid.

It might also include additional fields such as type of ZIP (e.g. if it's a PO Box ZIP) or population, but the primary documented ones are those above. The documentation references a separate data definitions document for detailed fields, so likely the output is quite similar to what GeoFindZip returned, plus coordinates.

- **Usage, Duration, Timestamp:** Standard metadata.

If the city is not found or has no ZIP codes (which is unlikely since if a city exists it has at least one ZIP), `Status` could be `"Error"` or simply `totalResults=0`.

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string state = "NJ";
string city = "Morristown";
string county = "Morris";

using(HttpClient client = new HttpClient())
{
    string url =
$"https://geodata.cdxtech.com/api/geoziplistcity?key={key}&state={state}"
              +
$"&city={Uri.EscapeDataString(city)}&county={Uri.EscapeDataString(county)}&fo
rmat=xml";
    string xml = client.GetStringAsync(url).Result;
    Console.WriteLine(xml);
}
```

JavaScript Example:

```
fetch('https://geodata.cdxtech.com/api/geoziplistcity?key=YOUR_KEY&state=NJ&c
ity=Morristown&format=json')
  .then(res => res.json())
  .then(data => {
    data.results.forEach(z => {
      console.log(z.zipCode, z.latitude, z.longitude);
    });
  });
```

(This example omits county; if Morristown existed in multiple NJ counties, the result might include ZIPs from both, but in reality Morristown is primarily in Morris County.)

VBA Example:

```
Dim key$, state$, city$
key = "YOUR_KEY"
state = "NJ"
city = "Morristown"
Dim url$
url = "https://geodata.cdxtech.com/api/geoziplistcity?key=" & key & "&state="
& state & "&city=" & URLEncode(city) & "&format=csv"
Dim http As Object: Set http = CreateObject("MSXML2.XMLHTTP")
http.Open "GET", url, False
http.Send
If http.Status = 200 Then
    ' CSV columns: ZipCode,City,County,State,Latitude,Longitude
    Debug.Print http.responseText
End If
```

Sample Output

JSON Example: (ZIP list for Morristown, NJ)


```
{
  "service": "GeoZipListCity",
  "status": "Success",
  "tokenCharge": 1,
  "totalResults": 4,
  "results": [
    { "zipCode": "07960", "city": "MORRISTOWN", "county": "MORRIS", "state":
"NJ", "latitude": 40.7964, "longitude": -74.4840 },
    { "zipCode": "07961", "city": "MORRISTOWN", "county": "MORRIS", "state":
"NJ", "latitude": 40.7964, "longitude": -74.4840 },
    { "zipCode": "07962", "city": "MORRISTOWN", "county": "MORRIS", "state":
"NJ", "latitude": 40.7964, "longitude": -74.4840 },
    { "zipCode": "07963", "city": "MORRISTOWN", "county": "MORRIS", "state":
"NJ", "latitude": 40.7964, "longitude": -74.4840 }
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.011,
  "timeStamp": "2025-04-28T10:44:45-04:00"
}
```

The latitude/longitude here are identical for all four because they all correspond to essentially the same area (Morristown's centroid). These ZIPs (07960-07963) are specific to Morristown (07961-63 are PO box ZIPs or unique ZIPs, hence same lat/long as the main 07960).

XML Example:


```
<Root>
  <Service>GeoZipListCity</Service>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <TotalResults>4</TotalResults>
  <Results>
    <ZipCode>07960</ZipCode>
    <City>MORRISTOWN</City>
    <County>MORRIS</County>
    <State>NJ</State>
    <Latitude>40.7964</Latitude>
    <Longitude>-74.4840</Longitude>
  </Results>
  <Results>
    <ZipCode>07961</ZipCode>
    <City>MORRISTOWN</City>
    <County>MORRIS</County>
    <State>NJ</State>
    <Latitude>40.7964</Latitude>
    <Longitude>-74.4840</Longitude>
  </Results>
  <Results>
    <ZipCode>07962</ZipCode>
    <City>MORRISTOWN</City>
    <County>MORRIS</County>
    <State>NJ</State>
    <Latitude>40.7964</Latitude>
    <Longitude>-74.4840</Longitude>
  </Results>
  <Results>
    <ZipCode>07963</ZipCode>
    <City>MORRISTOWN</City>
    <County>MORRIS</County>
    <State>NJ</State>
    <Latitude>40.7964</Latitude>
    <Longitude>-74.4840</Longitude>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.011</Duration>
  <TimeStamp>2025-04-28T10:44:45-04:00</TimeStamp>
</Root>
```


If the city spans multiple counties and we didn't specify a county, the list might include ZIPs from each relevant county but still labeled with their respective county in each result.

GeoZipListCounty

Overview

GeoZipListCounty provides the list of all ZIP codes within a given county (and state). You specify the state and county, and it returns every ZIP code in that county, along with associated city and lat/long for each. This is useful for getting a comprehensive ZIP list for county-based analyses.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 10 token per request.
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geoziplistcounty`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`state`** – 2-letter state abbreviation.
  - **`county`** – County name (not case-sensitive; can include spaces if applicable, e.g., `"Los Angeles"`).
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geoziplistcounty?key=YOUR_KEY&state=NJ&county=Morris&format=csv
```

This will retrieve all ZIP codes in Morris County, NJ.

Response Details

GeoZipListCounty returns all ZIP codes for the specified county. Each result entry includes:

- **ZipCode:** The ZIP code.
- **City:** A city associated with that ZIP (likely the primary city or one city name for that ZIP).
- **County:** County name (should match the query county for all).
- **State:** State abbreviation.
- **Latitude, Longitude:** Coordinates of the ZIP's centroid.

Expect potentially dozens of results depending on the county size. For instance, Morris County, NJ has many ZIP codes.

If the county name or state is invalid, no results will be returned (Status might be Success with 0 results, or an Error message).

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string state = "NJ";
string county = "Morris";

using(HttpClient client = new HttpClient())
{
    string url =
$"https://geodata.cdxtech.com/api/geoziplistcounty?key={key}&state={state}&co
unty={Uri.EscapeDataString(county)}&format=json";
    string json = client.GetStringAsync(url).Result;
    Console.WriteLine(json.Substring(0, 200) + "..."); // print first part
for brevity
}
```

JavaScript Example:

```
fetch('https://geodata.cdxtech.com/api/geoziplistcounty?key=YOUR_KEY&state=NJ
&county=Morris&format=json')
  .then(res => res.json())
  .then(data => {
    console.log(`Found ${data.totalResults} ZIP codes in Morris County,
NJ.`);
    // Print first 5:
    data.results.slice(0,5).forEach(z => {
      console.log(z.zipCode, z.city);
    });
  });
```

VBA Example:

```
Dim key$, st$, county$
key = "YOUR_KEY"
st = "NJ"
county = "Morris"
Dim url$
url = "https://geodata.cdxtech.com/api/geoziplistcounty?key=" & key &
"&state=" & st & "&county=" & URLEncode(county) & "&format=csv"
Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", url, False
xhr.Send
If xhr.Status = 200 Then
    Dim csv As String: csv = xhr.ResponseText
    ' Split and output first line (header) and first data line as example
    Dim lines: lines = Split(csv, vbNewLine)
    Debug.Print lines(0)
    If UBound(lines) >= 1 Then Debug.Print lines(1)
End If
```

Sample Output

JSON Example: (First few ZIPs in Morris County, NJ)

```
{
  "service": "GeoZipListCounty",
  "status": "Success",
  "tokenCharge": 1,
  "totalResults": 39,
  "results": [
    { "zipCode": "07005", "city": "BOONTON", "county": "MORRIS", "state":
"NJ", "latitude": 40.904, "longitude": -74.417 },
    { "zipCode": "07034", "city": "LAKE HIAWATHA", "county": "MORRIS",
"state": "NJ", "latitude": 40.882, "longitude": -74.378 },
    { "zipCode": "07045", "city": "MONTVILLE", "county": "MORRIS", "state":
"NJ", "latitude": 40.912, "longitude": -74.360 },
    { "zipCode": "07046", "city": "MOUNTAIN LAKES", "county": "MORRIS",
"state": "NJ", "latitude": 40.890, "longitude": -74.436 },
    { "zipCode": "07054", "city": "PARSIPPANY", "county": "MORRIS", "state":
"NJ", "latitude": 40.859, "longitude": -74.425 },
    ... (34 more) ...
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.020,
  "timeStamp": "2025-04-28T10:45:15-04:00"
}
```

The above shows 5 of 39 results for Morris County. Each entry lists the ZIP and a city (usually one of the main municipalities for that ZIP). All have county MORRIS, NJ and coordinates.

XML Example: (first and last entry example)

```
<Root>
  <Service>GeoZipListCounty</Service>
  <Status>Success</Status>
  <TokenCharge>10</TokenCharge>
```

```
  <TotalResults>39</TotalResults>
  <Results>
    <ZipCode>07005</ZipCode>
    <City>BOONTON</City>
    <County>MORRIS</County>
    <State>NJ</State>
    <Latitude>40.904</Latitude>
    <Longitude>-74.417</Longitude>
  </Results>
  ...
  <Results>
    <ZipCode>07981</ZipCode>
    <City>WHIPPANY</City>
    <County>MORRIS</County>
    <State>NJ</State>
    <Latitude>40.822</Latitude>
    <Longitude>-74.425</Longitude>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.020</Duration>
  <TimeStamp>2025-04-28T10:45:15-04:00</TimeStamp>
</Root>
```

This XML snippet shows the first ZIP (07005 Boonton) and last ZIP (07981 Whippany) out of the 39 returned for Morris County.

GeoZipListState

Overview

GeoZipListState provides the list of all ZIP codes in a given U.S. state. It requires only the state as input and will return every ZIP code in that state, typically along with each ZIP's primary city and possibly other info. This is essentially a full state ZIP code listing.

Be cautious: for large states (like California, Texas, etc.), the number of ZIP codes is large (hundreds), so the response will be correspondingly large.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 100 token per request.
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geozipliststate`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`state`** – 2-letter state abbreviation. (This must be a valid US state or territory code.)
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geozipliststate?key=YOUR_KEY&state=NJ&format=xml
```

This gets all ZIP codes in New Jersey.

Response Details

GeoZipListState returns all ZIP codes in the specified state. Each result entry likely includes:

- **ZipCode:** The ZIP code.
- **City:** One primary city for that ZIP (possibly the USPS preferred city).
- **County:** The county in which the ZIP code is primarily located. (If a ZIP spans counties, it might list one or all – likely one per entry, perhaps the main county).
- **State:** State abbreviation.
- **Latitude, Longitude:** Coordinates of the ZIP centroid.

For example, for NJ (New Jersey), you would get every ZIP from 07001 (Avenel) through 089xx, Newark, etc., up to Newark's last ZIP and beyond. The output can easily be several hundred lines (NJ has ~730 ZIP codes).

- **Usage, Duration, Timestamp:** As usual.

Note: Because of the size, CSV format might be useful if you plan to import into a database or spreadsheet.

