maXbox_starter160 The Open Railway Map API
鉄道地図を開く

JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays).

The OpenRailwayMap (ORM) is an OpenStreetMap-based project designed to create a detailed map of the world's railway infrastructure. It provides a variety of tools, APIs, and map styles to visualize railway data, including tracks, stations, signals, and speed limits.

Key Features and Usage

The OpenRailwayMap database is built on OpenStreetMap data and is available under the Open Database License (ODbL). The rendered map tiles are distributed under the CC-BY-SA 2.0 license. The project is non-commercial and maintained by volunteers, relying on donations for its operation.

The database supports querying railway-related information through a RESTful API. This API allows users to search for stations by name or reference code and retrieve mileage data for specific railway lines. The API documentation is hosted on GitHub.

Map Styles and Tile Access

The map tiles are rendered in Web Mercator projection and are available in different styles, such as:

- Standard: Displays railway infrastructure like tracks, stations, and switches.
- Signals: Visualizes railway signals and train protection systems.
- Maxspeed: Shows maximum speeds and speed signals for railway lines.

Tiles can be accessed via URLs in the format:

http://${s}.tiles.openrailwaymap.org/${style}/${z}/${x}/${y}.png
Here, ${s} can be replaced with subdomains (a, b, or c) for faster loading, and ${style} specifies the map style.

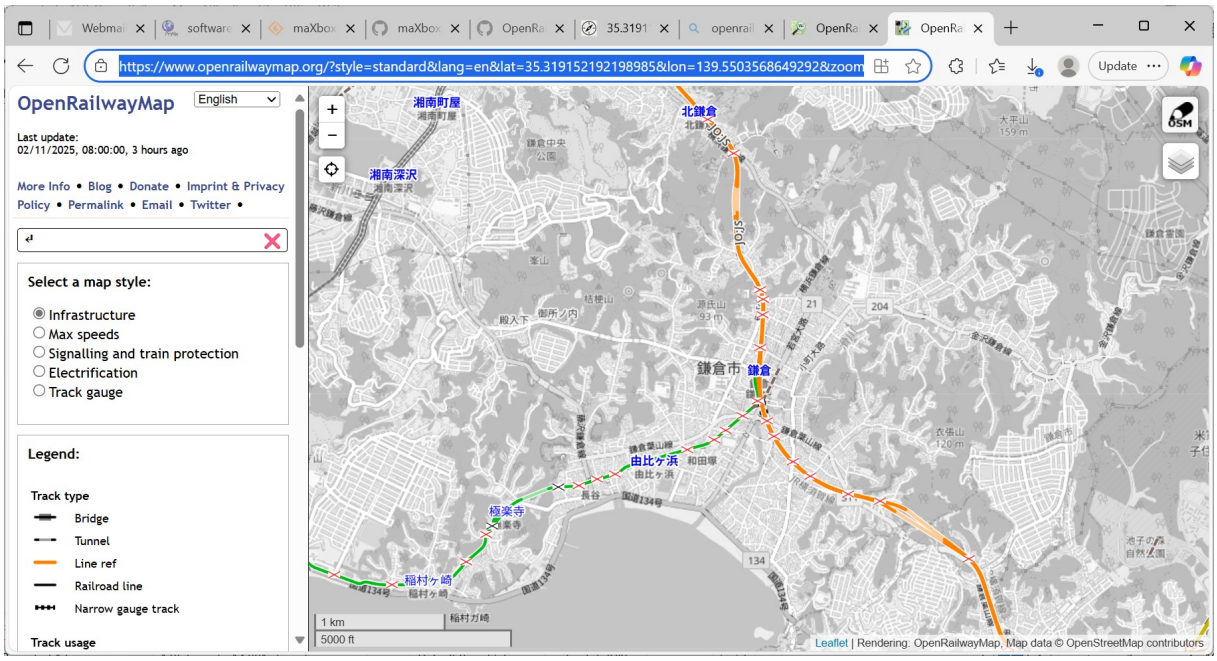
Integration with Tools

The OpenRailwayMap tiles can be integrated into various mapping libraries and tools:

- OpenLayers and Leaflet: Easily add OpenRailwayMap layers using their respective APIs.
- OsmAnd: Overlay OpenRailwayMap tiles on offline maps with the "Online maps" plugin.
- QGIS: Load OpenRailwayMap tiles as XYZ layers for GIS analysis.

The OpenRailwayMap API and tiles are free for non-commercial, small-scale applications. Commercial use requires setting up a private server. Bulk requests and misuse of headers (e.g., faking user-agent) are prohibited. Applications must include proper attribution when using the tiles or API. For example we search for Kamakura Station, Japan:

[OpenRailwayMap
https://www.openrailwaymap.org/?style=standard&lang=en&lat=35.319152192198985&lon=139.5503568649292&zoom=14]



Kamakura Station 鎌倉

The project operates without guarantees of availability or support. Users requiring high reliability are encouraged to deploy their own instances of the API or image tile server.

