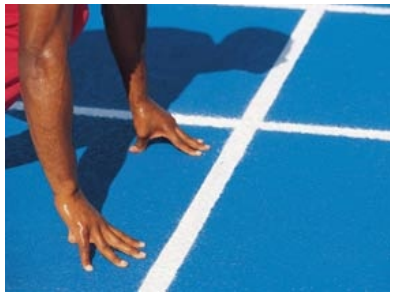
maXbox



OpenWeatherMap

Ein Online Service vom Feinsten

1.1 Better Weather mit OpenWeatherMap

Mal Hand aufs Herz, das Wetter spielt doch verrückt. Da gilt es kühlen Kopf zu bewahren und am besten selbst einen Wetterdienst einzurichten.

”OpenWeatherMap” ist ein Onlineservice, der eine frei nutzbare und gut dokumentierte Programmierschnittstelle (API) für Wetterdaten, Wetter-vorhersagen, sowie historische Wetterdaten für Entwickler von Events, Webanwendungen und mobilen Geräten bereitstellt. Vor allem die Prognosen stossen auf reges Interesse.

Die Comunity ist gross und mit allen Kanälen ausgestattet, Twitter z.b. wird viel genutzt, vor allem wenns regnet: @OpenWeatherMap. OWM verwendet dann standardmässig OpenStreetMap zur Darstellung der Wetterkarten.

Wie kann denn so ein Aufruf in seiner einfachsten Form aussehen:

```
writeln(GetGeoWeather('Klagenfurt', UrlWeatherReport25));
```

Bereits dieser Request liefert die wichtigsten Daten und Kennzahlen zur Weiterverarbeitung im JSON Format zurück. Somit hat man Zugriff auf freie Wetterdaten, Wettervorhersagen, sowie Wetterkarten mit Infos über Wolken, Sonnenaufgang, Windgeschwindigkeiten und Luftdruck zur Verfügung. Alle Wetterdaten lassen sich im JSON, XML oder HTML Format beziehen. Ein WSDL Endpunkt ist in Vorbereitung.

Die API ist effizient und flexibel genug, um weitere Parameter oder Sequenzen hinzuzufügen[1].

So lässt sich bereits ausführlich und Restful mit der URL interagieren:

[1] REST for Weather Forecasting

```
api.openweathermap.org/data/2.5/weather?q={city name}
```

Es sind Datenpunkte von rund 40,000 Wetterstatioen verfügbar und die Prognosen basieren auf bekannten Modellen. Nun lasst uns die Station programmieren. Als erstes hab ich die URL in eine Konstante gepackt:

```
Const
  UrlWeatherReport25=
                'http://api.openweathermap.org/data/2.5/weather?q=%s&units=metric';
```

Die Version ist bereits in der Schnittstelle erkennbar, in der einfachsten Form übergebe ich den Ort und eine Metrik, die mir Grad Celcius anstelle Kelvin liefert. Ansonsten müsste man einfach Kelvin –273.15 abziehen. Mit der REST-API Schnittstelle lassen sich natürlich externe Anwendungen mittels HTTP-Methoden (vor allem GET) in jeden Server integrieren. Damit kann ich auch Teilinformationen auslesen und Listen von z.B. Temperaturdaten, Objekt-Eigenschaften oder von Tags erstellen.

Weiter geht’s mit der GET Methode als Integration im Skript. Diese Methode dient als Funktionsmuster für viele Sprachen und besitzt als Argumente den Ort und eben den API-String. In der API muss dann nur noch der eigene Standort / ID verändert werden.
Die meisten Webmaster benötigen Wetterdaten in einem PHP oder maXbox Skript, dies hat mich dann zu diesem Tutorial bewegt. Der GET Methode wird ein Stream übergeben, die mir die JSON Daten zurückliefert:

```
function GetGeoWeather(const location: string;
                       const UrlRestAPI: string): string;
var IHTTP: TIdHTTP;
    IStream: TStringStream;
begin
  IHTTP:= TIdHTTP.Create(NIL);
  IStream:= TStringStream.Create('');
  try
    try
      IHTTP.Get1(Format(UrlRestAPI,[location]),IStream);
    except
      IHTTP.Get1(Format(UrlGeoLookupInfo2,[location]),IStream);
       //if something wrong try using a backup server.
    end;
    IStream.Seek(0,0);
    result:= 'GEO_Weather_Report: '+IStream.ReadString(IStream.Size);
  finally
    IHTTP.Free;
    IStream.Free;
  end;
end;
```