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string state = "NJ";

using(HttpClient client = new HttpClient())
{
    string url =
$"https://geodata.cdxtech.com/api/geozipliststate?key={key}&state={state}&format=csv";
    string csvData = client.GetStringAsync(url).Result;
```

```
    // For example, count the lines (ZIP codes):
    int zipCount = csvData.Split('\n').Length - 1; // minus header
    Console.WriteLine($"NJ has {zipCount} ZIP codes.");
}
```

JavaScript Example:

```
fetch('https://geodata.cdxtech.com/api/geozipliststate?key=YOUR_KEY&state=NJ&
format=json')
  .then(res => res.json())
  .then(data => {
    console.log(`First ZIP in NJ: ${data.results[0].zipCode} -
${data.results[0].city}`);
    console.log(`Total ZIP codes in NJ: ${data.totalResults}`);
  });
```

VBA Example:

```
Dim key$, st$
key = "YOUR_KEY"
st = "NJ"
Dim url$
url = "https://geodata.cdxtech.com/api/geozipliststate?key=" & key &
"&state=" & st & "&format=xml"
Dim xhr As Object: Set xhr = CreateObject("MSXML2.ServerXMLHTTP")
xhr.Open "GET", url, False
xhr.Send
If xhr.Status = 200 Then
    Dim xmlDoc As Object: Set xmlDoc = CreateObject("MSXML2.DOMDocument")
    xmlDoc.LoadXML xhr.ResponseText
    Dim count As Integer
    count = xmlDoc.SelectNodes("//Results").Length
    MsgBox st & " has " & count & " ZIP codes."
End If
```

Sample Output

JSON Example: (abridged – first 3 and last 2 ZIPs of NJ)

```
{
  "service": "GeoZipListState",
  "status": "Success",
  "tokenCharge": 100,
  "totalResults": 759,
  "results": [
    { "zipCode": "07001", "city": "AVENEL", "county": "MIDDLESEX", "state":
"NJ", "latitude": 40.582, "longitude": -74.285 },
    { "zipCode": "07002", "city": "BAYONNE", "county": "HUDSON", "state":
"NJ", "latitude": 40.668, "longitude": -74.114 },
    { "zipCode": "07003", "city": "BLOOMFIELD", "county": "ESSEX", "state":
"NJ", "latitude": 40.806, "longitude": -74.188 },
    ...,
    { "zipCode": "08904", "city": "HIGHLAND PARK", "county": "MIDDLESEX",
"state": "NJ", "latitude": 40.500, "longitude": -74.425 },
```

```
    { "zipCode": "08906", "city": "NEW BRUNSWICK", "county": "MIDDLESEX",
"state": "NJ", "latitude": 40.486, "longitude": -74.451 }
  ],
  "usage": { "used": 100, "remaining": 1000 },
  "duration": 0.045,
  "timeStamp": "2025-04-28T10:45:45-04:00"
}
```


This shows the first few ZIPs (Avenel, Bayonne, Bloomfield) and last ones (Highland Park, New Brunswick) in NJ. There are 759 results, which matches the number of active ZIPs in NJ.

XML Example: (first and last entries for NJ)


```
<Root>
  <Service>GeoZipListState</Service>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <TotalResults>759</TotalResults>
  <Results>
    <ZipCode>07001</ZipCode>
    <City>AVENEL</City>
    <County>MIDDLESEX</County
```


GeoRace

Overview

GeoRace provides demographic breakdown by race for the population of a given U.S. ZIP Code, based on U.S. Census data (2010 Census ZIP Code Tabulation Areas). It returns population counts and percentages for racial categories (such as White, Black or African American, Native American, Asian, Pacific Islander, Other, and Two or more races) within the ZIP. This is useful for understanding the racial composition of the ZIP's population.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 1 token per request.
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/georace`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - zipcoe – 5 digit zipcode.
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

https://geodata.cdxtech.com/api/georace?key=YOUR_KEY&zipcode=90210&format=json

This would fetch the race composition for ZIP code 90210 (Beverly Hills).

Response Details

GeoRace returns the population counts and percentage share for each race category in the ZIP. The output includes:

- **Service:** GeoRace
- **Status:** Success"` (or `"Error"` if the ZIP code is invalid or data unavailable).
- **Token** Charge: 1
- **Results:** An object (or array with one element) containing: zip code, city, state and population by race.including whitePopulation, blackPopulation, nativePopulation, asianPopulation, pacificPopulation, otherPopulation, multiRacePopulation.by count and percentage.

In JSON, you might see a structure like:

```json
{
  "zipCode": "90210",
  "city": "BEVERLY HILLS",
  "state": "CA",
  "whitePopulation": 16614,
  "whitePercentage": 82.1,
  "blackPopulation": 746,
  "blackPercentage": 3.7,
  "nativePopulation": 30,
  "nativePercentage": 0.1,
  "asianPopulation": 1801,
  "asianPercentage": 8.9,
  "pacificPopulation": 5,
  "pacificPercentage": 0.0,
  "otherPopulation": 483,
  "otherPercentage": 2.4,
  "multiRacePopulation": 558,
  "multiRacePercentage": 2.8,
  "totalPopulation": 20237
}
```

(The above numbers are illustrative, not actual, but demonstrate the format.)

- **Usage, Duration, TimeStamp:** Usage and timing info.

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string zip = "90210";
using(var client = new HttpClient())
{
    string url =
$"https://geodata.cdxtech.com/api/georace?key={key}&zipcode={zip}&format=xml"
;
    string xmlResult = client.GetStringAsync(url).Result;
    Console.WriteLine(xmlResult);
}
```

JavaScript Example:

```
fetch('https://geodata.cdxtech.com/api/georace?key=YOUR_KEY&zipcode=90210&for
mat=json')
  .then(res => res.json())
  .then(data => {
    console.log(`Total pop of 90210: ${data.totalPopulation}`);
    console.log(`White: ${data.whitePercentage}% (${data.whitePopulation}),
Asian: ${data.asianPercentage}% (${data.asianPopulation})`);
  });
```

VBA Example:

```
Dim key$, zip$
key = "YOUR_KEY"
zip = "90210"
Dim url$
url = "https://geodata.cdxtech.com/api/georace?key=" & key & "&zipcode=" &
zip & "&format=csv"
Dim http As Object: Set http = CreateObject("MSXML2.ServerXMLHTTP")
http.Open "GET", url, False
http.Send
If http.Status = 200 Then
    Debug.Print http.responseText  ' CSV with race data for the ZIP
End If
```

Sample Output

JSON Example: (Race breakdown for ZIP 07869, Randolph, NJ – data from 2010)

```
{
  "service": "GeoRace",
  "status": "Success",
```

```
  "tokenCharge": 1,
  "results": [
    {
      "zipCode": "07869",
      "city": "RANDOLPH",
      "state": "NJ",
      "whitePopulation": 21246,
      "whitePercentage": 82.23,
      "blackPopulation": 2879,
      "blackPercentage": 11.14,
      "nativePopulation": 75,
      "nativePercentage": 0.29,
      "asianPopulation": 885,
      "asianPercentage": 3.43,
      "pacificPopulation": 14,
      "pacificPercentage": 0.05,
      "otherPopulation": 739,
      "otherPercentage": 2.86,
      "multiRacePopulation": 0,
      "multiRacePercentage": 0.00,
      "totalPopulation": 25838
    }
  ]
}
```


In this example, ZIP 07869 had 25,838 people. About 82.23% were White, 11.14% Black, 3.43% Asian, etc., which sum up to 100%. (The multi-race category is 0% here, meaning negligible or none reported multiple races in that ZIP in 2010.)

XML Example:


```
<Root>
  <Service>GeoRace</Service>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Results>
    <ZipCode>07869</ZipCode>
    <City>RANDOLPH</City>
    <State>NJ</State>
    <WhitePopulation>21246</WhitePopulation>
    <WhitePercentage>82.23</WhitePercentage>
    <BlackPopulation>2879</BlackPopulation>
    <BlackPercentage>11.14</BlackPercentage>
    <NativePopulation>75</NativePopulation>
    <NativePercentage>0.29</NativePercentage>
    <AsianPopulation>885</AsianPopulation>
    <AsianPercentage>3.43</AsianPercentage>
    <PacificPopulation>14</PacificPopulation>
    <PacificPercentage>0.05</PacificPercentage>
    <OtherPopulation>739</OtherPopulation>
    <OtherPercentage>2.86</OtherPercentage>
    <MultiRacePopulation>0</MultiRacePopulation>
    <MultiRacePercentage>0.00</MultiRacePercentage>
    <TotalPopulation>25838</TotalPopulation>
  </Results>
```

```
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.015</Duration>
  <TimeStamp>2025-04-28T10:46:15-04:00</TimeStamp>
</Root>
```

GeoGender

Overview

GeoGender provides the gender distribution for a given U.S. ZIP Code, based on Census 2010 data. It returns the male and female population counts, as well as the percentage of the population that is male vs female. This is useful for understanding the gender makeup of the ZIP's population.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 1 token per request.
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geogender`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`zipcode`** – 5-digit ZIP code to look up.
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geogender?key=YOUR_KEY&zipcode=90210&format=json
```

Response Details

GeoGender returns the male/female population counts for the ZIP and their percentages of the total population. Fields include:

- **Service:** `"GeoGender"`.

- **Results:** Containing:
  - **zipCode, city, state:** Identification of the area (similar to GeoRace).
  - **malePopulation, femalePopulation:** Number of males and females in the ZIP (2010 Census).
  - **malePercentage, femalePercentage:** Each gender as a percentage of the total.
  - **totalPopulation:** Total population (male + female).

For example, in JSON:

```
{
  "zipCode": "90210",
  "city": "BEVERLY HILLS",
  "state": "CA",
  "malePopulation": 9700,
  "malePercentage": 47.9,
  "femalePopulation": 10537,
  "femalePercentage": 52.1,
  "totalPopulation": 20237
}
```

- **Usage, Duration, Timestamp:** Standard metadata.

Coding Examples

C# Example:

```
string url =
$"https://geodata.cdxtech.com/api/geogender?key=YOUR_KEY&zipcode=10001&format
=xml";
string xml = new WebClient().DownloadString(url);
Console.WriteLine(xml);
```

JavaScript Example:

```
fetch('https://geodata.cdxtech.com/api/geogender?key=YOUR_KEY&zipcode=10001&f
ormat=json')
  .then(res => res.json())
  .then(data => {
    console.log(`${data.zipCode}: Males=${data.malePercentage}%,
Females=${data.femalePercentage}%`);
  });
```

VBA Example:

```
Dim key$, zip$
key = "YOUR_KEY"
zip = "10001"
Dim url$
url = "https://geodata.cdxtech.com/api/geogender?key=" & key & "&zipcode=" &
zip & "&format=csv"
Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", url, False
```

```
xhr.Send
If xhr.Status = 200 Then
    Debug.Print xhr.responseText  ' e.g., 10001,NEW YORK,NY,
MalePop,FemalePop,TotalPop,Male%,Female%
End If
```

Sample Output

JSON Example: (Gender breakdown for ZIP 07869, Randolph, NJ)

```
{
  "service": "GeoGender",
  "status": "Success",
  "tokenCharge": 1,
  "results": [
    {
      "zipCode": "07869",
      "city": "RANDOLPH",
      "state": "NJ",
      "malePopulation": 12427,
      "malePercentage": 49.14,
      "femalePopulation": 12864,
      "femalePercentage": 50.86,
      "totalPopulation": 25291
    }
  ]
}
```

In this example, ZIP 07869 had 12,427 males (49.14%) and 12,864 females (50.86%), for a total of 25,291.

XML Example:

```
<Root>
  <Service>GeoGender</Service>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Results>
    <ZipCode>07869</ZipCode>
    <City>RANDOLPH</City>
    <State>NJ</State>
    <MalePopulation>12427</MalePopulation>
    <MalePercentage>49.14</MalePercentage>
    <FemalePopulation>12864</FemalePopulation>
    <FemalePercentage>50.86</FemalePercentage>
    <TotalPopulation>25291</TotalPopulation>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.010</Duration>
  <TimeStamp>2025-04-28T10:46:45-04:00</TimeStamp>
</Root>
```

GeoGeneral

Overview

GeoGeneral returns general geographic and administrative data for a given U.S. ZIP Code. It provides location details such as the primary city name, county, state, latitude/longitude, and other related information (like time zone and area code). This is a convenient way to retrieve basic profile data about a ZIP Code.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 1 token per request.
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`).

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geogeneral`
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`zipcode`** – 5-digit ZIP code.
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geogeneral?key=YOUR_KEY&zipcode=10001&format=json
```

Response Details

GeoGeneral returns a variety of general fields for the ZIP Code, which may include:

- **Service:** `"GeoGeneral"`.
- **Results:** Containing fields such as:
  - **zipCode, city, state:** The ZIP code, primary city name, and state.

