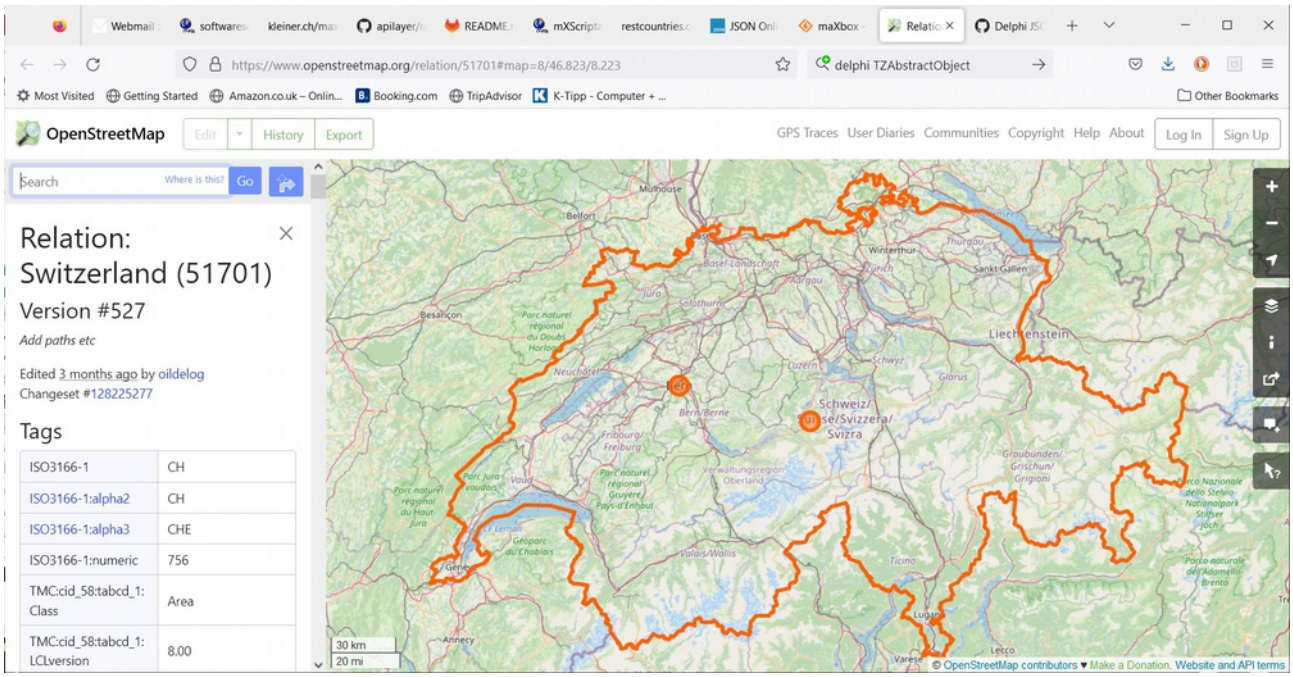
Restcountries API

maXbox Starter 104 – Countries API microservices.

"A map is worth a thousand words.
   An interface is worth a thousand maps.".

The purpose of this API is to get information about countries via a RESTful API. It supports restcountries and providing it as a free solution for developers.
The restcountries project has been acquired by APIlayer. APILayer is an API marketplace where also your API can reach mores audiences.

The result can be a map like screenshot below:



Pic: 1180_osm_map_tutor104.jpg

As you can see we use OpenStreetMap which is a map of the world, created by people like you and free to use under an open licence.

This project of Restcountries is inspired on restcountries.eu by Fayder Florez. Although the original project has now moved to a subscription base API, this project is still Open Source and Free to use and we don't need an API key before.