For further details, visit the OpenRailwayMap GitHub repository or the OpenStreetMap Wiki page.
https://wiki.openstreetmap.org/wiki/OpenRailwayMap/API

So first we call the REST-API to get a stations facility information in JSON:

```
const URL_ORM_GET9 =
            'https://api.openrailwaymap.org/v2/facility?name=%s&limit=1';

function API_GEOLocation_ORM9(AURL, aloc, aApikey: string;
                                                verbose: boolean): Tlatlong;
var Httpreq: THttpRequestC; httpres: string; jsn: TMcJsonItem;
begin
  httpreq:= THttpRequestC.create(self);
  httpreq.headers.add('Accept: application/json; charset=utf-8');
  //httpreq.headers.add('X-Api-Key:'+aAPIkey);
  httpreq.useragent:= USERAGENT5;
  httpreq.SecurityOptions:= [soSsl3,soPct,soIgnoreCertCNInvalid];
  try
    if httpreq.get(Format(AURL,[aloc])) then begin
       httpres:= (httpreq.Response.ContentAsUTF8String)
       writeln('conttype '+httpreq.Response.ContentType);
       if verbose then writ('debug back '+formatJson(httpres));
       jsn:= TMcJsonItem.Create;
       jsn.AsJSON:= httpres;
       writ('debug name: '+jsn.at(0,'name').asstring)
       writ('debug operator: '+jsn.at(0,'operator').asstring)
       result.lat:= jsn.at(0,'latitude').asnumber;
       //in the api now fixed
       result.long:= jsn.at(0,'longitude').asnumber;

       result.descript:= Format('Coords: lat %2.5f lng %2.5f %s
                                        osm_id: %s operator: %s',
                  [result.lat,result.long,jsn.at(0,'name').asstring,
                                       jsn.at(0,'osm_id').asstring,
                                       jsn.at(0,'operator').asstring]);

    end else Writeln('APIError '+inttostr(Httpreq.Response.StatusCode2));
  except
    writeln('EWI_APIHTTP:
                  '+ExceptiontoString(exceptiontype,exceptionparam));
  finally
    writeln('Status3: '+gethttpcod(httpreq.Response.statuscode2))
    httpreq.Free;
    sleep(200);
    jsn.Free;
  end;
end;
```

The API returns JSON formatted data with following fields:

- latitude: latitude
- longitude: longitude
- osm_id: OSM node ID
- rank: an importance rank calculated by taking the public transport route relations into account using this station/halt. All OSM tags present on this object. The following tags are very often in use. See the OSM wiki and Taginfo for a more comprehensive list of possible tags.

- name: name
- uic_name: UIC station name
- railway:ref: reference assigned by the operator of the infrastructure
- railway: type of the facility following Tagging rules), e.g. station, halt, junction, yard.
- operator: operator of the infrastructure

Open Railway Map:_____
conttype application/json
debug back [


```
  {
    "osm_id": 506122717,
    "name": "鎌倉",
    "railway": "station",
    "ref": null, "train": "yes",
    "name:en": "Kamakura",
    "name:es": "Kamakura JR",
    "name:it": "Kamakura",
    "name:ja": "鎌倉",
    "name:ko": "가마쿠라",
    "name:zh": "镰仓",
    "operator": "東日本旅客鉄道",
    "wikidata": "Q932895",
    "wikipedia": "ja:鎌倉駅",
    "name:ja-Hira": "かまくら",
    "name:ja-Latn": "Kamakura",
    "public_transport": "station",
    "latitude": 35.31911869967492,
    "longitude": 139.5504286,
    "rank": 12
  }
]
```


debug name: \u938c\u5009
debug operator: \u6771\u65e5\u672c\u65c5\u5ba2\u9244\u9053
Status3: SC_OK
Coords: lat 35.31912 lng 139.55043 鎌倉 osm_id: 506122717 operator: 東日本旅客鉄道
mX5 🐞 executed: 02/11/2025 08:48:43 Runtime: 0:0:4.305 Memload: 64% use

I got also a mail from a bugfix which is now solved[1]. Thank you for the bug report and the comment reminding me. I adapted the frontend JavaScript code as well because it assumed latitude and longitude to be swapped as well. It seems that the old API had this bug, too. 😳

The call for the map above has to be set with the coordinates, which we got from the facility API:

```
OpenWeb('https://www.openrailwaymap.org/?style=standard&lang=en&lat='+
                    flots(togeo.lat)+'&lon='+flots(togeo.long)+'&zoom=14');
```

