

# AGSI DATENSPEICHERUNG ~~maXbox~~ SEITE 1/9

maXbox Starter 99 – Datendarstellung von Gas in Speichern als Zeitleiste AGSI-Datensatz



läuft unter Python3, Delphi, Jupyter-Notebook, Lazarus und maXbox4.

## EINLEITUNG

Dieses Data-Science-Tutorial erklärt die so genannte **AGSI-Datenspeicherung** und seine Visualisierung der Zeitleiste. **AGSI** steht für **Aggregated Gas Storage Inventory** und bietet Ihnen die Möglichkeit, immer auf dem Laufenden zu bleiben, wenn eine neue Serviceankündigung oder ein Update von einem unserer Datenanbieter auf der Website veröffentlicht wird. Die **Gas Infrastructure Europe (GIE)** stellt auch verwandte Daten wie die Storage Map und die Storage Investment Database unter <https://www.gie.eu/publications/maps/> zur Verfügung.

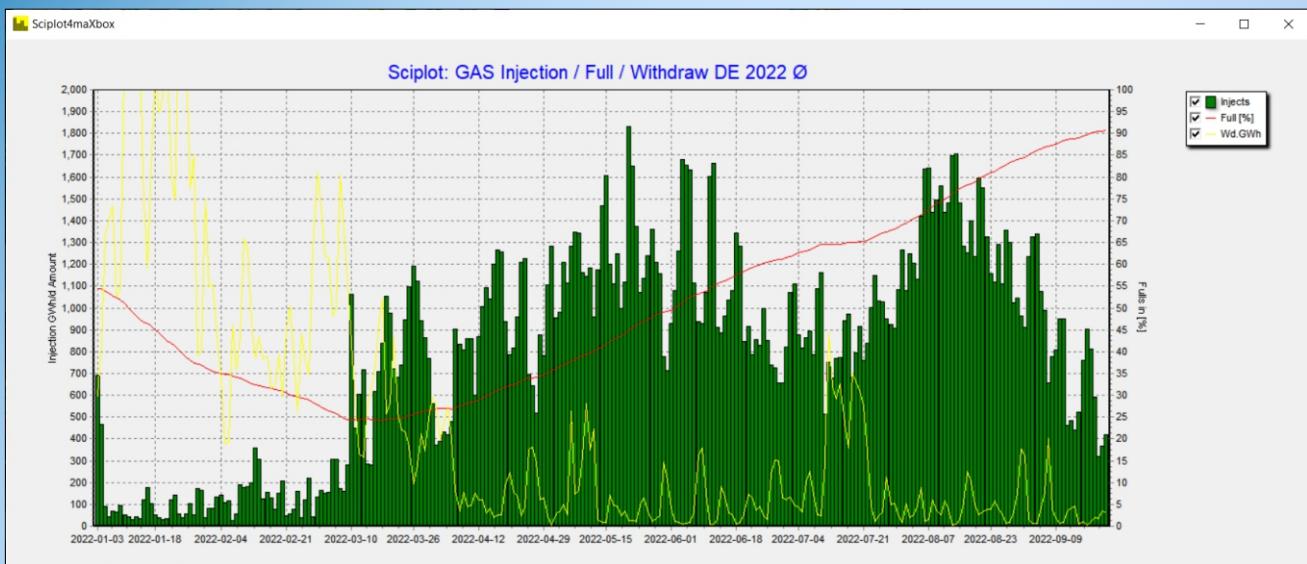


Abbildung 1: Das Ergebnis der Daten ist das Diagramm.

Wir verwenden `WinHttp.WinHttpRequest`, `JSONObjects` und die **TEECharts-Bibliothek** beim Laden und Testen des Plots. Außerdem wird ein **API-Schlüssel** benötigt, den Sie zunächst unter <https://agsi.gie.eu/account> finden.

Daten werden jeden Tag um 19:30 Uhr MEZ und ein zweites Mal um 23:00 Uhr aktualisiert. Bevor wir in den Code eintauchen, hier der Hauptteil des Skripts:

```
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));
```