- **county:** County name in which the ZIP lies.
- **countyFIPS, stateFIPS:** The FIPS codes for the county and state.
- **latitude, longitude:** Coordinates of the ZIP's geographic center.
- **timeZone:** Time zone of the area (e.g., `EST` for Eastern Standard).
- **areaCodes:** Telephone area code(s) serving that ZIP (if multiple, separated by `/`).
- **region:** Census region of the U.S. (e.g., Northeast, Midwest, South, West).
- **division:** Census division (a subdivision of regions, e.g., Middle Atlantic, Pacific).
- **daylightSavings:** A flag (maybe `Y`/`N`) indicating if the area observes daylight savings time.
- **MSA, MSAName:** (If available) The metropolitan statistical area code and name that the ZIP is part of.
- **landArea, waterArea:** Land and water area of the ZIP (likely in square miles).
- **population:** *If* included, the total population (though detailed demographics are in GeoDemographics).

Note: The exact fields in GeoGeneral can be extensive. It essentially aggregates identification and geographic attributes of the ZIP.

For example, JSON for ZIP 10001 (New York, NY) might look like:

```
{
  "zipCode": "10001",
  "city": "NEW YORK",
  "state": "NY",
  "county": "NEW YORK",
  "countyFIPS": "061",
  "stateFIPS": "36",
  "latitude": 40.7505,
  "longitude": -73.9960,
  "timeZone": "EST",
  "areaCodes": "212/917/646",
  "region": "Northeast",
  "division": "Mid-Atlantic",
  "daylightSavings": "Y",
  "MSA": "35620",
  "MSAName": "New York-Newark-Jersey City, NY-NJ-PA",
  "landArea": 0.62,
  "waterArea": 0.0
  // ... possibly more fields
}
```

In this example, 10001 is in New York County (FIPS 061) in New York State (FIPS 36), in the Northeast region (Middle Atlantic division). It has multiple area codes (212, 917, 646) and observes DST. MSA 35620 corresponds to the New York City metro area. Land area is 0.62 sq mi (no water area).

- **Usage, Duration, Timestamp:** as usual.

Coding Examples

C# Example:

```
string json = new
WebClient().DownloadString("https://geodata.cdxtech.com/api/geogeneral?key=YO
UR_KEY&zipcode=10001&format=json");
Console.WriteLine(json);
```

JavaScript Example:

```
fetch('https://geodata.cdxtech.com/api/geogeneral?key=YOUR_KEY&zipcode=30301&
format=json')
  .then(res => res.json())
  .then(data => {
    console.log(`${data.zipCode} - ${data.city}, ${data.state}`);
    console.log(`County: ${data.county}, Timezone: ${data.timeZone},
AreaCodes: ${data.areaCodes}`);
  });
```

VBA Example:

```
Dim key$, zip$
key = "YOUR_KEY"
zip = "30301"
Dim url$
url = "https://geodata.cdxtech.com/api/geogeneral?key=" & key & "&zipcode=" &
zip & "&format=csv"
Dim http As Object: Set http = CreateObject("MSXML2.XMLHTTP")
http.Open "GET", url, False
http.Send
If http.Status = 200 Then
    Debug.Print http.responseText  ' CSV with all general fields for the ZIP
End If
```

Sample Output

JSON Example: (General data for ZIP 90210, Beverly Hills, CA)


```
{
  "service": "GeoGeneral",
  "status": "Success",
  "tokenCharge": 1,
  "results": [
    {
      "zipCode": "90210",
      "city": "BEVERLY HILLS",
      "state": "CA",
      "county": "LOS ANGELES",
      "countyFIPS": "037",
      "stateFIPS": "06",
      "latitude": 34.088808,
      "longitude": -118.406125,
      "timeZone": "PST",
      "areaCodes": "310/424",
      "region": "West",
```

```
      "division": "Pacific",
      "daylightSavings": "Y",
      "MSA": "31080",
      "MSAName": "Los Angeles-Long Beach-Anaheim, CA",
      "landArea": 9.86,
      "waterArea": 0.0
    }
  ]
}
```

This indicates that 90210 (Beverly Hills) is in Los Angeles County (FIPS 037), California (FIPS 06). It lies in the Pacific division of the West region. It has area codes 310 and 424. Land area ~9.86 sq mi. (Population is not included here since that would be part of GeoDemographics.)

XML Example:

```
<Root>
  <Service>GeoGeneral</Service>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Results>
    <ZipCode>90210</ZipCode>
    <City>BEVERLY HILLS</City>
    <State>CA</State>
    <County>LOS ANGELES</County>
    <CountyFIPS>037</CountyFIPS>
    <StateFIPS>06</StateFIPS>
    <Latitude>34.088808</Latitude>
    <Longitude>-118.406125</Longitude>
    <TimeZone>PST</TimeZone>
    <AreaCodes>310/424</AreaCodes>
    <Region>West</Region>
    <Division>Pacific</Division>
    <DaylightSavings>Y</DaylightSavings>
    <MSA>31080</MSA>
    <MSAName>Los Angeles-Long Beach-Anaheim, CA</MSAName>
    <LandArea>9.86</LandArea>
    <WaterArea>0.0</WaterArea>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.018</Duration>
  <TimeStamp>2025-04-28T10:47:15-04:00</TimeStamp>
</Root>
```

GeoDemographics

Overview

GeoDemographics provides a comprehensive set of demographic and economic statistics for a given U.S. ZIP Code. It aggregates data from sources like the 2010 Census and American Community Survey to give details on population, households, income, housing, and more. This function is essentially a one-stop for ZIP-level demographics, including many variables beyond basic counts.

Settings and Requirements

- **API Key:** Required.
- **Token Usage:** 5 token per request (despite the breadth of data).
- **Formats:** Supports `json`, `xml`, or `csv` (default `json`). Given the number of fields, CSV or JSON might be easier for parsing.

Request Details

- **Base URL:** `https://geodata.cdxtech.com/api/geodemographics:contentReference[oaicite:165]{index=165}】
- **HTTP Method:** GET
- **Parameters:**
  - **`key`** – Your API key.
  - **`zipcode`** – 5-digit ZIP code.
  - **`format`** – `json`, `xml`, or `csv`. Default `json`.

Example Request:

```
https://geodata.cdxtech.com/api/geodemographics?key=YOUR_KEY&zipcode=10001&format=json
```

Response Details

GeoDemographics returns an extensive set of fields. Key categories of data include:

- **Population Stats:** total population, population by age groups (median age, etc.), by race (similar to GeoRace), by gender (similar to GeoGender). Also population density (people per square mile/km).
- **Household Stats:** number of households, average household size, median household income, median family income, per capita income, poverty rates.
- **Housing Stats:** number of housing units, occupancy rates (occupied vs vacant), owner vs renter percentages, median home value, median rent.
- **Economic/Work Stats:** perhaps unemployment rate, labor force participation, etc. (if available at ZIP level).
- **Education Levels:** possibly educational attainment percentages (high school, bachelor's, etc.).
- **Miscellaneous:** Could include things like the predominant industry or travel time to work (in some datasets).

Because the output is very detailed, the documentation suggests referring to a data definitions resourc】 for all fields. Typically, every piece of data that might be relevant at a ZIP level is included.

For example, a partial JSON for ZIP 10001 might include:

```
{
  "zipCode": "10001",
  "city": "NEW YORK",
  "state": "NY",
  "population": 21102,
  "medianAge": 35.6,
  "populationDensityPerSqMi": 34035.5,
  "households": 10455,
  "averageHouseholdSize": 2.01,
  "medianHouseholdIncome": 73654,
  "perCapitaIncome": 53874,
  "housingUnits": 11228,
  "ownerOccupiedPercentage": 10.5,
  "renterOccupiedPercentage": 89.5,
  "vacantHousingPercentage": 5.3,
  "medianHomeValue": 850000,
  "medianGrossRent": 1650,
  "whitePercentage": 54.3,
  "blackPercentage": 15.8,
  "asianPercentage": 20.6,
  "hispanicPercentage": 17.1,
  "highSchoolOrHigherPercent": 92.4,
  "bachelorsOrHigherPercent": 68.9
  // ... etc.
}
```

(The above numbers are hypothetical for illustration.)

In XML, each of those would be an element under `<Results>`. In CSV, each would be a column in a single row for the ZIP.

GeoDemographics essentially combines what GeoRace and GeoGender provide (and possibly GeoGeneral's region/area fields) plus many more socio-economic indicators in one response.

- **Usage, Duration, Timestamp:** included as usual.

Coding Examples

C# Example:

```
string key = "YOUR_KEY";
string zip = "90210";
using(HttpClient client = new HttpClient())
{
    string json =
client.GetStringAsync($"https://geodata.cdxtech.com/api/geodemographics?key={
key}&zipcode={zip}&format=json").Result;
    // The JSON is extensive; parse as needed, for example:
    dynamic data = Newtonsoft.Json.JsonConvert.DeserializeObject(json);
    Console.WriteLine($"Population: {data.results[0].population}, Median
Income: {data.results[0].medianHouseholdIncome}");
}
```

JavaScript Example:

```
fetch('https://geodata.cdxtech.com/api/geodemographics?key=YOUR_KEY&zipcode=9
0210&format=json')
  .then(res => res.json())
  .then(data => {
    let demo = data.results[0];
    console.log(`ZIP ${demo.zipCode}: Pop=${demo.population},
MedAge=${demo.medianAge}, MedIncome=$${demo.medianHouseholdIncome}`);
    console.log(`Education: ${demo.bachelorsOrHigherPercent}% with bachelor's
or higher`);
  });
```

VBA Example:

```
Dim key$, zip$
key = "YOUR_KEY"
zip = "90210"
Dim url$
url = "https://geodata.cdxtech.com/api/geodemographics?key=" & key &
"&zipcode=" & zip & "&format=csv"
Dim http As Object: Set http = CreateObject("MSXML2.XMLHTTP")
http.Open "GET", url, False
http.Send
If http.Status = 200 Then
    Dim line As String
    line = http.responseText  ' Single-line CSV with many columns
    ' For brevity, split and show first 5 fields:
    Dim fields: fields = Split(line, ",")
    Dim i As Integer
    For i = 0 To 4
```

```
        Debug.Print fields(i);
    Next i
End If
```

Sample Output

JSON Example: (Partial demographics for ZIP 07869, Randolph, NJ)

```
{
  "service": "GeoDemographics",
  "status": "Success",
  "tokenCharge": 5,
  "results": [
    {
      "zipCode": "07869",
      "city": "RANDOLPH",
      "state": "NJ",
      "population": 25838,
      "medianAge": 41.3,
      "populationDensityPerSqMi": 1290.3,
      "households": 8920,
      "averageHouseholdSize": 2.89,
      "medianHouseholdIncome": 132333,
      "perCapitaIncome": 48765,
      "housingUnits": 9210,
      "ownerOccupiedPercentage": 86.5,
      "renterOccupiedPercentage": 13.5,
      "vacantHousingPercentage": 3.1,
      "medianHomeValue": 550000,
      "medianGrossRent": 1750,
      "whitePercentage": 82.23,
      "blackPercentage": 3.43,
      "asianPercentage": 11.14,
      "hispanicPercentage": 7.24,
      "multiRacePercentage": 0.00,
      "highSchoolOrHigherPercent": 95.7,
      "bachelorsOrHigherPercent": 60.4
      // ... more fields ...
    }
  ]
}
```

(Note: The above values are illustrative, not actual data for 07869 except population which we know from earlier sections. Median income, etc., are plausible estimates.)