We use *WinHttp.WinHttpRequest* with *HTTPGet*, *JSONObjects* and *TGraphics* library with loading and testing the REST-client. Also we don't need to pass the API-key as a request-header, so in any case you can get a key first if you want at: https://apilayer.com/marketplace
More about the JSON lib you get at:

http://www.softwareschule.ch/examples/jsonlib.htm

Simply put, an endpoint of one end of a communication channel. When an API interacts with another system, the touchpoints of this communication are considered endpoints. For APIs, an endpoint can include a URL of a server or service. Each endpoint is the location from which APIs can access the resources they need to carry out their function based on versions.

Currently in restcountries there are 3 versions:

1. Version 2 is the original version from restcountries.eu by the origin of Fayder Florez
2. Version 3 is the implementation from this project
3. Version 3.1 adds named values to the flags object like this

For our example we use version 3.1.
The data represents is JSON data with all the text extracted and even the language of the text to scan is auto detected. Before we dive into code this is the main part of the script:

```
Const
  RESTCountries = 'https://restcountries.com/v3.1/name/%s';
  RESTCountriesC = 'https://restcountries.com/v2/capital/%s';


function GetRestCountriesJSON2(const URLCountry, Datafeed, APIKEY: string):
                                                                    string;
var encodURL, tmpstr: string;
    mapStrm: TStringStream; jo,jo2: TJSONObject; ajar: TJSONArray;
    //jconv:TJSONConverter;
begin
  //datafeed:= 'Vienna';
  encodURL:= Format(URLCountry,[HTTPEncode(Datafeed),APIKEY]);
  mapStrm:= TStringStream.create('');
  try
    HttpGet(EncodURL, mapStrm);  //WinInet
    mapStrm.Position:= 0;
     writeln('  ');
    // jo:= TJSONObject.Create4(HIDDENT4VALID);

    //important hack: we have to replace json node from [{ to { !
    tmpstr:= StringReplace(mapStrm.datastring, '[{"name',
                     '{"name',[rfReplaceAll, rfIgnoreCase]);
    writeln(tmpstr)
    writeln('  ');
    jo:= TJSONObject.Create4(''+tmpstr+'}');

    writeln('capital: '+(jo.getstring('capital')));
    writeln('nativename: '+

jo.getjsonobject('name').getjsonobject('nativename').getjsonobject('fra').getstr
ing('common'));
    writeln('len translations: '+itoa(jo.getjsonobject('translations').length))
    jo2:= jo.getjsonobject('translations');
    ajar:= jo.names;
    writeln('opt: '+ajar.optstring(6))
    writeln('len node names: '+itoa(ajar.length))
    for it:= 0 to jo2.length-1 do
       writeln(jo2.getstring(jo2.keys[it]));
```

```
    ajar:= jo.getjsonarray('borders');
    writeln('len names borders: '+itoa(ajar.length))
    for it:= 0 to ajar.length-1 do
             writeln(ajar.getstring(it));
    jo2:= jo.getjsonobject('languages');
    writeln('len names languages: '+itoa(jo2.length))
    for it:= 0 to jo2.length-1 do begin
      //writeln(jo2.getstring(it));
      writeln(jo2.getstring(jo2.keys[it]));
    end;
    writeln(jo.getjsonobject('flags').getstring('png'))
    openweb(jo.getjsonobject('maps').getstring('openStreetMaps'));
  except
    //writeln('Error: '+mapstrm.datastring);
    writeln('E: '+ExceptiontoString(exceptiontype, exceptionparam));
  finally
    mapStrm.Free;
    encodURL:= '';
    jo.Free;
    //ajar.Free; //jo2.Free;
  end;
end;
```

The main part function opens a connection with *HttpGet(EncodURL, mapStrm);*, invokes the API and results a stream which we convert to a datastring. A RESTful API needs to have one and exactly one entry point. The URL of the entry point needs to be communicated to API clients so that they can find the API. Technically speaking, the entry point can be seen as a singleton resource that exists outside any collection.