# AGSI DATENSPEICHERUNG ~~maxBox~~ SEITE 2/6

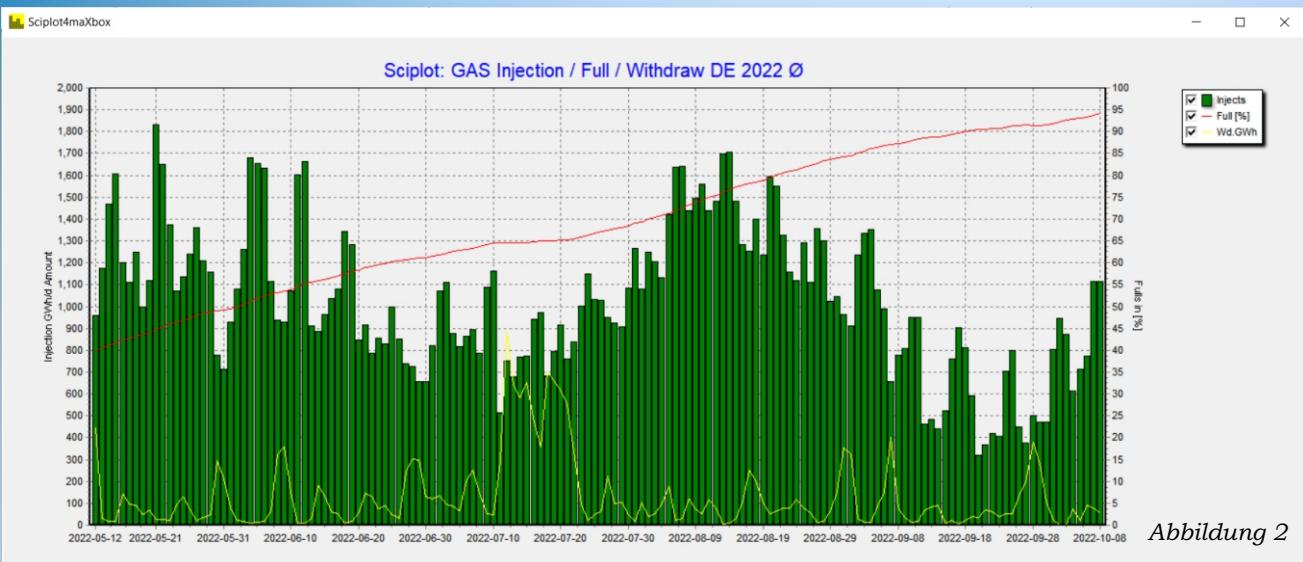
maXbox Starter 99 – Datendarstellung von Gas in Speichern als Zeitleiste AGSI-Datensatz

starter

Der Hauptteil erzeugt ein Formular, ruft die **API** auf und zeichnet die Daten auf. **GIE** bietet einen **API-Dienst** (Application Programming Interface) auf seiner **AGSI** und **ALSI** Plattformen zur Veröffentlichung von Transparenz.

Über den API-Zugang können die Verbraucher die **AGSI**- und **ALSI-Website** umgehen und direkt und kontinuierlich auf die Daten zugreifen. Er ermöglicht das Extrahieren, Filtern und Aggregieren der Daten und erstellen Sie bei Bedarf beliebige Teilmengen, ohne jeden Datensatz einzeln von der Website herunterladen zu müssen. Das **API-Exportformat** ist **JSON**.

Zum Beispiel eine Teilmenge von 150 Tagen:



Die veröffentlichten Datensätze basieren auf der **EIC-Code-Zuordnungstabelle**, die **ACER** zur Verfügung gestellt wurde. Die Speicherdaten sind nach Unternehmen und Land aggregiert. Mit dem Anruf übergebe ich Land, Startdatum und Anzahl der Tage:

```
_getEnergyStreamJSON2(URL_AGSIPI2,'DE,2022-01-03,150',AGSI_APIKEY);
```

Alle verfügbaren Datensätze können auch im **Excel**-, **CSV**- und **JSON**-Format heruntergeladen werden. Die Daten in diesem Bericht zeigen eine aggregierte Ansicht - einzelne Datensätze nach Unternehmen und Lagerstätte sind ebenfalls zugänglich.

Beginnen wir also mit dem **API-Aufruf**:

```
_HttpResponse _= getEnergyStreamJSON2(URL_AGSIPI2,'DE,2022-01-03,150',AGSI_APIKEY);
```

Dieser Befehl und dieses Skript führen **WinHttp.WinHttpRequest** aus. Wenn Sie scheitern, erhalten Sie eine Reihe von Ausnahmen wie die folgende:

**EXCEPTION: WinHttp.WinHttpRequest**: Die für diesen Vorgang erforderlichen Daten sind noch nicht verfügbar, oder Sie haben einen gültigen Schlüssel vergessen:



# AGSI DATENSPEICHERUNG ~~maxBox~~ SEITE 3/6

maxBox Starter 99 – Datendarstellung von Gas in Speichern als Zeitleiste AGSI-Datensatz

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

Das Lustige ist die **JSON**-formatierte Ausnahme.

Seien Sie auch vorsichtig bei der Offenlegung Ihres Schlüssels, den ich von **Git** erhalte:

GitGuardian hat den folgenden Google API-Schlüssel in Ihrem GitHub-Konto entdeckt.