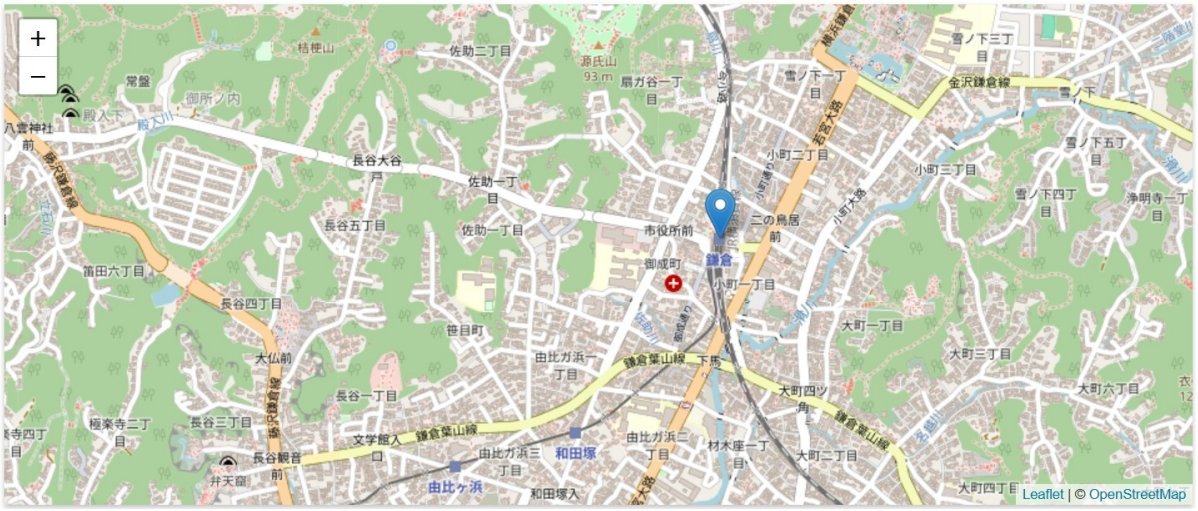
1 https://github.com/OpenRailwayMap/OpenRailwayMap-api/issues/6

Most of the UTF-8 we do de-serialize with the function *jsonunescape()*:

```
writ(jsonunescape((togeo.descript),#13#10));
```
>>> Coords: lat 35.31912 lng 139.55043 鎌倉 osm_id: 506122717 operator: 東日本旅客鉄道

A string is a sequence of zero or more Unicode characters [UNICODE].
An object is an unordered collection of zero or more name/value pairs, where a name is a string and a value is a string, number, boolean, null, object, or array.
An array is an ordered sequence of zero or more values.

The latitude 35.319118699674918 and longitude 139.5504286 shown on map.
GPS coordinates: 35° 19' 8.8273" N 139° 33' 1.5430" E



OpenWeb('https://www.latlong.net/c/?lat='+flots(togeo.lat)+'&long='+flots(togeo.long));

Most API requests should normally take no longer than 3 seconds. In any case, you should write your application so that it will time out requests after ca. 5 seconds. Applications must send a unique HTTP User-Agent. If map tiles are embedded into a website, browsers must send a valid HTTP referer instead.

And then I was searching for an unicode-enumerator and after 5h we got the JSONUnescape() function which was iterator enable: (Here's how you'd do it in Python: *text = text.encode('latin-1').decode('utf-8')* [2]

```
writeln(JSONUnescape('\u3040\u3041\u3042\u3043\u963b\u9644',#1
//https://en.wikipedia.org/wiki/Hiragana_%28Unicode_block%29
for it1:= 4 to 9 do
 for it2:= 0 to 15 do begin
   write(JSONUnescape('\u30'+itoa(it1)+inttohex(it2,1),#10)+'
   if it2 = 15 then writeln(' ')
 end;
```

Katakana is a Unicode block containing katakana characters for the Japanese and Ainu languages.

2 Unescape and get Unicode String in Kotlin - Stack Overflow

```
//https://en.wikipedia.org/wiki/Hiragana_(Unicode_block)
for i:= 10 to 15 do
  for j:= 0 to 15 do begin
   write(JSONUnescape('\u30'+inttohex(i,1)+inttohex(j,1),#10)+
   if j = 15 then writeln(' ')
end;
```

IS THERE ANY DIFFERENCE BETWEEN SHOTOKAI & SHOTOKAN?
No, they are the name of Shotokai's Hombu (main) Dojo (Shotokan) and the name of the association (Shotokai).

17/09/2025 – 09:27 Mt Fuji from Chureito Pagoda

17/09/2025 – 10:06 Mt Fuji from Chureito Pagoda

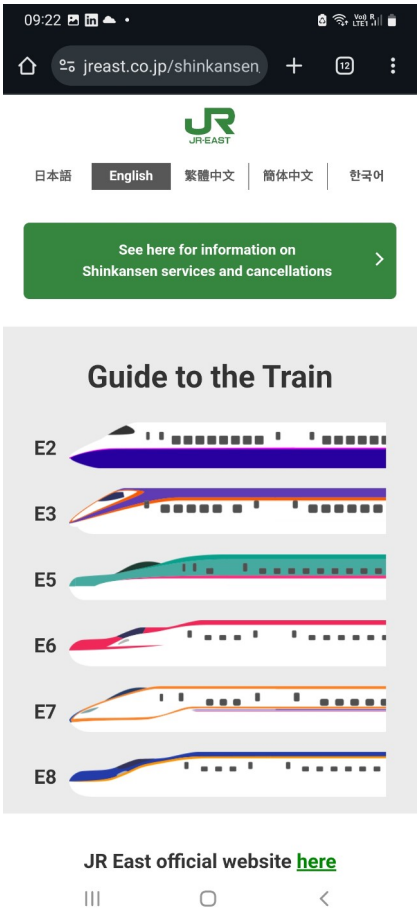
Chureito Pagoda (Buddhist) is an essential building of Arakurayama Sengen shrine (Shinto) , located in Fujiyoshida City, in Yamanashi prefecture. The pagoda towers on the heights of the shrine's park, facing the city, with an unobstructed view on Mount Fuji.