This shows how GeoDemographics consolidates many data points: population, densities, income, housing occupancy, and even repeats some race percentages. Typically, one would select the specific fields of interest from this output.

XML Example: (partial for the same ZIP)

```
<Root>
  <Service>GeoDemographics</Service>
```

```
  <Status>Success</Status>
  <TokenCharge>5</TokenCharge>
  <Results>
    <ZipCode>07869</ZipCode>
    <City>RANDOLPH</City>
    <State>NJ</State>
    <Population>25838</Population>
    <MedianAge>41.3</MedianAge>
    <PopulationDensityPerSqMi>1290.3</PopulationDensityPerSqMi>
    <Households>8920</Households>
    <AverageHouseholdSize>2.89</AverageHouseholdSize>
    <MedianHouseholdIncome>132333</MedianHouseholdIncome>
    <PerCapitaIncome>48765</PerCapitaIncome>
    <HousingUnits>9210</HousingUnits>
    <OwnerOccupiedPercentage>86.5</OwnerOccupiedPercentage>
    <RenterOccupiedPercentage>13.5</RenterOccupiedPercentage>
    <VacantHousingPercentage>3.1</VacantHousingPercentage>
    <MedianHomeValue>550000</MedianHomeValue>
    <MedianGrossRent>1750</MedianGrossRent>
    <WhitePercentage>82.23</WhitePercentage>
    <BlackPercentage>3.43</BlackPercentage>
    <AsianPercentage>11.14</AsianPercentage>
    <HispanicPercentage>7.24</HispanicPercentage>
    <MultiRacePercentage>0.00</MultiRacePercentage>
    <HighSchoolOrHigherPercent>95.7</HighSchoolOrHigherPercent>
    <BachelorsOrHigherPercent>60.4</BachelorsOrHigherPercent>
    <!-- ... other fields ... -->
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.030</Duration>
  <TimeStamp>2025-04-28T10:47:45-04:00</TimeStamp>
</Root>
```

As shown, the GeoDemographics output is very detailed. Typically, users will focus on a subset of these fields relevant to their analysis (for example, just population and income, or education levels, etc.). The CDXGeoData documentation provides definitions for each field. When using the API, you receive all fields in the response (even if some values are null or 0 for certain ZIPs).

GeocodeHere

Overview

GeocodeHere performs forward geocoding of addresses using the HERE mapping service. It takes address components (street, city, state, ZIP, country) as input and returns the corresponding latitude and longitude, along with a standardized address match from the HERE API.

Settings and Requirements

- **key** (Required): Your API authentication key.
- **address** (Required): Street address (street and number) part of the location (e.g., `"123 Main St"`).
- **city** (Required): City name for the address.
- **state** (Required): State or province for the address (for US addresses, use state abbreviation, e.g., `"NJ"`).
- **zipcode** (Optional): ZIP or postal code. While optional, providing it can improve geocoding accuracy. (No default; if unknown, it can be left blank.)
- **country** (Optional): Country code (e.g., `"US"` for United States). Default is `"us"` if not provided.
- **format** (Optional): Response format, `json` or `xml`. Default is `json`.

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/geocode/here`
Example Request:

```
perl
CopyEdit
GET
https://geodata.cdxtech.com/api/geocode/here?key=YOUR_KEY&address=2+West+Hano
ver+Ave&city=Randolph&state=NJ&zipcode=07869&country=us&format=json
```

Response Details

The **GeocodeHere** response contains the geocoding result from HERE. In JSON format, it will include:

- `service`: `"GeocodeHere"`.
- `status`: `"Success"` (if a match is found).
- `totalResults`: number of matches (usually 1 for a specific address query).
- `results`: an object with details of the geocoded address. Typically, this includes:
    - `bestMatch` or similar: the full formatted address returned.
    - `latitude` and `longitude`: coordinates of the matched address.
    - `street`, `city`, `state`, `zip`, `county`, `country`: Components of the address as identified by HERE.
    - Possibly a `confidence` or score indicating match accuracy (if provided by the API).

XML format will have analogous tags for these fields.

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string address = "2 west hanover ave";
string city = "randolph";
string state = "nj";
string zipcode = "07869";
string country = "us";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    StringBuilder url = new StringBuilder("/api/geocode/here?");
    url.Append("key=").Append(key);
    url.Append("&address=").Append(Uri.EscapeDataString(address));
    url.Append("&city=").Append(Uri.EscapeDataString(city));
    url.Append("&state=").Append(state);
    url.Append("&zipcode=").Append(zipcode);
    url.Append("&country=").Append(country);
    url.Append("&format=").Append(format);

    HttpResponseMessage message = client.GetAsync(url.ToString()).Result;
    // Process the response message...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/geocode/here/', true);
xhr.onload = function() {
    console.log(this.responseText);
};
```

```
xhr.send('key=YOUR_KEY&address=2%20west%20hanover%20ave&city=randolph&state=n
j&zipcode=07869&country=us&format=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim address As String: address = "2 west hanover ave"
Dim city As String: city = "randolph"
Dim state As String: state = "nj"
Dim zipcode As String: zipcode = "07869"
Dim country As String: country = "us"
Dim format As String: format = "json"

Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/geocode/here?" & _
             "key=" & key & "&address=" & Replace(address, " ", "%20") & _
             "&city=" & city & "&state=" & state & "&zipcode=" & zipcode & _
             "&country=" & country & "&format=" & format

Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim geoResult As String
    geoResult = xhr.responseText  ' JSON or XML string
    ' Handle the geocoding result...
End If
```

Sample Outputs

JSON Response Example:

```
json
CopyEdit
{
    "service": "GeocodeHere",
    "url":
"https://geodata.cdxtech.com/api/geocode/here?address=2%20West%20Hanover%20Av
e&city=Randolph&state=NJ&zipcode=07869&country=us&key=YOUR_KEY",
    "status": "Success",
    "tokenCharge": 1,
    "message": null,
    "totalResults": 1,
    "results": {
        "bestMatch": "2 W Hanover Ave, Randolph, NJ 07869-4212, United
States",
        "latitude": "40.82653",
        "longitude": "-74.56786",
        "street": "2 W Hanover Ave",
        "city": "Randolph",
        "state": "New Jersey",
        "zip": "07869-4212",
```

```
        "county": "Morris",
        "country": "United States"
    },
    "usage": {
        "used": 2597,
        "usedPercentage": 1.315582843,
        "remaining": 197403,
        "remainingPercentage": 98.684417157
    },
    "duration": 0.9913322,
    "timeStamp": "2017-07-24T09:55:29.5369088-04:00"
}
```


XML Response Example:


```
xml
CopyEdit
<Root>
  <Service>GeocodeHere</Service>

<Url>https://geodata.cdxtech.com/api/geocode/here?address=2%20West%20Hanover%
20Ave&amp;amp;city=Randolph&amp;amp;state=NJ&amp;amp;zipcode=07869&amp;amp;country=us&amp;amp;key
=YOUR_KEY&amp;amp;format=xml</Url>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Message />
  <TotalResults>1</TotalResults>
  <Results>
    <BestMatch>2 W Hanover Ave, Randolph, NJ 07869-4212, United
States</BestMatch>
    <Latitude>40.82653</Latitude>
    <Longitude>-74.56786</Longitude>
    <Street>2 W Hanover Ave</Street>
    <City>Randolph</City>
    <State>New Jersey</State>
    <Zip>07869-4212</Zip>
    <County>Morris</County>
    <Country>United States</Country>
  </Results>
  <Usage>
    <Used>2597</Used>
    <UsedPercentage>1.315582843</UsedPercentage>
    <Remaining>197403</Remaining>
    <RemainingPercentage>98.684417157</RemainingPercentage>
  </Usage>
  <Duration>0.9913322</Duration>
  <TimeStamp>2017-07-24T09:55:29.5369088-04:00</TimeStamp>
</Root>
```

GeocodeHereSingle

Overview

GeocodeHereSingle is a simplified version of the HERE forward geocoding service where the entire address is provided as one single query string. This is useful for geocoding when you have a full address in one line, rather than separate components.

Settings and Requirements

- **key** (Required): Your API authentication key.
- **query** (Required): The full address or place query (point of interest, landmark, or single-line address).
- **format** (Optional): `json` or `xml`. Default is `json`.

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/geocode/here/single`
Example Request:

```
pgsql
CopyEdit
GET
https://geodata.cdxtech.com/api/geocode/here/single?key=YOUR_KEY&query=2+West
+Hanover+Ave,+Randolph,+NJ&format=json
```

Response Details

The response for **GeocodeHereSingle** is similar to **GeocodeHere**, except that the input address is not split into parts. The service will return the best match for the provided single-line query. The output contains the matched address details and coordinates, just like the standard GeocodeHere:

- `bestMatch`: The full matched address.
- `latitude`, `longitude`: Coordinates of the address.
- Address components (`street`, `city`, `state`, etc.) as determined.

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string addressQuery = "2 west hanover ave, randolph, nj";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geocode/here/single?key={key}&query={Uri.EscapeDataString(addressQuery
)}&format={format}";
    HttpResponseMessage message = client.GetAsync(url).Result;
    // Handle response...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/geocode/here/single/',
true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_KEY&query=2%20west%20hanover%20ave,%20randolph,%20nj&forma
t=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim addressQuery As String: addressQuery = "2 west hanover ave, randolph, nj"
Dim format As String: format = "json"

Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/geocode/here/single?" & _
             "key=" & key & "&query=" & Replace(addressQuery, " ", "%20") &
"&format=" & format

Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim resultJson As String
    resultJson = xhr.responseText
    ' Process JSON result...
End If
```

