maXbox Starter 99 – Data representation of gas in storage as a timeline AGSI dataset

starter expert + D11 + jupiter +

runs under Python3, Delphi, Jupyter-Notebook, Lazarus and maXbox4.

INTRODUCTION

This data science tutorial explains the so called **AGSI** data storage and his visualisation of the time line. **AGSI** is the **Aggregated Gas Storage Inventory** and offers you the possibility to be kept up to date whenever a new service announcement or update from one of our data providers is posted on the website. The **Gas Infrastructure Europe** (**GIE**) is also providing related data such as the **Storage Map** and **Storage Investment Database** at **https://www.gie.eu/publications/maps/**

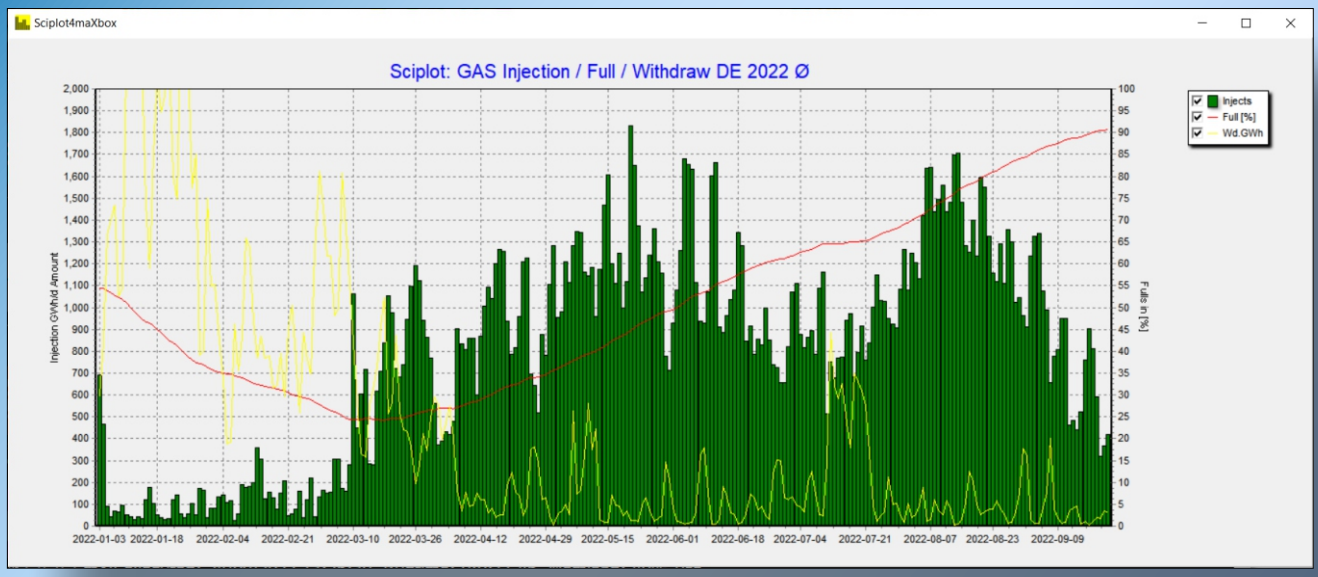

Figure 1: The result of the data is the chart.

We use **WinHttp.WinHttpRequest**, **JSONObjects** and **TEECharts** library with loading and testing the plot. Also an **API**-key is needed, get the key first at: **https://agsi.gie.eu/account**

The data represents gas in storage at the end of the previous gas day. Data is updated every day at 19:30 **CET** and a second time at 23:00. Before we dive into code this is the main part of the script:

```
plotform:= getForm2(1400, 600, clsilver, 'Sciplot4maXbox');
plotform.icon.loadfromresourcename(hinstance,'ZHISTOGRAM');

HttpResponse:=
    getEnergyStreamJSON2(URL_AGSIAPI2,'DE,2022-01-03,150',AGSI_APIKEY);
JSON2Plot(plotform, letGenerateJSON2(HttpResponse));
```