The best way for travellers to spot Mount Fuji when traveling from Kyoto is through the windows of the Tokaido Shinkansen (bullet train) rather than from natural viewpoints in Kyoto itself. You cannot see Mount Fuji from standard tourist locations in Kyoto city or from public viewpoints due to the distance and intervening mountains.

Shinkansen (Bullet Train) 新幹線

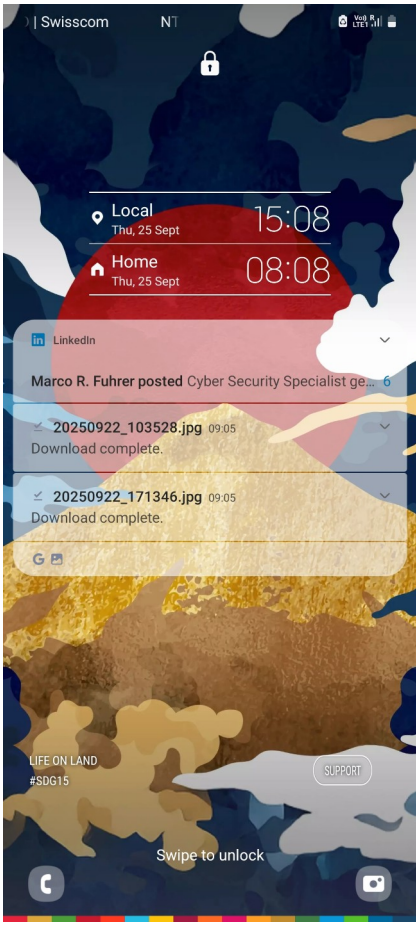
- Most reliable and iconic view for tourists: Take the Tokaido Shinkansen from Kyoto to Tokyo.
- Where to sit best: For the best view, sit on the left side of the train facing the direction of travel toward Tokyo (a seat E in standard class, D in Green Car).
- Where you see Fuji: The mountain comes into view for several minutes near Shin-Fuji station, about halfway between Kyoto and Tokyo, weather permitting.

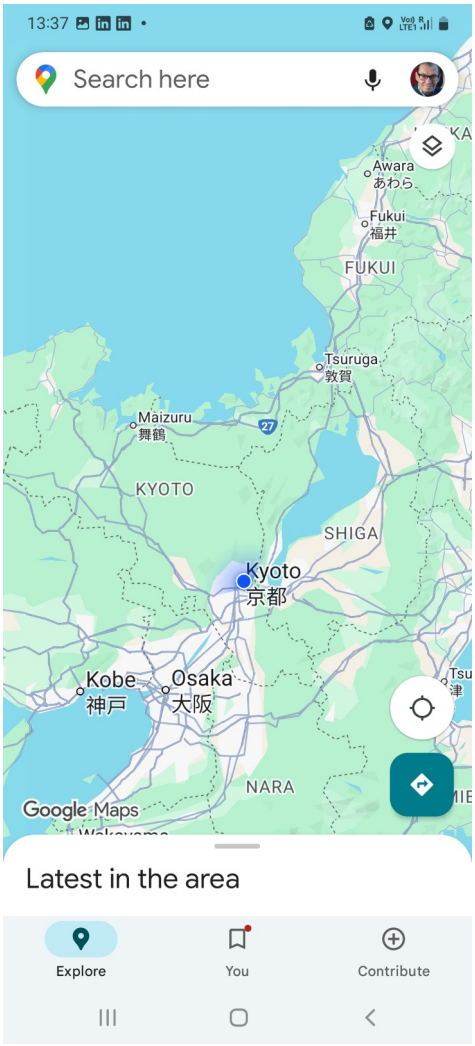
Tips: Fuji is visible faster when heading toward Tokyo than in the opposite direction. Weather is crucial—a clear day is needed for a good view.



The windows of the Shinkansen bullet train between Kyoto and Tokyo are by far the best and most practical way for visitors in Kyoto to see Mount Fuji. Public viewpoints or city locations in Kyoto do not offer a view of Fuji-san.

You can see Mount Fuji from the train on several different train journeys. However, one train offers the best views of Japan's iconic mountain: the Fujisan View Express. The Fujisan View Express features an interior that looks more like a cafe than a train, and staff on board helps for travel experience.





The Fuji Five Lakes area north of Mount Fuji is squeezed between the majestic mountain and the Japanese Alps. Like many other places in Japan, a small community railway connects the area.
But this community railway is different. It does not only serve the community (with local trains running once every 30 minutes), but it also serves visitors who want a particularly great view of Mount Fuji.
Another great time for scenic views is during Sakura season. Typically, the cherry blossoms flower from mid-March to mid-April, although it can be considerably earlier.
The effects of global warming also affect cherry blossoms.