Sample Outputs

JSON Response Example:

```
json
CopyEdit
{
    "service": "GeocodeHereSingle",
    "url":
"https://geodata.cdxtech.com/api/geocode/here/single?key=YOUR_KEY&query=2%20W
est%20Hanover%20Ave,%20Randolph,%20NJ&format=json",
    "status": "Success",
    "tokenCharge": 1,
    "message": null,
    "totalResults": 1,
    "results": {
        "bestMatch": "2 W Hanover Ave, Randolph, NJ 07869-4212, United
States",
        "latitude": "40.82653",
        "longitude": "-74.56786",
        "street": "2 W Hanover Ave",
        "city": "Randolph",
        "state": "New Jersey",
        "zip": "07869-4212",
        "county": "Morris",
        "country": "United States"
    },
    "usage": {
        "used": 100,
        "remaining": 1000
    },
    "duration": 0.2501234,
    "timeStamp": "2017-07-24T10:00:00.1234567-04:00"
}
```

XML Response Example:

```
xml
CopyEdit
<Root>
  <Service>GeocodeHereSingle</Service>

<Url>https://geodata.cdxtech.com/api/geocode/here/single?key=YOUR_KEY&amp;que
ry=2%20West%20Hanover%20Ave,%20Randolph,%20NJ&amp;format=xml</Url>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Message />
  <TotalResults>1</TotalResults>
  <Results>
    <BestMatch>2 W Hanover Ave, Randolph, NJ 07869-4212, United
States</BestMatch>
    <Latitude>40.82653</Latitude>
    <Longitude>-74.56786</Longitude>
    <Street>2 W Hanover Ave</Street>
    <City>Randolph</City>
```

```
    <State>New Jersey</State>
    <Zip>07869-4212</Zip>
    <County>Morris</County>
    <Country>United States</Country>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.2501234</Duration>
  <TimeStamp>2017-07-24T10:00:00.1234567-04:00</TimeStamp>
</Root>
```

GeocodeHereReverse

Overview

GeocodeHereReverse performs reverse geocoding using the HERE service. Given a latitude and longitude, it returns the nearest or most appropriate address (typically the address at that coordinate or closest to it).

Settings and Requirements

- **key** (Required): Your API key.
- **latitude** (Required): The latitude coordinate for the reverse geocoding query.
- **longitude** (Required): The longitude coordinate for the reverse geocoding query.
- **format** (Optional): `json` or `xml`. Default is `json`.

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/geocode/here/reverse`
Example Request:

```
pgsql
CopyEdit
GET
https://geodata.cdxtech.com/api/geocode/here/reverse?key=YOUR_KEY&latitude=40
.82653&longitude=-74.56786&format=json
```

Response Details

A successful **GeocodeHereReverse** response will include the address details corresponding to the provided coordinates. The JSON output structure typically includes:

- `bestMatch`: The address found at or near the given coordinates.
- `latitude`, `longitude`: Echo of the input coordinates (or coordinates of the matched address, which should be very close).
- `street`, `city`, `state`, `zip`, `county`, `country`: The components of the address identified.

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string lat = "40.82653";
string lon = "-74.56786";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geocode/here/reverse?key={key}&latitude={lat}&longitude={lon}&format={
format}";
    HttpResponseMessage message = client.GetAsync(url).Result;
    // Process the response...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/geocode/here/reverse/',
true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_KEY&latitude=40.82653&longitude=-74.56786&format=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim lat As String: lat = "40.82653"
Dim lon As String: lon = "-74.56786"
Dim format As String: format = "json"

Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/geocode/here/reverse?" & _
```

```
             "key=" & key & "&latitude=" & lat & "&longitude=" & lon &
"&format=" & format

Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim resp As String
    resp = xhr.responseText
    ' Process reverse geocoding response...
End If
```

Sample Outputs

JSON Response Example:

```
json
CopyEdit
{
    "service": "GeocodeHereReverse",
    "url":
"https://geodata.cdxtech.com/api/geocode/here/reverse?latitude=40.82653&longi
tude=-74.56786&key=YOUR_KEY",
    "status": "Success",
    "tokenCharge": 1,
    "message": null,
    "totalResults": 1,
    "results": {
        "bestMatch": "2 W Hanover Ave, Randolph, NJ 07869-4212, United
States",
        "latitude": "40.82653",
        "longitude": "-74.56786",
        "street": "2 W Hanover Ave",
        "city": "Randolph",
        "state": "New Jersey",
        "zip": "07869-4212",
        "county": "Morris",
        "country": "United States"
    },
    "usage": {
        "used": 100,
        "remaining": 1000
    },
    "duration": 0.1876543,
    "timeStamp": "2017-07-24T10:05:00.6543210-04:00"
}
```

XML Response Example:

```
xml
CopyEdit
<Root>
  <Service>GeocodeHereReverse</Service>
```

```
<Url>https://geodata.cdxtech.com/api/geocode/here/reverse?latitude=40.82653&a
mp;longitude=-74.56786&amp;key=YOUR_KEY&amp;format=xml</Url>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Message />
  <TotalResults>1</TotalResults>
  <Results>
    <BestMatch>2 W Hanover Ave, Randolph, NJ 07869-4212, United
States</BestMatch>
    <Latitude>40.82653</Latitude>
    <Longitude>-74.56786</Longitude>
    <Street>2 W Hanover Ave</Street>
    <City>Randolph</City>
    <State>New Jersey</State>
    <Zip>07869-4212</Zip>
    <County>Morris</County>
    <Country>United States</Country>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.1876543</Duration>
  <TimeStamp>2017-07-24T10:05:00.6543210-04:00</TimeStamp>
</Root>
```


GeoRouteHere

Overview

GeoRouteHere computes a route between an origin and a destination using the HERE routing service. It can return the travel distance and time for driving, walking, or other modes of transport, with options for routing mode (fastest or shortest route).

Settings and Requirements

- **key** (Required): Your API key.
- **origin** (Required): The starting location for the route. This can be specified as coordinates (`latitude,longitude`), or as a recognizable address or place name.
- **destination** (Required): The end location for the route (in the same format as origin).
- **transportmode** (Optional): Mode of transportation. Options include `car`, `truck`, or `pedestrian`. Default is `car`.
- **routingmode** (Optional): Route optimization mode. Options are `fast` (fastest route) or `short` (shortest route). Default is `fast`.
- **geocode** (Optional): If set to `true`, the API will geocode the provided origin and destination strings internally. If both origin and destination are provided as coordinates already, this can be left as `false`. Default is `false`.

- **format** (Optional): `json` or `xml`. Default is `json`.

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/georoute/here`
Example Request:

```
pgsql
CopyEdit
GET
https://geodata.cdxtech.com/api/georoute/here?key=YOUR_KEY&origin=40.8481,-
74.5726&destination=40.9253,-
74.1785&transportmode=car&routingmode=fast&geocode=false&format=json
```

Response Details

The **GeoRouteHere** response provides the distance and travel time for the route. Key elements in a JSON response include:

- `travelDistance`: The total distance of the route, usually in kilometers or miles (units may be implicit or provided).
- `travelDuration`: The estimated travel time in seconds.
- These values are typically contained in a `results` array (or object) under `results`. For example, `results[0].travelDistance` and `results[0].travelDuration`.

The response does not necessarily include turn-by-turn directions in this format, but focuses on summary route information (distance and duration). The XML format similarly will include `<TravelDistance>` and `<TravelDuration>` elements.

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string origin = "40.8481,-74.5726";
string destination = "40.9253,-74.1785";
string transportMode = "car";
string routingMode = "fast";
string geocodeFlag = "false";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    StringBuilder url = new StringBuilder("/api/georoute/here?");
    url.Append("key=").Append(key);
    url.Append("&origin=").Append(origin);
    url.Append("&destination=").Append(destination);
```

```
    url.Append("&transportmode=").Append(transportMode);
    url.Append("&routingmode=").Append(routingMode);
    url.Append("&geocode=").Append(geocodeFlag);
    url.Append("&format=").Append(format);

    HttpResponseMessage message = client.GetAsync(url.ToString()).Result;
    // Process route response...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/georoute/here/', true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_KEY&origin=40.8481,-74.5726&destination=40.9253,-
74.1785&transportmode=car&routingmode=fast&geocode=false&format=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim origin As String: origin = "40.8481,-74.5726"
Dim destination As String: destination = "40.9253,-74.1785"
Dim transportMode As String: transportMode = "car"
Dim routingMode As String: routingMode = "fast"
Dim geocodeFlag As String: geocodeFlag = "false"
Dim format As String: format = "json"

Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/georoute/here?" & _
             "key=" & key & "&origin=" & origin & "&destination=" &
destination & _
             "&transportmode=" & transportMode & "&routingmode=" &
routingMode & _
             "&geocode=" & geocodeFlag & "&format=" & format

Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim routeResponse As String
    routeResponse = xhr.responseText  ' contains distance and duration info
    ' Process the routeResponse...
End If
```

Sample Outputs

JSON Response Example:

```
json
CopyEdit
{
    "service": "GeoRouteHere",
    "url":
"https://geodata.cdxtech.com/api/georoute/here?key=YOUR_KEY&origin=40.8481,-
74.5726&destination=40.9253,-
74.1785&transportmode=car&routingmode=fast&geocode=false&format=json",
    "status": "Success",
    "tokenCharge": 1,
    "message": null,
    "totalResults": 1,
    "results": [{
        "travelDistance": 12.345,
        "travelDuration": 1560.0
    }],
    "usage": {
        "used": 50,
        "remaining": 9950
    },
    "duration": 0.1234567,
    "timeStamp": "2025-01-01T12:00:00.0000000-05:00"
}
```

XML Response Example:

```
xml
CopyEdit
<Root>
  <Service>GeoRouteHere</Service>

<Url>https://geodata.cdxtech.com/api/georoute/here?key=YOUR_KEY&amp;origin=40
.8481,-74.5726&amp;destination=40.9253,-
74.1785&amp;transportmode=car&amp;routingmode=fast&amp;geocode=false&amp;form
at=xml</Url>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Message />
  <TotalResults>1</TotalResults>
  <Results>
    <TravelDistance>12.345</TravelDistance>
    <TravelDuration>1560</TravelDuration>
  </Results>
  <Usage>
    <Used>50</Used>
    <Remaining>9950</Remaining>
  </Usage>
  <Duration>0.1234567</Duration>
  <TimeStamp>2025-01-01T12:00:00.0000000-05:00</TimeStamp>
</Root>
```

GeocodeGoogle

Overview

GeocodeGoogle is a forward geocoding service that uses the Google Maps Geocoding API via CDXGeoData. It converts a given street address into geographic coordinates and returns the standardized address components from Google's geocoding results.

Settings and Requirements

- **key** (Required): Your API key.
- **address** (Required): Street address line (e.g., `"1600 Amphitheatre Pkwy"`).
- **city** (Required): City name.
- **state** (Required): State/region.
- **zipcode** (Optional): ZIP or postal code (if applicable).
- **country** (Optional): Country code or name (default is `"us"` if not specified).
- **format** (Optional): `json` or `xml`. Default is `json`.

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/geocode/google`
Example Request:

```
pgsql
CopyEdit
GET
https://geodata.cdxtech.com/api/geocode/google?key=YOUR_KEY&address=1600+Amph
itheatre+Pkwy&city=Mountain+View&state=CA&zipcode=94043&country=us&format=jso
n
```

Response Details

The **GeocodeGoogle** response includes the geocoded address information from Google. In JSON format, it will typically contain:

- `service`: `"GeocodeGoogle"`.
- `results`: an object with the geocoding outcome, including:
  - `formattedAddress` or `bestMatch`: the full standardized address.
  - `latitude` and `longitude`: coordinates of the address.
  - Individual address components: `street`, `city`, `state` (or region name), `zip` (postal code), `county`, and `country`.
  - A `confidence` or accuracy score (Google often provides a location_type or similar indicator of accuracy).