You can modify a lot of member parameters in JSON as you like and interact with the API from many languages. The API export format is JSON, e.g. our name/capital call see below:

Pic: 1180_jsonstruct_tutor104.png

To validate JSON we use jsonlint:
https://jsonlint.com/

JSONLint is a validator and re-formatter for JSON, a lightweight data-interchange format. Copy and paste, directly type, or input a URL in the editor above and let JSONLint tidy and validate your messy JSON code.
I had to modify the Json root from [{ to { to get a valid parse result:

```
/imoortant hack: we have to replace json node from [{ to { !
    tmpstr:= StringReplace(mapStrm.datastring, '[{"name',
                           '{"name',[rfReplaceAll, rfIgnoreCase]);
```

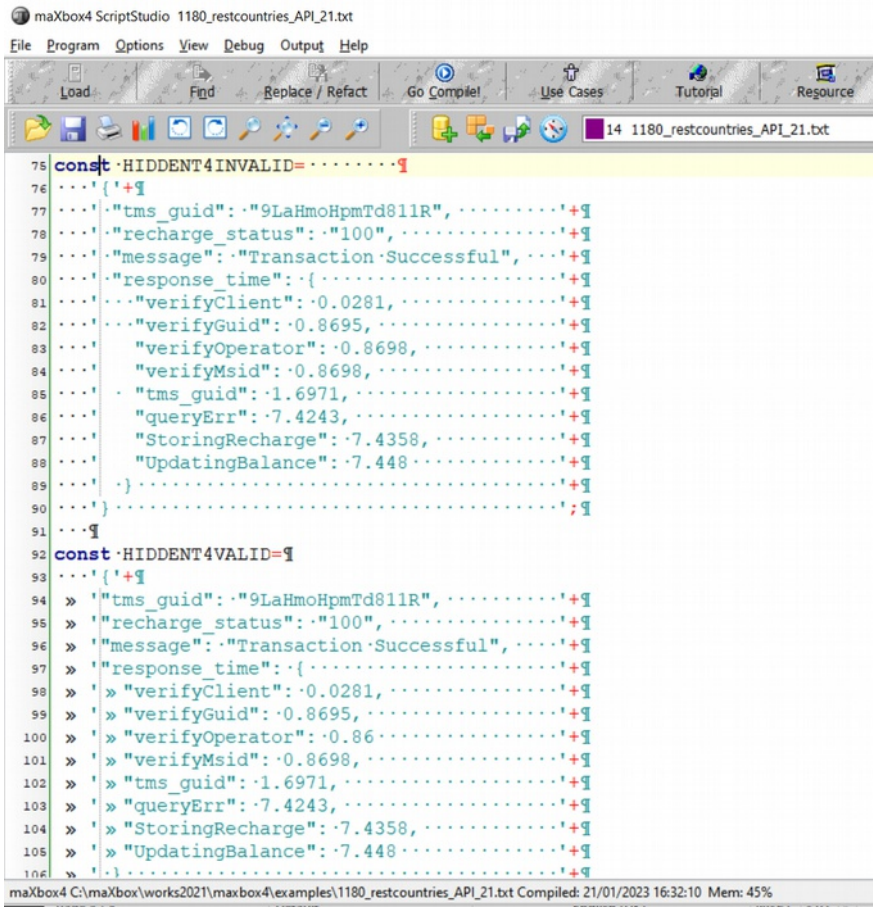
JSONLint is by the way an online editor, validator, and reformat tool for JSON, which allows you to directly type your code, copy and paste it, or input a URL containing your code. It will validate your JSON content according to JS standards, informing you of every human-made error, which happens for a multitude of reasons - one of them being the lack of focus, for example a not valid and a valid JSON:

```
const HIDDENT4INVALID=
   '{'+
   ' "tms_guid": "9LaHmoHpmTd811R",          '+
   ' "recharge_status": "100",               '+
   ' "message": "Transaction Successful",    '+
   ' "response_time": {                      '+
   '   "verifyClient": 0.0281,               '+
   '   "verifyGuid": 0.8695,                 '+
   '   "verifyOperator": 0.8698,             '+
   '   "verifyMsid": 0.8698,                 '+
   '   "tms_guid": 1.6971,                   '+
   '   "queryErr": 7.4243,                   '+
   '   "StoringRecharge": 7.4358,            '+
   '   "UpdatingBalance": 7.448              '+
   '  }                                      '+
   '}                                        ';

const HIDDENT4VALID=
   '{'+
      '"tms_guid": "9LaHmoHpmTd811R",           '+
      '"recharge_status": "100",                '+
      '"message": "Transaction Successful",     '+
      '"response_time": {                       '+
      '     "verifyClient": 0.0281,             '+
      '     "verifyGuid": 0.8695,               '+
      '     "verifyOperator": 0.86              '+
      '     "verifyMsid": 0.8698,               '+
      '     "tms_guid": 1.6971,                 '+
      '     "queryErr": 7.4243,                 '+
      '     "StoringRecharge": 7.4358,          '+
      '     "UpdatingBalance": 7.448            '+
      ' }                                       '+
   '}';
```