The other season when the views becomes extra scenic is Fall. The bright yellow and red leaves mix with the green of conifers to form a dusty, purplish, deep orange, which makes for a great contrast with the majesty of Mount Fuji.
Display: 富士山, 小山町, 駿東郡, 静岡県, 日本

It rises to 12,388 feet (3,776 metres) near the Pacific Ocean coast in Yamanashi and Shizuoka ken (prefectures) of central Honshu, about 60 miles (100 km) west of the Tokyo-Yokohama metropolitan area.

Geolocation GUI & API

To get the code (coordinates) for the geolocation we need an API:

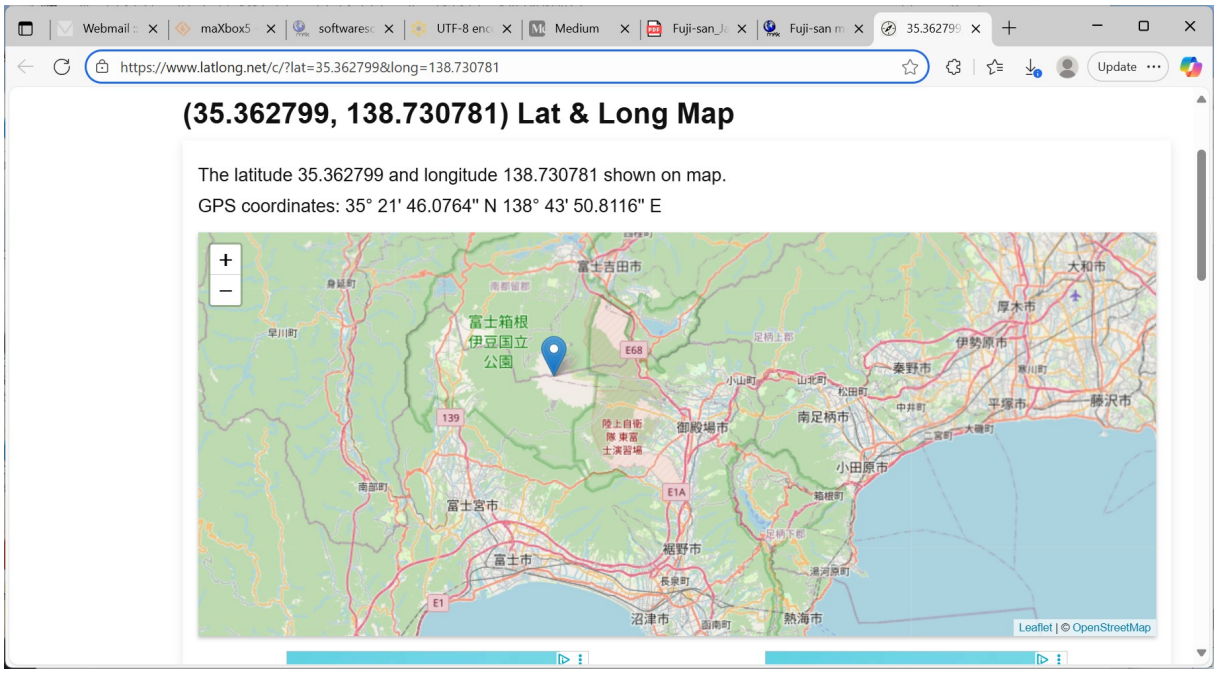
```
const GEOLOCURL9 = 'https://nominatim.openstreetmap.org/search?format=json&q=%s';
      URL_APILAY_GEO = 'https://api.apilayer.com/geo/country/capital/%s';
```

The Nominatim Geolocation API with OpenStreetMap is a service that allows web applications to obtain the geographical location of a device. This API can be particularly useful for applications that need to provide location-based services or features, such as mapping, navigation, or location-aware content.

```
function API_GEOLocation_OSM9(AURL, aloc, aApikey: string):
                                                        Tlatlong;
var Httpreq: THttpRequestC; httpres: string;
    jsn: TMcJsonItem;
begin
  httpreq:= THttpRequestC.create(self);
  httpreq.headers.add('Accept: application/json; charset=utf-8');
  //httpreq.headers.add('X-Api-Key:'+aAPIkey);
  //httpreq.SecurityOptions:= [soSsl3,soPct,soIgnoreCertCNInvalid];
  try
    if httpreq.get(Format(AURL,[aloc])) then begin
       httpres:= (httpreq.Response.ContentAsUTF8String)
       //writeln('conttype '+httpreq.Response.ContentType);
       httpreq.useragent:= USERAGENT5;
       writ('debug back '+formatJson(httpres));
       jsn:= TMcJsonItem.Create;
       jsn.AsJSON:= httpres;
       result.lat:= jsn.at(0,'lat').asnumber;
       result.long:= jsn.at(0,'lon').asnumber;
       result.descript:=
                Format('Coords: lat %2.5f lng %2.5f %s osm_id: %s ',
                  [result.lat,result.long,jsn.at(0,'name').asstring,
                                    jsn.at(0,'osm_id').asstring]);
    end else Writeln('APIError
                          '+inttostr(Httpreq.Response.StatusCode2));
  except
    writeln('EWI_APIHTTP:
                 '+ExceptiontoString(exceptiontype,exceptionparam));
  finally
    writeln('Status3: '+gethttpcod(httpreq.Response.statuscode2))
    httpreq.Free;
    sleep(200);
    jsn.Free;
  end;
end;
```