The main part generates a form, invokes the **API** and plots the data.
GIE is providing an **API** (Application Programming Interface) service on its **AGSI** and **ALSI** transparency publication platforms.
Using **API** access, consumer can bypass the **AGSI** and **ALSI** website and get hold of the data directly and continuously. It enables to extract, filter, aggregate the data and create any subset as required, without having to download each dataset separately from the website. The **API** export format is **JSON.** For example a subset of 150 days:

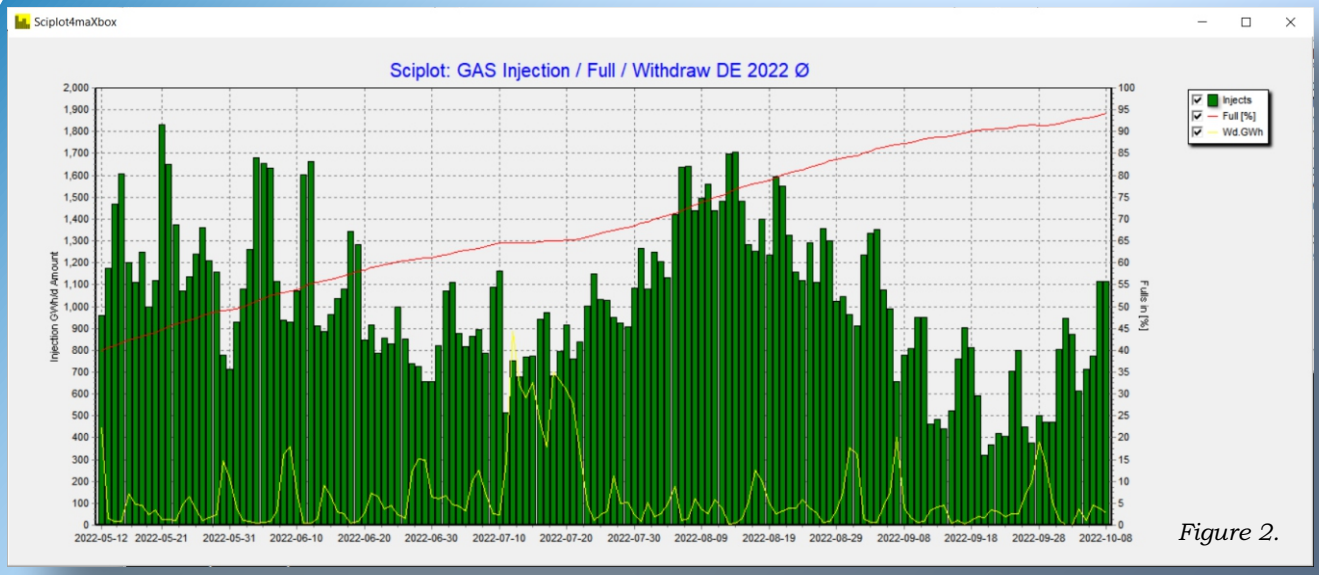

Figure 2.

The published datasets are based on the **EIC** code mapping table provided to **ACER**. Storage data is aggregated by company, and country.
With the call I pass country, start-date and amount of days:

```
_getEnergyStreamJSON2(URL_AGSIAPI2,'DE,2022-01-03,150',AGSI_APIKEY);
```

All available datasets can be downloaded also in **Excel, CSV** and **JSON** format. The data in this report shows an aggregated view – individual datasets by company and storage facility are also accessible.

