**Google Maps Wrapper**

Documentation

Author:

<https://github.com/ScryptSnake/GoogleMapsWrapper>

Version: 10/2024

# Project Overview:

GoogleMapsWrapper is a project that interfaces with the Google Maps API ecosystem. While Google offers dozens of API’s for geocoding/mapping, this software targets only 3 primary API services:

1. GeocodeAPI: acquires geolocation data, to include street address, local municipality, state, country and other information. Elevation information can also be returned from this service. All requests in this service are passed a single GPS coordinate parameter to retrieve data.
2. StaticMapsAPI: produces a static image of a map. Maps are highly customizable and may contains placemarks and other objects on the map.
3. JavascriptAPI: spawns a web browser to work interactively with a map. Data can be sent/received from .NET.

# Project Status:

This project is under development--particularly, the JavascriptAPI is incomplete.

# Recommended Usage:

This software is best utilized as a supplement to an existing project, but can be used for simple requests (see Usage examples).

# Usage Prerequisites:

This software requires a valid API key from Google.

See: <https://developers.google.com/maps>

# Usage Disclaimer:

This software attempts to optimize the use of requests to Google’s API. However, users are solely responsible for all requests made from their accounts. The developer assumes no liability for any charges incurred due to excessive requests or for the frequency of requests sent to Google’s services. Further, the developer does not make any guarantees of accuracy of data yielded from this software.

# General Architecture:

This application can be conceptually divided into 7 primary components:

1. Api (namespace: *GoogleMapsWrapper.Api…*). These are higher level objects that contain the core functionality of end usage. The primary entry point of the application is *GoogleMapsApi* which exposes each “sub-API” (*GeocodeApi* and *StaticMapsApi*).
2. Engine (namespace: *GoogleMapsWrapper.Engine…*). This object is responsible for sending requests and receiving responses. Requests are constructed in an *API* and passed to the engine to process. The engine is capable of retrieving responses in two types: JsonDocument and byte array.
3. Requests (namespace: *GoogleMapsWrapper.Requests…*). This object is constructed by an *API* and contains relevant details about the request, as well as it’s Uri. Note that the user’s API key should never be present in this object (See security).
4. Responses (namespace: GoogleMapsWrapper.Responses…). A response implements the generic interface IResponse<T> is returned from the *engine.* It contains details about the request, as well as relevant data returned. Additionally, the response contains a Parse() method allowing the consumer to parse the response with a *Parser* into a *container* (See Parsers, Containers).
5. Parsers (namespace: *GoogleMapsWrapper.Parsers*). A parser is an object which is responsible for taking a response of some data type (JSON or byte array) and containerizing the data into an IContainer (container).
6. Containers (namespace: *GoogleMapsWrapper.Containers…):* These are DTO or data containers that hold parsed information from a response.
7. Elements (*GoogleMapsWrapper.Elements…):* Elements are objects which model map features. This includes *Map, Marker, Polyline*. Elements are used throughout the application, primarily for passing as a parameter to an API method.

Note: The javascript API is conceptualized as a different component entirely.

# Security

A user’s API key should be contained within the IConfiguration passed at the application entry point (GoogleMapsWrapper.Api.GoogleMapsApi). The *engine* is also constructed via the IConfiguration and therefore has access to the key. When requests are created and passed to the engine for sending, the provided request’s Uri does **not** contain the user’s key. Likewise, when a response is returned from the engine, the *SentRequest* property (stored request) of the response does not contain the Uri with the key. The engine constructs a private *KeyedRequest* object from the received request, which contains the key, and therefore can be appended to the Uri and sent via HttpClient. This choice in design was chosen to ensure stored requests within responses (*SentRequest* property) do not expose the key to other parts of the application.