The Geolocation API works by accepting an HTTPS request with data from cell towers and WiFi access points that a mobile client can detect. It returns latitude and longitude coordinates along with a radius indicating the accuracy of the result. This is especially useful for devices that do not have native geolocation features as a GPS.
Then we open with openWeb() a call to visualise the location:

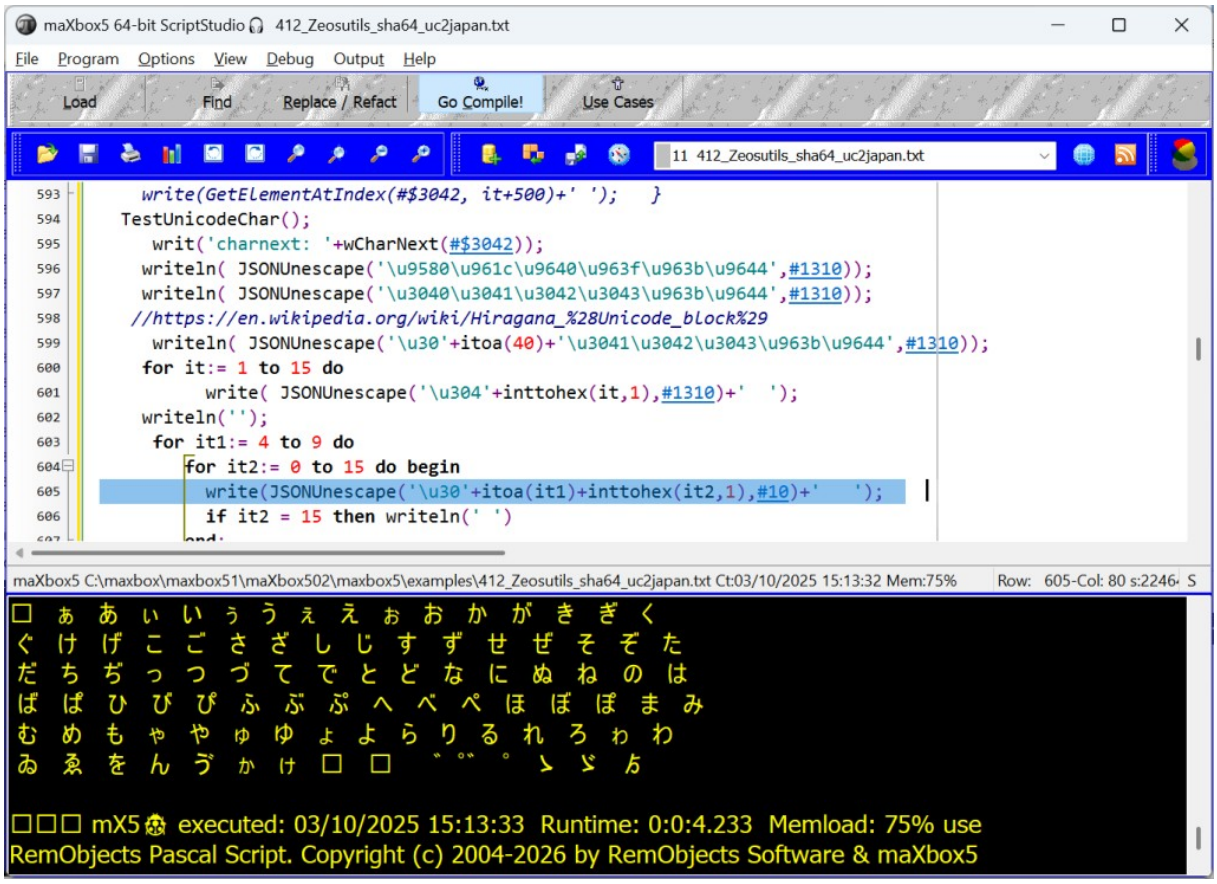
```
  //-------------------Input From To Example-----------------//
  //fromlat:=  '46.9479';    fromlong:= '7.44744';          //Bern
  //tolat:=    '49.1857';    tolong:=   '-2.1102';          //St. Jersey
  //-----------------------------------------------------------//

t_latlong:=
API_GEOLocation_OSM9(GEOLOCURL9,'Fuji-san, Japan','L_APIKEY');

writeln('OSM _from: '+f_latlong.descript);
writeln('OSM _to:   '+t_latlong.descript);

OpenWeb('https://www.latlong.net/c/?lat='+flots(t_latlong.lat)
                                  +'&long='+flots(t_latlong.long));
```



www.latlong.net

Here is an example of how to make a control request as debug understanding to the Geolocation API using cURL and `ThttpRequestC` in maXbox (If you're in a client-side environment, investigating about cross-browser support is mandatory for a well supported web app):