So lets start with the **API** call:

```
_HttpResponse := getEnergyStreamJSON2(URL_AGSIAPI2,'DE,2022-01-03,150',AGSI_APIKEY);
```

This command and script runs **`WinHttp.WinHttpRequest`**. When you fail you get a bunch of exceptions like the following:

Exception: **`WinHttp.WinHttpRequest`**: The data necessary to complete this operation is not yet available; or you missed a valid key:

```
AGSIPost: Failed at getting response: 403{
 "error": {
   "code": 403,
   "message": "The request is missing a valid API key.",
   "status": "PERMISSION_DENIED"
 }}
```

The funny thing is the **JSON** formatted exception.
Be also carefully to expose your key as I get from **Git**: **GitGuardian** has detected the following **Google API Key** exposed within your **GitHub** account.
Next is the formatting of the get-call with a valid **API**-key request in the function **energyStream()**

```
function getEnergyStreamJSON2(AURL, feedstream, aApikey:string): string;
 ...
  encodURL:= Format(AURL,[HTTPEncode(asp[0]),(asp[1]),asp[2]]);
  writeln(encodurl) //debug
  hr:= httpRq.Open('GET', encodURL, false);
  httpRq.setRequestheader('user-agent',USERAGENTE);
  httpRq.setRequestheader('x-key',aAPIkey);
 ...
```

And where is the fabulous content-type? As far as I understood there are only two places in a web-request where to set a content-type:

❶ The client sets a content type for the body he is sending to the server (e.g. for get and post).
❷ The server sets a content type for the response.

A sender that generates a message containing a payload body has to generate a Content-Type header field in that message unless the intended media type of the enclosed representation is unknown to the sender; otherwise we failed at getting response: 503503 - Service Unavailable.

```
('headers={"Content-Type":"application/json"}')
 httpRq.setRequestheader('Content-Type',application/json);
```

It means that the content-type **HTTP** header should be set only for **PUT** and **POST** requests.
GET requests can have "Accept" headers, which say which types of content the client understands.
The server can then use that to decide which content type to send back.
As an option you can also use **TALWinInetHttpClient**.
It is a easy to use **WinInet-based** protocol and supports **HTTPs.**
HTTP client component which allows to post and get any data from the Web via **HTTP** protocol.

```
function TALHTTPClient_Get(aUrl: AnsiString; feedstream, aApikey: string): string;
Var LHttpClient: TALWininetHttpClient;  asp: TStringArray;
begin
  LHttpClient:= TALWininetHttpClient.create;
  asp:= splitStr(feedstream,',');
  LHttpClient.url:= Format(AURL,[HTTPEncode(asp[0]),(asp[1]),asp[2]]);
  LHttpClient.RequestMethod:= HTTPmt_Get; //HTTPrm_Post;
  LHttpClient.RequestHeader.UserAgent:=USERAGENTE;
  //LHttpClient.RequestHeader.CustomHeaders:=
  LHttpClient.RequestHeader.RawHeaderText:='x-key:'+aAPIkey;
  try
    result:= LHttpClient.Get1(LHttpClient.url); //overload;
  finally
    LHttpClient.Free;
  end;
end;
```

Any missing or incomplete data is also be visible on **AGSI.**
Next we step to the conversion of our **JSON** response for the plot with `TJSONObject`:

```
function letGenerateJSON2(HttpRqresponseText: string): TJSONArray;
var jo: TJSONObject;
begin
   jo:= TJSONObject.Create4(HttpRqresponseText);
   try
      //writeln(jo.getstring('data'));
      writeln(itoa(jo.getjsonarray('data').getjsonobject(0).length))
      writeln(itoa(jo.getjsonarray('data').length))
      result:= jo.getjsonarray('data');
      //write out to check
   for it:= 0 to result.length-1 do
      writeln(result.getjsonobject(it).getstring('gasDayStart')+':'+
            result.getjsonobject(it).getstring('injection'));
   except
      writeln('EJson: '+ExceptiontoString(exceptiontype, exceptionparam));
   end;
end;
```