Das Wetter ist ja ein Phänomen, über das fast alle sprechen auch welche die nichts zu sagen haben. Dereinst ist es vielleicht möglich das Wetter zu schreiben, nicht nur als „Weather Report“ zu lesen. Für eine weitere App als Prognose der Wetterdaten hab ich einfach die API als Konstante ausgewechselt und noch die Sprache parametrisiert:

```
http://api.openweathermap.org/data/2.5/forecast/daily?
id=524901&lang={lang}
```

Ideologisch ist OpenWeatherMap inspiriert von OpenStreetMap und Wikipedia, die Informationen und Dienste allgemein und kostenlos zur Verfügung stellen.

Die Skripte zum Testen sind auch online verfügbar:

http://www.softwareschule.ch/examples/640_rest_weather_report.txt

http://www.softwareschule.ch/examples/640_rest_geocode.txt

Der Aufruf zeigt dann folgendes Bild als JSON Result:

```
writeln(GetGeoWeather('cologne', UrlWeatherReport25));

writeln(GetGeoWeather('bern', UrlWeatherReport25));
```

GEO_Weather_Report: {"coord":{"lon":7.45,"lat":46.95},"weather":
[{"id":803,"main":"Clouds","description":"broken clouds","icon":"04d"}],"base":"cmc
stations","main":
{"temp":19.27,"pressure":1014,"humidity":68,"temp_min":15.56,"temp_max":23.89},"wind":
{"speed":1.5,"deg":20.506},"clouds":{"all":75},"dt":1439805254,"sys":
{"type":1,"id":6013,"message":0.0048,"country":"CH","sunrise":1439785803,"sunset":143983
6653},"id":2661552,"name":"Bern","cod":200}

Interessant ist die Unixtime bei <sunrise> oder <sunset>, die ich konvertieren muss (gilt für UTC+1 Timestamp):

```
Writeln(DatetimeToStr(UnixToDateTime(1440045272)));
```

Dies gilt für alle Zeiten nach dem 01.01.1970 bis zum 19.01.2038. Was ist eigentlich ein Unix Timestamp? Der Unix-Timestamp gibt die Anzahl der Sekunden an, die seit dem 01.01.1970 vergangen sind. Zeitangaben in diesem Format sind z.B. bei der Codierung von PHP, REST-Diensten und MySQL-Datenbanken zu finden.
Zu den Ortsnamen ist noch anzufügen, dass man Eindeutigkeit mit einer ID erlangen kann, also Köln funktioniert als „Koeln“ und nicht als englisches „Cologne“. Als Testreferenz nutze ich immer die Koordinaten im longitude N und latitude E Format, also bei Köln:

GEO_Weather_Report: {"coord":{"lon":6.95,"lat":50.93},

Kiruna und Klagenfurt wird des öfteren als Referenz zu Europa eingesetzt. Dies bringt uns zum letzten Thema genannt Kartendarstellung. Wenn einmal die Koordinaten bekannt sind, die ja im JSON Result als <lon> und <lat> daherkommen, kann ich in einem weiteren Aufruf auch gleich die Karte einbringen. Das API sieht dann so aus:

Const

```
  UrlMapQuestAPIReverse= 'http://open.mapquestapi.com/nominatim/v1/reverse.php?
format=%s&json_callback=renderExampleThreeResults&lat=%s&lon=%s';
```

Man nennt das auch Geo Reverse Code, da ich aus den Koordinaten den Ort und die Karte zurück gewinne. Ich kombiniere dann die Karte indem ich die erhaltenen Koordinaten in die API `UrlMapQuestAPIReverse` füttere und das Searchresult in OpenStreetMap darstelle.