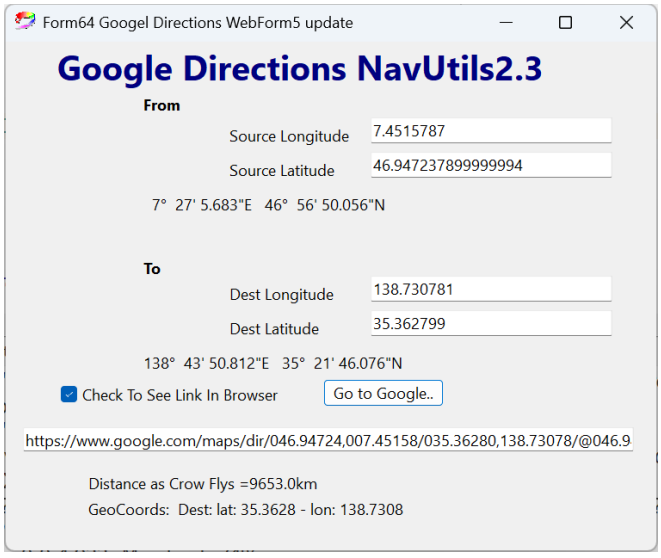
```
debug back [
  {
    "place_id": 244435545,
    "licence": "Data © OpenStreetMap contributors, ODbL 1.0.
                                                http:\/\/osm.org\/copyright",
    "osm_type": "node",
    "osm_id": 714354378,
    "lat": "35.3627990",
    "lon": "138.7307810",
    "class": "natural",
    "type": "volcano",
    "place_rank": 18,
    "importance": 0.6066588006906172,
    "addresstype": "volcano",
    "name": "富士山",
    "display_name": "富士山, 小山町, 駿東郡, 静岡県, 日本",
    "boundingbox": [
      "35.3627490", "35.3628490",
      "138.7307310", "138.7308310"
    ]
  } ]
```



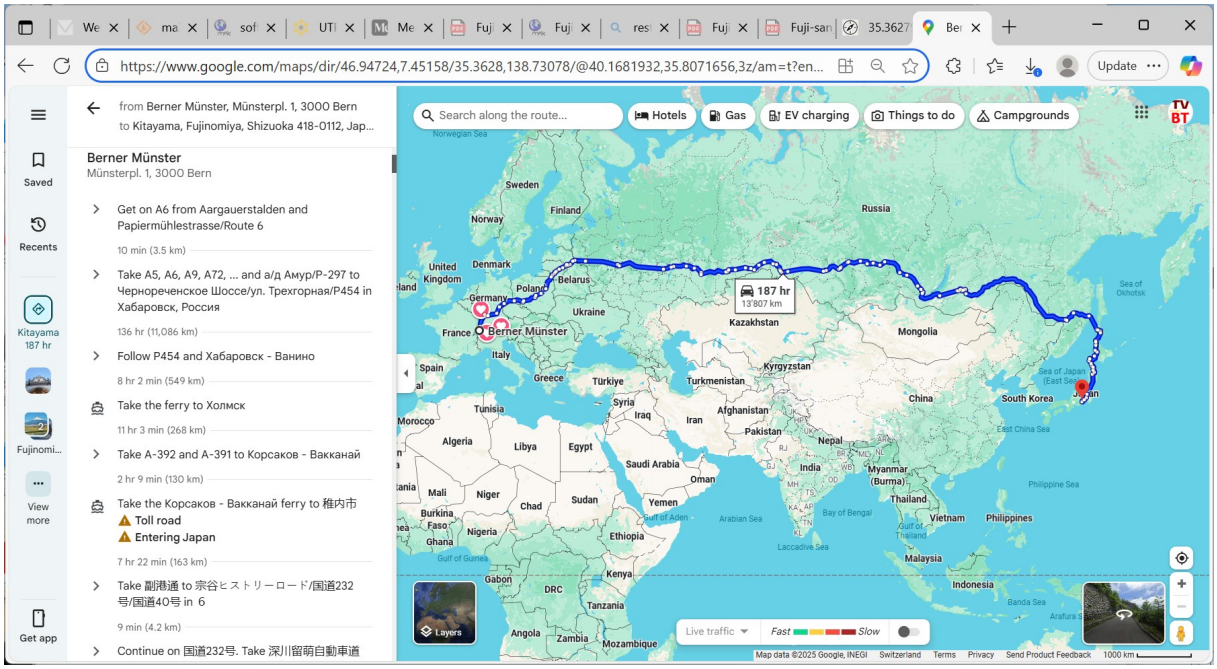
JSON string escaping is the process of converting special characters within strings into a format that can be safely transmitted and parsed as valid JSON. This involves prefixing certain chars with backslashes or converting them to Unicode escape sequences.

We need to mention that this does not do trivial escaping by default and the result might be much more escaped than you want. For ex., quotes, triangle brackets and many letters will be converted to Unicode chars and will only be usable if you unescape at the other end.