And this JSON Array as above function returns, we pass to the next plot:

```
procedure JSON2Plot(form1: TForm; jar: TJSONArray);
var chart1: TChart; cnt: integer; sumup,tmp2,tmp: double; gday: string;
begin
  form1.onclose:= @Form_CloseClick;
  chart1:= ChartInjector(form1);
  sumup:=0; tmp2:=0; tmp:=0;
  try
    for cnt:= 0 to jar.length-1 do
      begin
        //writeln(locate.getjsonobject(it).getstring('gasDayStart')+':'+
        tmp:= jar.getjsonobject(jar.length-1-cnt).getdouble('injection');
        tmp2:= jar.getjsonobject(jar.length-1-cnt).getdouble('full');
        sumup:= sumup+tmp;
        gday:= jar.getjsonobject(jar.length-1-cnt).getstring('gasDayStart');
        chart1.Series[0].Addxy(cnt,tmp,gday,clgreen);
        chart1.Series[1].Addxy(cnt,tmp2,'',clred);
        chart1.Series[2].Addxy(cnt,jar.getjsonobject(jar.length-1-cnt).getdouble('withdrawal'),'',clyellow);
      end;
  except
    writeln('EPlot: '+ExceptiontoString(exceptiontype, exceptionparam));
  end;
  PrintF('Landrange %d: Injection sum: %.2f',[jar.length-1,sumup]);
end;
```

As we can see we have 4 series to plot (including timeline):

❶	**Injection**	(Injection during gas day)
❷	**Full**	(Storage / WGV (in%))
❸	**Withdrawal**	(Withdrawal during gas day (2 digits accuracy)).
❹	**GasDayStart**	(The start of the gas day reported)

The time series is a result of the gas day and a trend is available.
"Gas day" means the period from 5:00 to 5:00 **UTC** the following day for winter time and from 4:00 to 4:00 **UTC** the following day when daylight saving is applied. Gas day is to be interpreted as `UTC+1` for **CET** or `UTC+2` in summer time for **CEST.** (Definition: see **CAM Network Code** specific-cations).
API access is provided in a **REST**-like interface (**Representational State Transfer**) exposing database resources in a **JSON** format with content-type in the mentioned `Response Header`.

maXbox Starter 99 – Data representation of gas in storage as a timeline AGSI dataset

Response Headers

X-Firefox-Http3	h3
alt-svc	h3=":443"; ma=86400, h3-29=":443"; ma=86400
cache-control	no-cache, private
cf-cache-status	DYNAMIC
cf-ray	7580b85f6dae0200-ZRH
content-encoding	br
content-type	application/json
date	Mon, 10 Oct 2022 16:26:53 GMT
server	cloudflare
x-robots-tag	noindex

Request Headers

Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding	gzip, deflate, br
Accept-Language	en-GB,en;q=0.5
Alt-Used	agsi.gie.eu
Connection	keep-alive
Host	agsi.gie.eu
Sec-Fetch-Dest	document
Sec-Fetch-Mode	navigate
Sec-Fetch-Site	none
Sec-Fetch-User	?1
Sec-GPC	1
TE	trailers
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0

Figure 3.

The code of the data science vision contains example usage, and runs under **Python3, Delphi, Jupyter-Notebook, Lazarus** and `maXbox4`. **NOTE** that the **API** service is made available to the public free of charge. Only the data as currently available on the platforms is made available.
Tip: To extract data direct from the system, you can click on one of these links in a browser (web traffic versus **API** traffic): AGSI+ `https://agsi.gie.eu/api?type=eu`