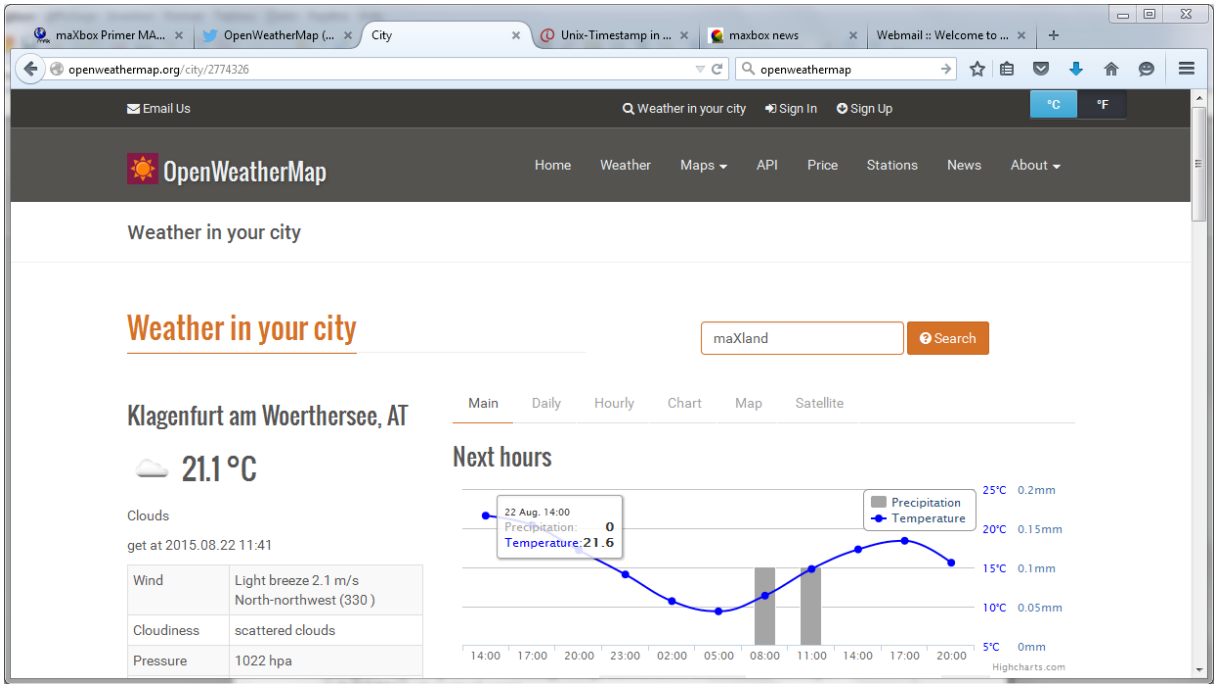
```
<?xml version="1.0" encoding="UTF-8"?>
-<searchresults more_url="http://mq-open-search-ext-
la02.ihost.aol.com:8000/nominatim/v1/search?
format=xml&exclude_place_ids=705429&q=krumpendorf+strandweg"
exclude_place_ids="705429" polygon="false" querystring="krumpendorf strandweg"
attribution="Data © OpenStreetMap contributors, ODbL 1.0.
http://www.openstreetmap.org/copyright" timestamp="Sun, 16 Aug 15 12:56:58 +0000">
        <place icon="http://mq-open-search-ext-
la02.ihost.aol.com:8000/nominatim/v1/images/mapicons/transport_train_station2.p.20.png"
importance="0.201" type="station" class="railway" display_name="Krumpendorf, Strandweg,
Gemeinde Krumpendorf am Wörthersee, Klagenfurt-Land, Region Klagenfurt-Villach, Kärnten,
9201, Österreich" lon="14.2218479" lat="46.6264738"temperature="18.62"
boundingbox="46.6264738,46.6264738,14.2218479,14.2218479" place_rank="30"
osm_id="247629819" osm_type="node" place_id="705429"/>
</searchresults>
```