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string address = "1600 Amphitheatre Pkwy";
string city = "Mountain View";
string state = "CA";
string zipcode = "94043";
string country = "us";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geocode/google?key={key}&address={Uri.EscapeDataString(address)}&city=
{Uri.EscapeDataString(city)}&state={state}&zipcode={zipcode}&country={country
}&format={format}";
    HttpResponseMessage message = client.GetAsync(url).Result;
    // Handle the response...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/geocode/google/', true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_KEY&address=1600%20Amphitheatre%20Pkwy&city=Mountain%20Vie
w&state=CA&zipcode=94043&country=us&format=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim address As String: address = "1600 Amphitheatre Pkwy"
Dim city As String: city = "Mountain View"
Dim state As String: state = "CA"
Dim zipcode As String: zipcode = "94043"
Dim country As String: country = "us"
Dim format As String: format = "json"

Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/geocode/google?" & _
             "key=" & key & "&address=" & Replace(address, " ", "%20") & _
             "&city=" & Replace(city, " ", "%20") & "&state=" & state & _
             "&zipcode=" & zipcode & "&country=" & country & "&format=" &
format

Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
```

```
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim geoJson As String
    geoJson = xhr.responseText
    ' Process the JSON geocode result...
End If
```

Sample Outputs

JSON Response Example:

```
json
CopyEdit
{
    "service": "GeocodeGoogle",
    "url":
"https://geodata.cdxtech.com/api/geocode/google?address=1600%20Amphitheatre%2
0Pkwy&city=Mountain%20View&state=CA&zipcode=94043&country=us&key=YOUR_KEY",
    "status": "Success",
    "tokenCharge": 1,
    "message": null,
    "totalResults": 1,
    "results": {
        "bestMatch": "1600 Amphitheatre Parkway, Mountain View, CA 94043,
USA",
        "latitude": "37.422388",
        "longitude": "-122.084188",
        "street": "1600 Amphitheatre Parkway",
        "city": "Mountain View",
        "state": "California",
        "zip": "94043",
        "county": "Santa Clara County",
        "country": "United States"
    },
    "usage": {
        "used": 100,
        "remaining": 1000
    },
    "duration": 0.2000000,
    "timeStamp": "2025-01-01T12:01:00.0000000-05:00"
}
```

XML Response Example:

```
xml
CopyEdit
<Root>
  <Service>GeocodeGoogle</Service>

<Url>https://geodata.cdxtech.com/api/geocode/google?address=1600%20Amphitheat
re%20Pkwy&amp;city=Mountain%20View&amp;state=CA&amp;zipcode=94043&amp;country
=us&amp;key=YOUR_KEY&amp;format=xml</Url>
  <Status>Success</Status>
```

```
  <TokenCharge>1</TokenCharge>
  <Message />
  <TotalResults>1</TotalResults>
  <Results>
    <BestMatch>1600 Amphitheatre Parkway, Mountain View, CA 94043,
USA</BestMatch>
    <Latitude>37.422388</Latitude>
    <Longitude>-122.084188</Longitude>
    <Street>1600 Amphitheatre Parkway</Street>
    <City>Mountain View</City>
    <State>California</State>
    <Zip>94043</Zip>
    <County>Santa Clara County</County>
    <Country>United States</Country>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.2000000</Duration>
  <TimeStamp>2025-01-01T12:01:00.0000000-05:00</TimeStamp>
</Root>
```

GeocodeGoogleSingle

Overview

GeocodeGoogleSingle forwards a single-line address query to Google's geocoding service. It is similar to GeocodeGoogle, but instead of providing the address in multiple parts, the entire address is given as one string.

Settings and Requirements

- **key** (Required): API key.
- **query** (Required): The full address or place query in one string.
- **format** (Optional): `json` or `xml`. Default is `json`.

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/geocode/google/single`
Example Request:

```
pgsql
CopyEdit
GET
https://geodata.cdxtech.com/api/geocode/google/single?key=YOUR_KEY&query=1600
+Amphitheatre+Pkwy,+Mountain+View,+CA&format=json
```

Response Details

The output of **GeocodeGoogleSingle** will mirror that of GeocodeGoogle, providing the matched address and coordinates. With a single-line query, Google will interpret the various components automatically. The JSON result will contain the full matched address and its components (street, city, state, etc.) and coordinates.

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string addressQuery = "1600 Amphitheatre Pkwy, Mountain View, CA";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geocode/google/single?key={key}&query={Uri.EscapeDataString(addressQue
ry)}&format={format}";
    HttpResponseMessage message = client.GetAsync(url).Result;
    // Process message...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/geocode/google/single/',
true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_KEY&query=1600%20Amphitheatre%20Pkwy,%20Mountain%20View,%2
0CA&format=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim addressQuery As String: addressQuery = "1600 Amphitheatre Pkwy, Mountain
View, CA"
Dim format As String: format = "json"

Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/geocode/google/single?" & _
             "key=" & key & "&query=" & Replace(addressQuery, " ", "%20") &
"&format=" & format
```

```
Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim googleGeoJson As String
    googleGeoJson = xhr.responseText
    ' Process Google geocode single-line result...
End If
```

Sample Outputs

JSON Response Example:

```
json
CopyEdit
{
    "service": "GeocodeGoogleSingle",
    "url":
"https://geodata.cdxtech.com/api/geocode/google/single?key=YOUR_KEY&query=160
0%20Amphitheatre%20Pkwy,%20Mountain%20View,%20CA&format=json",
    "status": "Success",
    "tokenCharge": 1,
    "message": null,
    "totalResults": 1,
    "results": {
        "bestMatch": "1600 Amphitheatre Parkway, Mountain View, CA 94043,
USA",
        "latitude": "37.422388",
        "longitude": "-122.084188",
        "street": "1600 Amphitheatre Parkway",
        "city": "Mountain View",
        "state": "California",
        "zip": "94043",
        "county": "Santa Clara County",
        "country": "United States"
    },
    "usage": {
        "used": 100,
        "remaining": 1000
    },
    "duration": 0.1500000,
    "timeStamp": "2025-01-01T12:02:00.0000000-05:00"
}
```

XML Response Example:

```
xml
CopyEdit
<Root>
  <Service>GeocodeGoogleSingle</Service>

<Url>https://geodata.cdxtech.com/api/geocode/google/single?key=YOUR_KEY&amp;q
uery=1600%20Amphitheatre%20Pkwy,%20Mountain%20View,%20CA&amp;format=xml</Url>
```

```
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Message />
  <TotalResults>1</TotalResults>
  <Results>
    <BestMatch>1600 Amphitheatre Parkway, Mountain View, CA 94043,
USA</BestMatch>
    <Latitude>37.422388</Latitude>
    <Longitude>-122.084188</Longitude>
    <Street>1600 Amphitheatre Parkway</Street>
    <City>Mountain View</City>
    <State>California</State>
    <Zip>94043</Zip>
    <County>Santa Clara County</County>
    <Country>United States</Country>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.1500000</Duration>
  <TimeStamp>2025-01-01T12:02:00.0000000-05:00</TimeStamp>
</Root>
```

GeocodeGoogleReverse

Overview

GeocodeGoogleReverse uses Google's geocoding to reverse-geocode coordinates into a human-readable address. Provide latitude and longitude, and it returns the nearest address or location name according to Google Maps.

Settings and Requirements

- **key** (Required): API key.
- **latitude** (Required): Latitude of the location.
- **longitude** (Required): Longitude of the location.
- **format** (Optional): `json` or `xml`. Default is `json`.

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/geocode/google/reverse`
Example Request:

```
pgsql
CopyEdit
GET
https://geodata.cdxtech.com/api/geocode/google/reverse?key=YOUR_KEY&latitude=
37.422388&longitude=-122.084188&format=json
```

Response Details

The **GeocodeGoogleReverse** service will return an address for the given coordinates. The JSON response will typically include:

- `bestMatch`: The address or place name that Google identifies at those coordinates.
- The address components (`street`, `city`, `state`, etc.) of that match.
- The original coordinates (latitude, longitude).
- If available, a `placeId` or other Google-specific identifiers might also be included.

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string latitude = "37.422388";
string longitude = "-122.084188";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/geocode/google/reverse?key={key}&latitude={latitude}&longitude={longit
ude}&format={format}";
    HttpResponseMessage message = client.GetAsync(url).Result;
    // Handle response...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/geocode/google/reverse/',
true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_KEY&latitude=37.422388&longitude=-
122.084188&format=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim latitude As String: latitude = "37.422388"
Dim longitude As String: longitude = "-122.084188"
Dim format As String: format = "json"
```

```
Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/geocode/google/reverse?" & _
             "key=" & key & "&latitude=" & latitude & "&longitude=" &
longitude & "&format=" & format

Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim revGeoJson As String
    revGeoJson = xhr.responseText
    ' Process reverse geocode result...
End If
```

Sample Outputs

JSON Response Example:

```
json
CopyEdit
{
    "service": "GeocodeGoogleReverse",
    "url":
"https://geodata.cdxtech.com/api/geocode/google/reverse?latitude=37.422388&lo
ngitude=-122.084188&key=YOUR_KEY",
    "status": "Success",
    "tokenCharge": 1,
    "message": null,
    "totalResults": 1,
    "results": {
        "bestMatch": "1600 Amphitheatre Parkway, Mountain View, CA 94043,
USA",
        "latitude": "37.422388",
        "longitude": "-122.084188",
        "street": "1600 Amphitheatre Parkway",
        "city": "Mountain View",
        "state": "California",
        "zip": "94043",
        "county": "Santa Clara County",
        "country": "United States"
    },
    "usage": {
        "used": 100,
        "remaining": 1000
    },
    "duration": 0.1800000,
    "timeStamp": "2025-01-01T12:03:00.0000000-05:00"
}
```

XML Response Example:

```
xml
CopyEdit
```

```
<Root>
  <Service>GeocodeGoogleReverse</Service>

<Url>https://geodata.cdxtech.com/api/geocode/google/reverse?latitude=37.42238
8&amp;longitude=-122.084188&amp;key=YOUR_KEY&amp;format=xml</Url>
  <Status>Success</Status>
  <TokenCharge>1</TokenCharge>
  <Message />
  <TotalResults>1</TotalResults>
  <Results>
    <BestMatch>1600 Amphitheatre Parkway, Mountain View, CA 94043,
USA</BestMatch>
    <Latitude>37.422388</Latitude>
    <Longitude>-122.084188</Longitude>
    <Street>1600 Amphitheatre Parkway</Street>
    <City>Mountain View</City>
    <State>California</State>
    <Zip>94043</Zip>
    <County>Santa Clara County</County>
    <Country>United States</Country>
  </Results>
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.1800000</Duration>
  <TimeStamp>2025-01-01T12:03:00.0000000-05:00</TimeStamp>
</Root>
```


GeoRouteGoogle

Overview

GeoRouteGoogle calculates a route between two locations using the Google Maps Directions service. It returns the total travel distance and duration for a specified travel mode (such as driving or walking).

Settings and Requirements

- **key** (Required): Your API key.
- **origin** (Required): Starting location (can be given as coordinates or an address string).
- **destination** (Required): Destination location (coordinates or address).
- **travelMode** (Optional): Mode of travel. Options include `driving`, `bicycling`, `walking`, `transit` (these correspond to Google's modes). Default is `driving`.
- **format** (Optional): `json` or `xml`. Default is `json`.

(Note: Google's routing may not use a separate "fast/short" routing mode parameter as HERE does; it usually provides the fastest route by default.)

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/georoute/google`
Example Request:

```
pgsql
CopyEdit
GET
https://geodata.cdxtech.com/api/georoute/google?key=YOUR_KEY&origin=34.102676
0,-118.4524720&destination=33.78750,-117.93320&travelMode=driving&format=json
```

Response Details

The **GeoRouteGoogle** response includes summary information about the route:

- `travelDistance`: The length of the route (in the units used by Google's API, typically meters which can be converted to miles or kilometers).
- `travelDuration`: The travel time in seconds. These are given in the `results` array/object in JSON. Additional metadata like `service`, `status`, etc., are included as usual. The service focuses on providing the distance and duration, not turn-by-turn directions.

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string origin = "34.1026760,-118.4524720";
string destination = "33.78750,-117.93320";
string travelMode = "driving";
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/georoute/google?key={key}&origin={origin}&destination={destination}&tr
avelMode={travelMode}&format={format}";
    HttpResponseMessage message = client.GetAsync(url).Result;
    // Process the message...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/georoute/google/', true);
xhr.onload = function() {
```

```
    console.log(this.responseText);
};
xhr.send('key=YOUR_KEY&origin=34.1026760,-118.4524720&destination=33.78750,-
117.93320&travelMode=driving&format=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim origin As String: origin = "34.1026760,-118.4524720"
Dim destination As String: destination = "33.78750,-117.93320"
Dim travelMode As String: travelMode = "driving"
Dim format As String: format = "json"

Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/georoute/google?" & _
             "key=" & key & "&origin=" & origin & "&destination=" &
destination & _
             "&travelMode=" & travelMode & "&format=" & format

Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim routeJson As String
    routeJson = xhr.responseText
    ' Process route distance/duration from routeJson...
End If
```

Sample Outputs

JSON Response Example:

```
json
CopyEdit
{
    "service": "GeoRouteGoogle",
    "url":
"https://geodata.cdxtech.com/api/georoute/google?key=YOUR_KEY&origin=34.10267
60,-118.4524720&destination=33.78750,-
117.93320&travelMode=driving&format=json",
    "status": "Success",
    "tokenCharge": 2,
    "message": null,
    "totalResults": 1,
    "results": [{
        "travelDistance": 48.370019,
        "travelDuration": 3249.0
    }],
    "usage": {
        "used": 2599,
        "usedPercentage": 1.316609338,
        "remaining": 197401,
```

```
        "remainingPercentage": 98.683390662
    },
    "duration": 0.1123689,
    "timeStamp": "2017-07-24T09:55:31.9213573-04:00"
}
```

XML Response Example:

```
xml
CopyEdit
<Root>
  <Service>GeoRouteGoogle</Service>

<Url>https://geodata.cdxtech.com/api/georoute/google?key=YOUR_KEY&amp;origin=
34.1026760,-118.4524720&amp;destination=33.78750,-
117.93320&amp;travelMode=driving&amp;format=xml</Url>
  <Status>Success</Status>
  <TokenCharge>2</TokenCharge>
  <Message />
  <TotalResults>1</TotalResults>
  <Results>
    <TravelDistance>48.370019</TravelDistance>
    <TravelDuration>3249</TravelDuration>
  </Results>
  <Usage>
    <Used>2599</Used>
    <UsedPercentage>1.316609338</UsedPercentage>
    <Remaining>197401</Remaining>
    <RemainingPercentage>98.683390662</RemainingPercentage>
  </Usage>
  <Duration>0.1123689</Duration>
  <TimeStamp>2017-07-24T09:55:31.9213573-04:00</TimeStamp>
</Root>
```

Token Usage

Overview

Token Usage is a utility service that returns information about your API usage (tokens consumed) and remaining quota. It can optionally provide a detailed breakdown of recent requests and their token charges.

Settings and Requirements

- **key** (Required): Your API authentication key.
- **details** (Optional): Flag to indicate whether detailed usage information should be returned. Acceptable values: `"true"`, `"1"`, `"yes"`, or `"on"` (these are treated equivalently). Default is `0`/`false` (meaning **do not** return detailed request breakdown).
- **format** (Optional): `json` or `xml`. Default is `json`.

Request Details

API Endpoint: `https://geodata.cdxtech.com/api/utility/tokenusage`
Example Request:

```
pgsql
CopyEdit
GET
https://geodata.cdxtech.com/api/utility/tokenusage?key=YOUR_KEY&details=1&for
mat=json
```

Response Details

A **Token Usage** response provides your current usage statistics. Without details, it will simply show the total tokens used and remaining. With `details=1`, the response includes a breakdown of recent service calls that consumed tokens. In JSON format:

- `service`: `"GeoDataUtility"` (indicating a utility service).
- If details are enabled, `results` will be an array of objects, each representing a request that used tokens, with fields:
  - `service`: the name of the API service called (e.g., "GeoGeneral", "GeoRadius", etc.).
  - `requestUrl`: the endpoint and parameters of that request (with sensitive info like the key omitted or partially shown).
  - `ipAddress`: the IP address from which the request was made.
  - `tokenCharge`: how many tokens that request consumed.
- Summary fields `used` and `remaining` appear under a `usage` object (or directly in the root for no-detail response), indicating the total tokens used and remaining on your account.
- `totalResults` corresponds to the number of entries in the results array (when details are on, this could be the count of recent requests listed, otherwise 1).

The XML output contains analogous information in `<Results>` elements and a `<Usage>` section.

Coding Examples

C#:

```
csharp
CopyEdit
string key = "YOUR_KEY";
string detailsFlag = "1";  // "1" to get detailed usage, "0" or "false" for
summary only
string format = "json";

using (HttpClient client = new HttpClient())
{
    client.BaseAddress = new Uri("https://geodata.cdxtech.com");
    string url =
$"/api/utility/tokenusage?key={key}&details={detailsFlag}&format={format}";
```

```
    HttpResponseMessage message = client.GetAsync(url).Result;
    // Process usage info...
}
```

JavaScript:

```
javascript
CopyEdit
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://geodata.cdxtech.com/api/utility/tokenusage/', true);
xhr.onload = function() {
    console.log(this.responseText);
};
xhr.send('key=YOUR_KEY&details=1&format=json');
```

VBA:

```
vba
CopyEdit
Dim key As String: key = "YOUR_KEY"
Dim detailsFlag As String: detailsFlag = "1"  ' or "0"
Dim format As String: format = "json"

Dim requestUrl As String
requestUrl = "https://geodata.cdxtech.com/api/utility/tokenusage?" & _
             "key=" & key & "&details=" & detailsFlag & "&format=" & format

Dim xhr As Object: Set xhr = CreateObject("MSXML2.XMLHTTP")
xhr.Open "GET", requestUrl, False
xhr.Send

If xhr.Status = 200 Then
    Dim usageResponse As String
    usageResponse = xhr.responseText
    ' Handle token usage info...
End If
```

Sample Outputs

JSON Response Example (Detailed):

```
json
CopyEdit
{
    "service": "GeoDataUtility",
    "url":
"https://geodata.cdxtech.com/api/utility/tokenusage?key=YOUR_KEY&details=1",
    "status": "Success",
    "tokenCharge": 0,
    "message": null,
    "totalResults": 1,
    "results": [
        {
            "service": "GeoGeneral",
```

```
            "requestUrl":
"https://geodata.cdxtech.com/api/GeoGeneral?key=YOUR_KEY&zipcode=07869&format
=",
            "ipAddress": "127.0.0.1",
            "tokenCharge": 1
        },
        {
            "service": "GeoZipListState",
            "requestUrl":
"https://geodata.cdxtech.com/api/geoziplistst ate?key=YOUR_KEY&state=pa&format
=",
            "ipAddress": "127.0.0.1",
            "tokenCharge": 100
        },
        {
            "service": "GeoGender",
            "requestUrl":
"https://geodata.cdxtech.com/api/geogender?key=YOUR_KEY&zipcode=07869&format=
",
            "ipAddress": "127.0.0.1",
            "tokenCharge": 1
        },
        {
            "service": "GeoRadius",
            "requestUrl":
"https://geodata.cdxtech.com/api/georadius/?key=YOUR_KEY&zipcode=07869&mileag
e=3",
            "ipAddress": "127.0.0.1",
            "tokenCharge": 3
        },
        ...
    ],
    "usage": {
        "used": 100,
        "remaining": 1000
    },
    "duration": 0.0021103,
    "timeStamp": "2017-02-22T17:51:43.8377347-05:00"
}
```

(Above: Only a portion of the `results` array is shown for brevity. The actual response may list multiple recent service calls.)

XML Response Example (Detailed):

```
xml
CopyEdit
<Root>
  <Service>GeoDataUtility</Service>

<Url>https://geodata.cdxtech.com/api/utility/tokenusage?key=YOUR_KEY&amp;deta
ils=1&amp;format=xml</Url>
  <Status>Success</Status>
  <TokenCharge>0</TokenCharge>
  <Message />
```

```
  <TotalResults>1</TotalResults>
  <Results>
    <Service>GeoGeneral</Service>

<RequestUrl>https://geodata.cdxtech.com/api/GeoGeneral?key=YOUR_KEY&amp;zipco
de=07869&amp;format=</RequestUrl>
    <IpAddress>127.0.0.1</IpAddress>
    <TokenCharge>1</TokenCharge>
  </Results>
  <Results>
    <Service>GeoZipListState</Service>

<RequestUrl>https://geodata.cdxtech.com/api/geozipliststate?key=YOUR_KEY&amp;
state=pa&amp;format=</RequestUrl>
    <IpAddress>127.0.0.1</IpAddress>
    <TokenCharge>100</TokenCharge>
  </Results>
  <Results>
    <Service>GeoGender</Service>

<RequestUrl>https://geodata.cdxtech.com/api/geogender?key=YOUR_KEY&amp;zipcod
e=07869&amp;format=</RequestUrl>
    <IpAddress>127.0.0.1</IpAddress>
    <TokenCharge>1</TokenCharge>
  </Results>
  <!-- Additional Results entries... -->
  <Usage>
    <Used>100</Used>
    <Remaining>1000</Remaining>
  </Usage>
  <Duration>0.0017093</Duration>
  <TimeStamp>2017-02-22T17:56:50.885135-05:00</TimeStamp>
</Root>
```

JSON Response Example (Summary only, details disabled):

```
json
CopyEdit
{
    "service": "GeoDataUtility",
    "url":
"https://geodata.cdxtech.com/api/utility/tokenusage?key=YOUR_KEY&details=0",
    "status": "Success",
    "tokenCharge": 0,
    "message": null,
    "totalResults": 1,
    "results": [
      {
        "used": 100,
        "remaining": 1000
      }
    ],
    "duration": 0.0012345,
    "timeStamp": "2025-01-01T12:05:00.0000000-05:00"
}
```

(Above: When `details` is not enabled, the `results` may simply contain the used and remaining counts, or these may appear under a `usage` object as shown in detailed mode.)