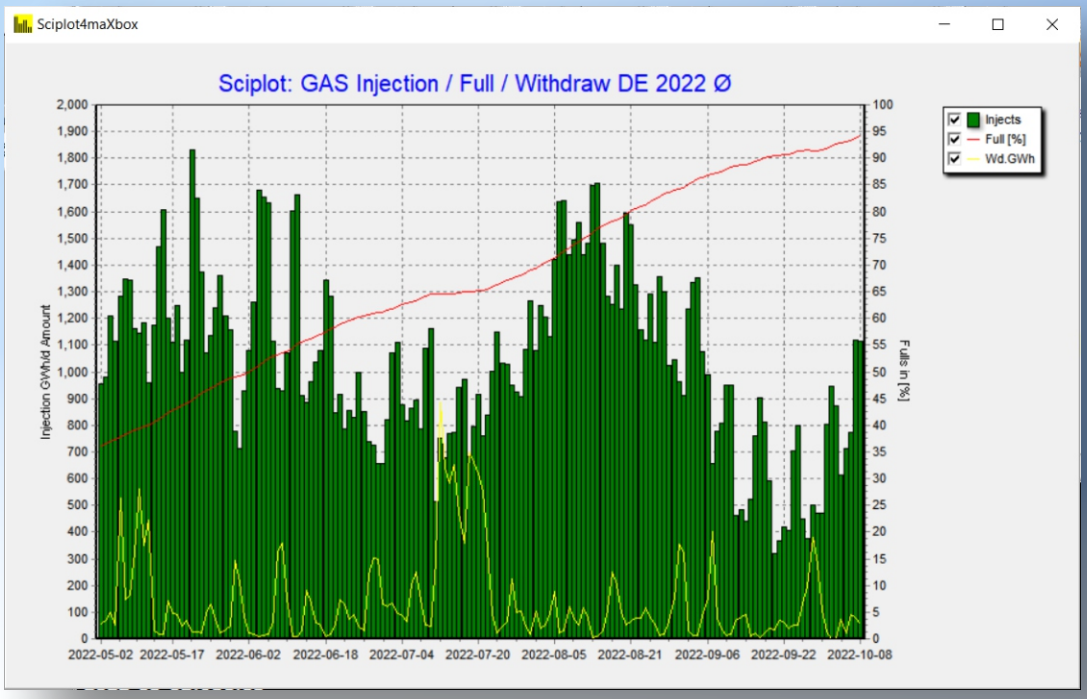

Figure 4.

AGSI DATA STORAGE

The scripts and images can be found: https://github.com/maxkleiner/agsi-data

REFERENCE:

https://agsi.gie.eu/api

https://www.gie.eu/transparency-platform/GIE_API_documentation_v006.pdf

https://svn.code.sf.net/p/alcinoe/code/demos/ALWinInetHTTPClient/_source/Unit1.pas

https://docwiki.embarcadero.com/Libraries/Sydney/en/System.Net.HttpClient.THTTPClient.Post

Doc and Tool: https://maxbox4.wordpress.com

Script Ref: 1154_energy_api_agsi_plot14.txt

APPENDIX: shows an WinAPIDownload class from maXbox4 integration

```
{*-------------------------------------------------------------------------*)
TWinApiDownload = class(TObject)
private
  fEventWorkStart: TEventWorkStart;
  fEventWork: TEventWork;
  fEventWorkEnd: TEventWorkEnd;
  fEventError: TEventError;
  fURL: string;
  fUserAgent: string;
  fStop: Boolean;
  fActive: Boolean;
  fCachingEnabled: Boolean;
  fProgressUpdateInterval: Cardinal;
  function GetIsActive: Boolean;
public
  constructor Create;
  destructor Destroy; override;
  function CheckURL(aURL: string): Integer;
  function Download(Stream: TStream): Integer; overload;
  function Download(var res: string): Integer; overload;
  function ErrorCodeToMessageString(aErrorCode: Integer): string;
  procedure Stop;
  procedure Clear;
  property UserAgent: string read fUserAgent write fUserAgent;
  property URL: string read fURL write fURL;
  property DownloadActive:Boolean read GetIsActive;
  property CachingEnabled:Boolean read fCachingEnabled write fCachingEnabled;
  property UpdateInterval:Cardinal read fProgressUpdateInterval
                   write fProgressUpdateInterval;
  property OnWorkStart: TEventWorkStart read fEventWorkStart
                     write fEventWorkStart;
  property OnWork: TEventWork read fEventWork write fEventWork;
  property OnWorkEnd: TEventWorkEnd read fEventWorkEnd write fEventWorkEnd;
  property OnError: TEventError read fEventError write fEventError;
end;
```