Auch hier ist ein vereinfachter Direktaufruf möglich:

```
 writeln(GetMapXGeoReverse('XML','50.94134','6.95813'))
```

```
GetMapXGeoReverse2('XML',topPath,'50.94133705','6.95812611100766')
then
<?xml version="1.0" encoding="UTF-8"?>
-<reversegeocode
querystring="format=XML&json_callback=renderExampleThreeResults&lat=50.9413
3705&lon=6.95812611100766" attribution="Data © OpenStreetMap contributors,
ODbL 1.0. http://www.openstreetmap.org/copyright" timestamp="Mon, 22 Sep 14
13:36:43 +0000">
  <result lon="6.95812611100766" lat="50.94133705" ref="Kölner Dom"
    osm_id="4532022" osm_type="way" place_id="40406499">Kölner Dom, 4,
      Domkloster, Ursula-Viertel, Altstadt-Nord, Innenstadt, Köln,
      Regierungsbezirk Köln, Nordrhein-Westfalen, 50667,
      Deutschland</result>-
      <addressparts>
            <place_of_worship>Kölner Dom</place_of_worship>
            <house_number>4</house_number>
            <pedestrian>Domkloster</pedestrian>
            <neighbourhood>Ursula-Viertel</neighbourhood>
```

```
            <suburb>Altstadt-Nord</suburb>
            <city_district>Innenstadt</city_district>
            <county>Köln</county>
            <state_district>Regierungsbezirk Köln</state_district>
            <state>Nordrhein-Westfalen</state>
            <postcode>50667</postcode>
            <country>Deutschland</country>
            <country_code>de</country_code>
        </addressparts>
</reversegeocode>
```




```
OpenMapX('cathedral cologne');
OpenMapX('50.94134 6.95813'); //string!
```

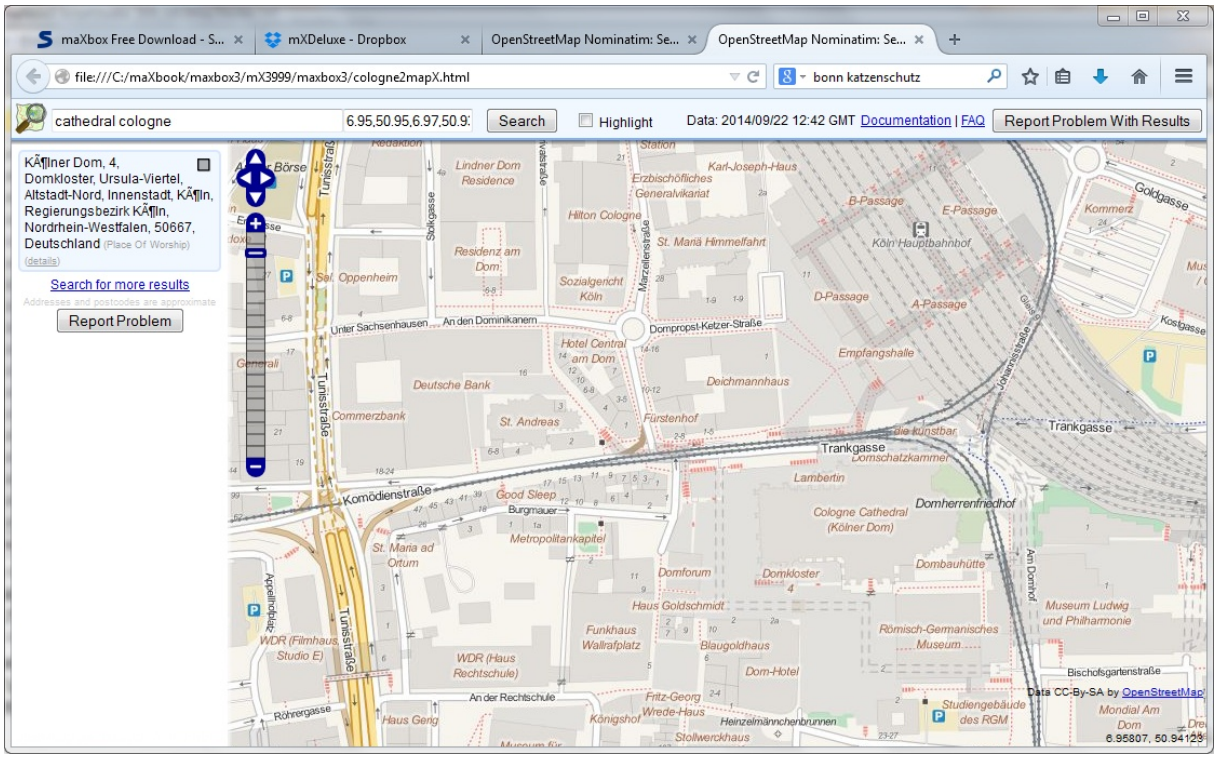
Mit einer GPS-Navigation lässt sich die Automatisierung nochmals vereinfachen. Die die UX, neudeutsch User Experience, deutlich erhöhen, da nun die Koordinaten der Wetterstation direkt zum API wandern und Karte wie Wetterdaten auf dem Schirm erscheinen; die Schnittstelle dazu findet man unter:

http://open.mapquestapi.com/nominatim/

http://www.softwareschule.ch/examples/509_GEOMap3.txt

```
procedure GetMapScript(C_form,apath: string; const Data:string);
var encURL: string;
    mapStream: TMemoryStream;
begin
  encURL:= Format(UrlMapQuestAPICode2, [c_form,HTTPEncode(Data)]);
  mapStream:= TMemoryStream.create;
  try
    HttpGet(EncURL, mapStream);    //WinInet
```

```
    mapStream.Position:= 0;
    mapStream.Savetofile(apath)
    OpenDoc(apath);
  finally
    mapStream.Free;
  end;
end;
```



Wenn nur ein Ort „gemappt“ wird, genügt die folgende API:

```
UrlMapQuestAPICode2='http://open.mapquestapi.com/nominatim/v1/search
.php?format=%s&json_callback=renderBasicSearchNarrative&q=%s';
```

Wie gesagt lassen sich alle Daten im JSON-Format anfordern inklusive die Wetterstationen mit den letzten Informationen, Städte mit dem aktuellen Wetter oder ein Lauf der Messungen von einzelnen Stationen.

Die folgende Note soll als Anregung für eine eigene Anwendung dienen. In eine Skript wird die zu erwartende Regenmenge von morgen ermittelt. So kann z.B. entschieden werden, ob es Sinn macht heute den Rasen und die Pflanzen zu bewässern. Besitzer von Solaranlagen oder Wärmepumpen können mit Hilfe einer Temperaturvorhersage dann auch ihr Energiemanagement optimieren und somit einen Teil des Internet of Things nutzen:

```
$content =
Sys_GetURLContent("http://api.openweathermap.org/data/2.5/fore
cast/daily?id=2850235&lang=de&mode=json");
$json = json_decode($content);
```

Feedback @
max@kleiner.com
Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

www.openweathermap.com

https://twitter.com/openweathermap

http://en.wikipedia.org/wiki/Reverse_geocoding
http://www.kleiner.ch/kleiner/gpsmax.htm

• Arduino Example.
http://www.softwareschule.ch/download/Arduino_C_2014_6_basta_box.pdf
http://sourceforge.net/projects/maxbox