In summary, the Token Usage service helps you monitor your CDXGeoData API consumption either at a summary level or in detail, ensuring you stay aware of your token usage and limits.

Glossary of Terms and Concepts

Below is an expanded glossary explaining key terms, fields, and concepts appearing in CDXGeoData services:

- **API Key:** A unique key provided by CDX Technologies to authorize access to CDXGeoData services. It must be included as the `key` parameter in every reques】.
- **Token / TokenCharge:** CDXGeoData uses a token-based billing system. Each request "costs" a certain number of tokens (shown as `TokenCharge` in the response】. For example, a simple lookup might cost 1 token, whereas an address verification might cost 5. The account's token usage is tracked in the `Usage` section of the response.
- **Usage (Used/Remaining):** The `Usage` block in responses shows how many tokens have been used and how many remain in your account or monthly quot】. E.g., `"used": 100, "remaining": 1000` means 100 tokens consumed so far, 1000 left.
- **Latitude & Longitude:** Geographic coordinates in decimal degrees indicating the location of an address or ZIP code. Latitude is north-south position (positive for northern hemisphere), longitude is east-west (negative for western hemisphere). Many services return lat/long for addresses or ZIP centroid】.
- **Bing Maps Key:** A key from Microsoft's Bing Maps required for certain services (GeoLocate, GeoRoute, etc.) that use Bing's geocoding or routing dat】. This key is distinct from the CDXGeoData API key and must be obtained from Bing Maps. In CDXGeoData, the Bing key is configured in your account settings; it is not passed as a parameter in each call.
- **HERE/Google Keys:** Not directly used in the REST services above, but mentioned in context (the Excel add-in can use Google or HERE for geocoding). If using those via CDXGeoData add-in, separate API keys are needed (not relevant for the above REST endpoints, which use Bing or internal data】.
- **Status (Success/Error):** Indicates if the request was successfully processed. `"Success"` means a valid result; `"Error"` means something went wrong (e.g., missing parameter,

invalid key, or no data found). An `Error` status is often accompanied by a `Message` explaining the error.

- **Message:** In error cases, this field provides details about the error. For example, `"Invalid Zip Code"` or `"Address not found"`. For successful calls, `Message` is usually blank or nul】.
- **City, State, County:** These fields identify the location corresponding to a ZIP code or address. City and State are the USPS preferred city name and state abbreviation. County is the county (or parish) name. Many results include these for reference, especially in ZIP-based service】.
- **FIPS Codes:** Federal Information Processing Standards codes – numeric codes for states and counties. `StateFIPS` is a two-digit code for the state, `CountyFIPS` is a three-digit code for the count】. For example, StateFIPS 34 = New Jersey, CountyFIPS 023 = Middlesex County (NJ).
- **Time Zone:** The time zone in which a ZIP or address is located (e.g., EST, CST, PST). Often given as standard time abbreviations. It may not reflect daylight vs standard time, just the base zon】.
- **Area Code:** Telephone area code(s) serving the region. Some ZIPs span multiple area codes, in which case they may be listed as "123/456" in outpu】.
- **MSA (Metropolitan Statistical Area):** A regional designation by the U.S. Census Bureau/OMB. An MSA is often identified by a five-digit code and a name. If a ZIP lies within an MSA, GeoGeneral/GeoDemographics may provide the MSA code and name (e.g., code 35620 for the New York-Newark-Jersey City MSA】.
- **Region/Division:** The Census Bureau divides the U.S. into 4 regions (Northeast, Midwest, South, West) and 9 divisions (e.g., Middle Atlantic, Pacific). These fields situate the state in those broader categorie】.
- **DaylightSavings (DST flag):** Indicates if the area observes Daylight Saving Time (`Y` or `N`). Most of the U.S. is "Y", while Arizona (except Navajo Nation) and Hawaii would be "N".
- **Address Standardization:** The process of formatting an address to USPS standards – used in GeoVerify. For example, converting "123 Main st., Apt 4" to "123 MAIN ST Apt 4". The output fields like `AddressOut`, `CityStateZipOut` are standardized result】.
- **Address Type:** In GeoVerify results, `AddressType` indicates address classification by USP】. Common values: `S` = Street or residential address, `P` = PO Box, `H` = High-rise or business, etc.
- **DPBC (Delivery Point Barcode):** A 11-digit number used by USPS to uniquely identify a mail delivery point. It's essentially the ZIP+4 plus two extra digits (often the house number or apartment identifier) and a check digit. GeoVerify returns this as `DPBC】, useful for barcoding mail.
- **ZIP+4:** The extended ZIP code including the 4-digit extension. GeoVerify outputs this as `Zip4` (just the extension) and `NineDigitZipcode` (ZIP+4 in one string】.
- **Preferred City/State:** For a given ZIP, USPS defines a single "preferred" city name (and state). Some ZIPs cover multiple municipalities or unincorporated areas; the preferred city is the one USPS uses. GeoVerify returns `PreferredCity` and `PreferredState` which may differ from the city in the input if, for example, the input used a valid but not preferred city name alia】.

- **PO Box (BoxType/BoxNumber):** If an address is a PO Box, GeoVerify will output the box information in `BoxType` (e.g., "PO BOX") and `BoxNumber` (e.g., "123") instead of a street addres】. Street addresses will leave these blank.
- **Radius Distance:** In GeoRadius, the distance from the center ZIP to another ZIP, usually measured "as the crow flies". The output `Distance` field in each result is this straight-line distanc】, and `DistanceUnit` is typically miles (or km if specified).
- **Centroid:** Many ZIP-based calculations use the geographic centroid of the ZIP Code area (the center point of the polygon that represents the ZIP). Distances in GeoDistance and GeoRadius are computed between centroids of ZIPs, not exact street addresses, which is appropriate for broad analyses but an approximation.
- **Driving Distance vs Straight-line Distance:** GeoRoute gives driving distance (following roads) which is usually longer than the straight-line distance. GeoDistance/GeoRadius give straight-line distances. For example, two ZIPs might be 10 miles apart straight-line (GeoDistance), but driving might be 12 miles due to road network (GeoRoute).
- **Travel Mode:** For GeoRoute/GeoRoutePoint, this refers to the mode of transportation: `driving`, `walking`, or `transit】. It affects travel time calculation and allowed route options.
- **Optimize (Route Optimization):** In routing, whether to optimize for shortest distance or shortest time. `distance` vs `time` vs options considering traffic/closure】.
- **Avoid Options:** Route parameters to avoid certain road types. E.g., avoiding highways or toll road】.
- **TransitDateTime / TransitTimeMode:** For transit routes, specify departure or arrival times. `transitDateTime` is the date/time, and `transitTimeMode` indicates whether it's an arrival or departure time (or last available trip】.
- **Distance Unit:** In several APIs, you can get output in miles or kilometers. Parameter `distanceUnit` (e.g., in GeoRoute, GeoRoutePoint, GeoRadius) lets you choose. Output field `distanceUnit` confirms what unit the distance is i】.
- **CSV Output:** When `format=csv`, the response is a plain text, comma-separated line (or multiple lines). The first line may be a header (field names) and following lines data. CSV output is handy for feeding directly into spreadsheets or databases, but unlike JSON/XML, it has no nested structure – e.g., GeoVerifySingle CSV will have all address fields in one line separated by comma】.
- **ZCTA:** ZIP Code Tabulation Area – a Census Bureau construct that approximates ZIP Code areas for statistical purposes (used for Census 2010 data like population by race). ZCTAs usually match ZIP codes in populated areas. CDXGeoData's demographic data is based on ZCTAs which align with ZIP codes. In rare cases, a ZIP (especially one that's only PO boxes or very new) might not have ZCTA data.
- **Confidence Level:** In geocoding (GeoLocate), a qualitative measure of how well the input matched to a result. E.g., "High" confidence means the geocoder is very sure about the resul】. "Low" might mean a guess or partial match. This helps users gauge the reliability of a geocode.
- **Reverse Geocoding:** Converting coordinates to an address (GeoLocatePoint is essentially reverse geocoding). A reverse geocode may not always return a precise street

address – if the point falls in the middle of a block, it might return an interpolated address or nearest known address, and will include a confidence measure.

- **Multi-stop Routing:** GeoRoutePoint allows routing through multiple waypoints. You pass them in as a single `wayPoints` string separated by underscore】. The route connecting them is computed, and total distance/time returned (optionally the path). This is different from computing separate pairwise routes.
- **PolyLine / Route Path:** The series of lat/long points that make up a route on a map. GeoRoutePoint can return these via `pathPoints` (in JSON) or similar structure in XML, which is useful for drawing the route on a map interface.
- **ZIP Type (Unique/Standard):** Not explicitly given in outputs above, but some ZIPs are "Unique" (for a specific organization) or "PO Box only". These might reflect in data: e.g., a unique ZIP might have no residential population. The API doesn't label ZIP type, but users can infer if population = 0 or `AddressType` returned as P.O. box only.
- **Percentages vs Counts:** Many demographic outputs provide both counts (absolute numbers) and percentages (relative). For instance, GeoRace/GeoGender give both population count and percentage. Use whichever is appropriate for your analysis (percentages allow easy comparison relative to different total populations).
- **Census 2010 vs ACS:** Most data (race, gender, total population, housing counts) are from the 2010 decennial census. Income, home value, etc., likely come from around 2010-2014 American Community Survey estimates. So remember that data is a snapshot around that era (CDX may update when new census data is available).
- **Delimiter (GeoVerifySingle):** The character used to separate address parts in a single-string address. E.g., comma (`,`), pipe (`|`), etc. You specify it with the `delimiter` parameter so the service knows how to split the string into address, city, state, etc】.
- **Private vs Public API:** CDXGeoData is a paid service – your API key usage is private to you. Always keep the key secure. The data itself (ZIP boundaries, census stats) is public domain, but the compilation and convenient access via API is what CDXGeoData provides.
- **HTTP Status Codes:** Generally, a successful query returns HTTP 200 OK with the results in the body. If your request is malformed or unauthorized (e.g., missing key or wrong URL), you might get an HTTP 400 or 401 with an error message.
- **XML Structure:** All XML responses have a root `<Root>` element, with child elements like `<Service>`, `<Status>`, `<Results>`, `<Usage>`, etc. Repeated items (like multiple results) appear as repeated `<Results>` elements or nested within one `<Results>` block depending on context.
- **JSON Structure:** JSON responses are structured as an object with fields like `service`, `status`, etc. The main data is often under a `results` array (even if only one result). For single-result services, you'll see `results: [ { ... } ]`. Always check if the data you want is nested inside an array.
- **Precision:** Numerical fields like latitude/longitude are given to sufficient precision (typically 5-6 decimal places). Distances and percentages are often given with 2 decimal places. Keep in mind rounding when summing percentages (they may total ~99.99 or 100.01 due to rounding).
- **TimeStamp:** The `TimeStamp` in responses is the server time the response was generated, in ISO 8601 format (often with timezone offset】. It can be used for logging or ensuring data recency, but it's not typically needed for data usage.

By understanding these terms and fields, developers can better interpret the output of CDXGeoData services and integrate the data correctly into their applications. Each service's detailed section above references many of these terms in context, and this glossary serves as a reference for what each means and how they relate to the data provided by CDXGeoData.