To get the google directions API we can build a GUI or call direct on the maXbox console to be independent of the GUI:



The result will be a REST-Url string sequence that can be opened in a browser:



[Berner Münster, Münsterpl. 1, 3000 Bern to Kitayama, Fujinomiya, Shizuoka 418-0112, Japan - Google Maps]

With another REST-Client we pass the URI encoded URL as a resource to the get() method from `TRestClient`:

```
function GetRestCountriesOSM(const URLCountry, faddress: string;
                                              map: boolean): Tlatlong;
var encodURL: string; urlid: TIduri;
    mapStrm: TStringStream; //jconv:TJSONConverter ;
    jo: TJSONObject; arest: TRestResource; display, wres: string;
begin
   //datafeed:= 'FranceTest';
   //encodURL:= Format(URLCountry,[HTTPEncode(Datafeed),APIKEY]);
   urlid:= TIdURI.create('');
   encodurl:=
      urlid.URLEncode('https://nominatim.openstreetmap.org/search?
                                          format=json&q='+fAddress);

   with TRestClient.create(self) do begin
     aRest:= Resource(encodURL);
     //HttpGet(EncodURL, mapStrm); //WinInet
     arest.ContentType('application/json; charset=utf-8');
     writ('content types '+arest.GetContentTypes);
     //arest.Authorization('Bearer '+MAPGPT_APIKEY2);
     arest.header('accept','application/json');
     ConnectionType:= hctWinInet;
     OnResponse:= @TRestOnResponseEvent2;
     //arest.post(mapStrm)
     wres:= aRest.get;
     writ(wres)
   end;
```

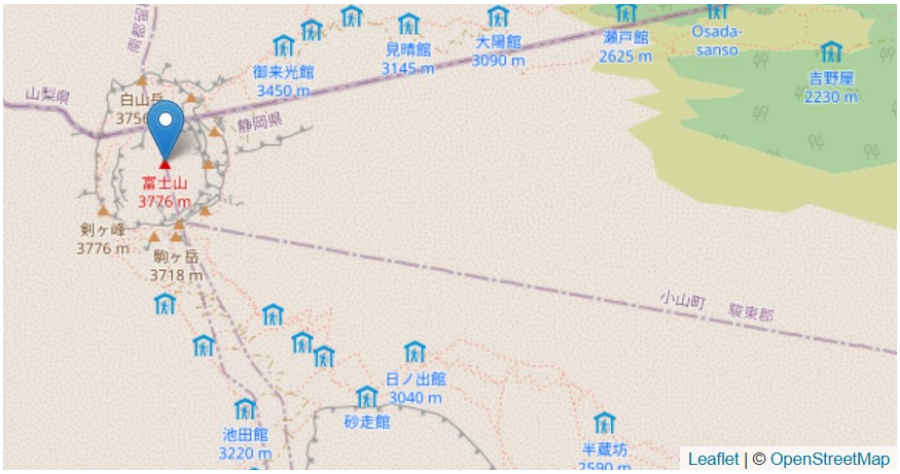
The content-type has to be set to UTF8 to get the Japanese chars from above:

content types application/json; charset=utf-8
The MIME media type for JSON text is application/json. The default encoding is UTF-8, but it's also necessary to realize what type of data is expected in your application, for example text/html.
Firebug will add a tab to the response showing you the JSON data formatted. If the MIME type is different, it will just show up as 'Response content':

```
procedure TRestOnResponseEvent2(ARestClient: TrestClient;
                                ResponseCode: Integer;
                                const ResponseContent: string);
 begin
   writeln('@addr of:'+objtostr(arestclient));
   //writeln('response cont: '+responsecontent)
   writeln('response code: '+itoa(responsecode));
   writeln('enabled compression'+
               botostr(arestclient.EnabledCompression));
   writeln('content-encoding:'+
               arestclient.responseheader['Content-Encoding']);
   writeln('verifycert: '+botostr(arestclient.verifycert));
 end;
```

You should in particular verify that you have set a custom HTTP referrer or HTTP user agent (windown.useragent:=) that identifies your application, and that you are not overusing the service with massive bulk requests. Otherwise you get following message:
* You have been blocked because you have violated theusage policyof OSM's Nominatim geocoding service. Please be aware that OSM's resources are limited and shared between many users. . *

The OSM Nominatim is a search engine for OpenStreetMap data. From this site you may search for a name or address (like Bahnhof, Graz, Austria), or look up place names by geographic coordinates.
Each result will link to details page where you can inspect what data about the object is saved in a database and investigate how the address of the object has been computed (URI & JSON for example).



Taiheikaku-Bridge, — Location Heian Jingu Shrine

Max Kleiner, Text, Code & Photos, November 2025

Ref: OpenRailwayMap/API – OpenStreetMap Wiki
https://www.openrailwaymap.org/
Fujisan View Express - The Best Way To See Mt Fuji On A Train
Japan 2025 – Breitschblog
Which JSON content type do I use? - Stack Overflow
https://wiki.openstreetmap.org/wiki/OpenRailwayMap/API

Script: https://github.com/maxkleiner/OpenRailwayMap-api/blob/master/tests/1444_OpenRailMap_Geolocation_distance14_uc_py.txt

Geocoding IV. Just like every actual house has its… | by Max Kleiner | Nerd For Tech | Medium

Fuji-san Moments. Fuji-san Moments 2 富士山 | by Max Kleiner | Oct, 2025 | Medium

maXbox4exe Digitally signed by maXbox4exe
Date: 2025.11.02 14:01:43 +01'00'