At a first sight you don't see a difference but it depends on the non printable characters like tab or space on the structure. The grey shaded areas are the problem for a non valid result.
If you use a Win computer for example you may end up with different results. This is possibly due to the way Win handles newlines or tabs.

Essentially, if you have just newline characters (\n) in your JSON and paste it into JSON Lint from a Win computer, it may validate it as valid erroneously since Win may need a carriage return (\r) as well to detect newlines properly. As a solution, either use direct URL input as we use, or make sure your content's newlines match or the code-page of UTF-8 match the architecture your system expects!



pic: 1180_jsonstruct_validator104.png

Using JSONLint, you can quickly find any errors that might've occurred, allowing you to focus more on the rest of your script-code than on a tiny error itself. But the hack I said before (replace [{ to { took me many hours cause few people said to replace from [{ to {[{, which also means in a context of arrays use [] not {} as the outer brackets but my library expects opening and closing curly brackets.

{"official":"Swiss\u0020Confederation","common":"Switzerland"}

A cool feature is the key-value iterator of a Tstringlist in combination with a hashmap as another stringlist:
The Names and Values by default have to be separated by =, in style of Windows INI files.

```
  writeln('len names languages: '+itoa(jo2.length))
  for it:= 0 to jo2.length-1 do
        writeln(jo2.getstring(jo2.keys[it]));
```

So the function *keys: TStringList;* can we use to get values as a string:

len names languages: 4
French
Swiss German
Italian
Romansh
https://flagcdn.com/w320/ch.png


```
function TJSONObject.keys: TStringList;
var
i : integer;
begin
 result:= TStringList.Create;
 for i:= 0 to myHashMap.Count-1 do begin
  result.add (myHashMap[i]);
 end;
end;
```

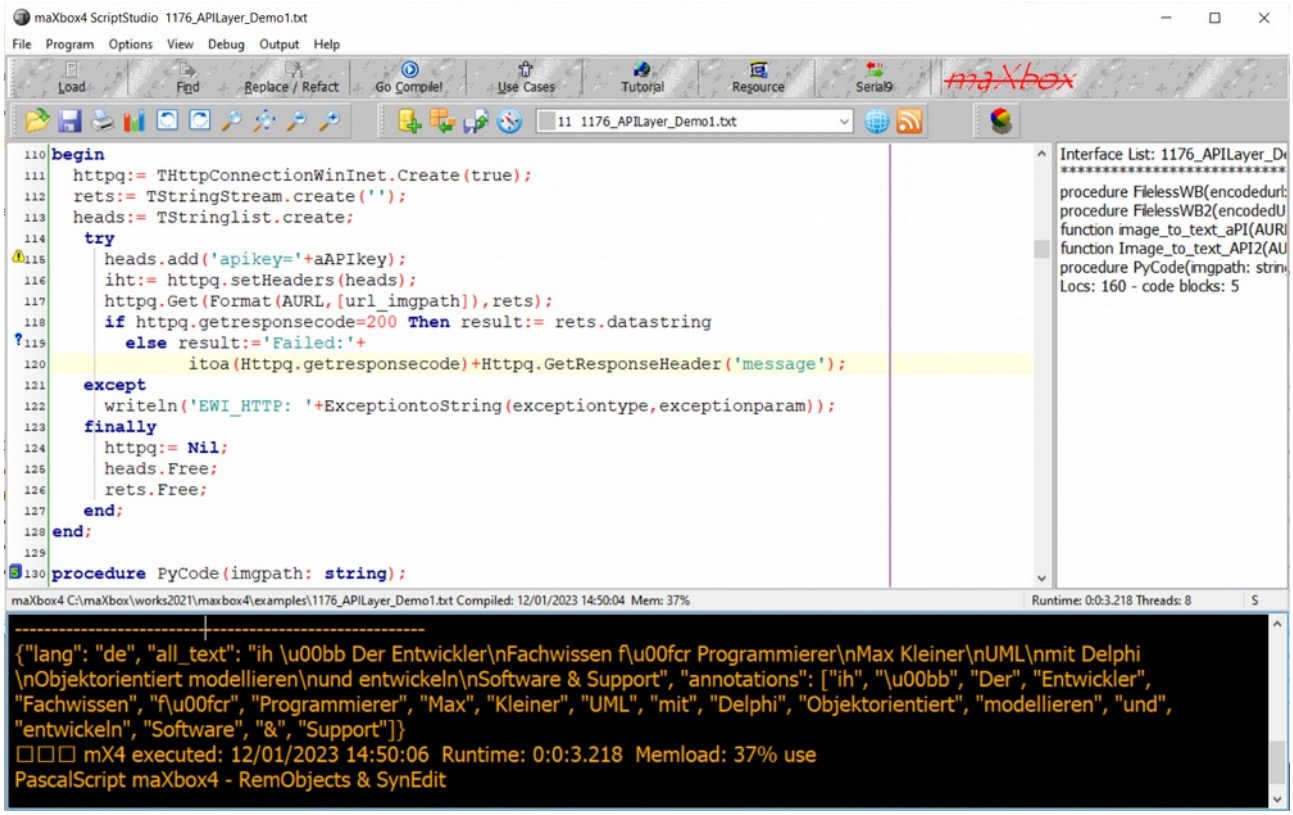
```
TJSONObject = class (TZAbstractObject)
private
 myHashMap: TStringList;
public
```

IndexOf Unlike find, IndexOf is used to find the location specified by a string. Call IndexOfName to locate the first occurrence of a name-value pair where the name part is equal to the Name parameter or differs only in case. IndexOfName returns the 0-based index of the string. If no string in the list has the indicated name, IndexOfName returns -1.

```
function TJSONObject.has(key: string): boolean;
begin
 result:= self.myHashMap.IndexOf(key) >= 0;
end;
```

IndexOf() overrides the ancestor method TStrings.indexOf(). It tries to optimize the search by executing a binary search if the list is sorted. The function returns the position of S if it is found in the list, or -1 if the string is not found in the list.

Furthermore the API access is provided in a REST-like interface (Representational State Transfer) exposing database resources or pre-trained models in a JSON format with content-type in the Response Header. Note: If a programming language is not listed in the Code Example from the live demo, you can still make API calls by using a HTTP request library written in our programming language, as we did with GET or POST.



Pic: 1176_apilayer_demo_mX4.png

Conclusion:

Restful Countries API allows users to explore the entire database for information on countries and their states, presidents, flag, population, covid19 stats and others.
Restful Countries API is organized around REST. Our API has predictable resource-oriented URLs, returns JSON-encoded responses and uses standard HTTP response codes, and verbs.

Reference:

https://github.com/maxkleiner/restcountries

https://apilayer.com/docs

Doc and Tool: https://maxbox4.wordpress.com
Script Ref: 1180_restcountries_API_21.txt

Appendix: A Delphi JSON Serialization with maXbox4 integration:
http://www.softwareschule.ch/examples/jsonlib.htm

Max Kleiner 21/01/2023