Als nächstes folgt die Formatierung des get-Aufrufs mit einer gültigen **API-Schlüsselanforderung** in der Funktion **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);  
  ...
```

Und wo ist der fabelhafte Content-Type? Soweit ich verstanden habe, gibt es nur zwei Stellen in einer Web-Anfrage, an denen ein Content-Type festgelegt werden kann:

- ① Der Client legt einen Inhaltstyp für den Body fest, den er an den Server sendet (z.B. für get und post ist das Senden an den Server (z.B. für **get** und **post**).
- ② Der Server legt einen Inhaltstyp für die Antwort fest.

Ein Absender, der eine Nachricht generiert, die einen Nutzdatenkörper enthält, muss ein Content-Type-Header-Feld in dieser Nachricht generieren, es sei denn, der beabsichtigte Medientyp der enthaltenen Darstellung ist dem Absender nicht bekannt; andernfalls konnten wir keine Antwort erhalten: 503503 - Dienst nicht verfügbar.

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

Es bedeutet, dass der HTTP-Header content-type nur für **PUT** gesetzt werden sollte und **POST** Anfragen. GET-Anfragen können "Accept"-Kopfzeilen haben, die angeben, welche Arten von Inhalten der Client versteht. Der Server kann dann anhand dieser Angaben entscheiden, welchen Inhaltstyp er zurücksenden soll. Als Option können Sie auch **TALWinInetHttpClient** verwenden.

Es ist ein einfach zu bedienendes WinInet-basiertes Protokoll und unterstützt HTTPS.

HTTP-Client-Komponente, mit der Sie beliebige Daten aus dem Web über das HTTP-Protokoll posten und abrufen können.

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



# AGSI DATENSPEICHERUNG ~~maxBox~~ SEITE 4/6

maXbox Starter 99 – Datendarstellung von Gas in Speichern als Zeitleiste AGSI-Datensatz

Alle fehlenden oder unvollständigen Daten werden auch auf **AGSI** angezeigt.

Als nächstes gehen wir zur Konvertierung unserer **JSON**-Antwort für die Darstellung mit **TJSONObject** über:

```
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;
```

Und dieses **JSON-Array**, das die obige Funktion zurückgibt, übergeben wir an den nächsten 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,cgreen);
        chart1.Series[1].Addxy(cnt,tmp2,"clred");
        chart1.Series[2].Addxy(cnt,jar.getJSONObject(jar.length-1-cnt).getdouble('withdrawal'),"cleyellow");
      end;
  except
    writeln('EPlot: '+ExceptionToString(exceptiontype, exceptionparam));
  end;
  PrintF('Landrange %d: Injection sum: %.2f',[jar.length-1,sumup]);
end;
```

Wie Sie sehen, haben wir 4 Serien zu zeichnen (einschließlich Zeitleiste):

- ① **Injection** (Injektion während des Gastages)
- ② **Vollständig** (Speicher / WGV (in%))
- ③ **Entnahme** (Entnahme während des Gastages (2-stellige Genauigkeit)).
- ④ **GasDayStart** (Der Beginn des gemeldeten Gastages)

Die Zeitreihe ist ein Ergebnis des Gastages und ein Trend ist verfügbar.

"Gastag" bezeichnet den Zeitraum von 5:00 Uhr bis 5:00 Uhr UTC des folgenden Tages bei Winterzeit und von 4:00 Uhr bis 4:00 Uhr UTC des folgenden Tages, wenn Sommerzeit gilt.

Der Gastag ist als UTC+1 für die MEZ oder UTC+2 in der Sommerzeit für die MESZ zu interpretieren (Definition: siehe CAM Network Code specific-cations).

Der API-Zugriff erfolgt über eine **REST**-ähnliche Schnittstelle (Representational State Transfer), die Datenbankressourcen in einem **JSON-Format** mit Content-Type im erwähnten Response Header bereitstellt.



# AGSI DATENSPEICHERUNG ~~maxBox~~ SEITE 5/6

maxBox Starter 99 – Datendarstellung von Gas in Speichern als Zeitleiste AGSI-Datensatz

## Response Headers

```
X-Firefox-User: 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
```

Abbildung 3.

Der Code der **Data Science Vision** enthält Anwendungsbeispiele und läuft unter **Python3**, **Delphi**, **Jupyter- Notebook**, **Lazarus** und **maXbox4**. **HINWEIS:** Der API-Service wird der Öffentlichkeit kostenlos zur Verfügung gestellt. Es werden nur die Daten zur Verfügung gestellt, die derzeit auf den Plattformen verfügbar sind.

Tipp: Um Daten direkt aus dem System zu extrahieren, können Sie in einem Browser auf einen der folgenden Links in einem browser klicken (Web Verkehr gegenüber API-Verkehr):  
 AGSI+ <https://agsi.gie.eu/api?type=eu>

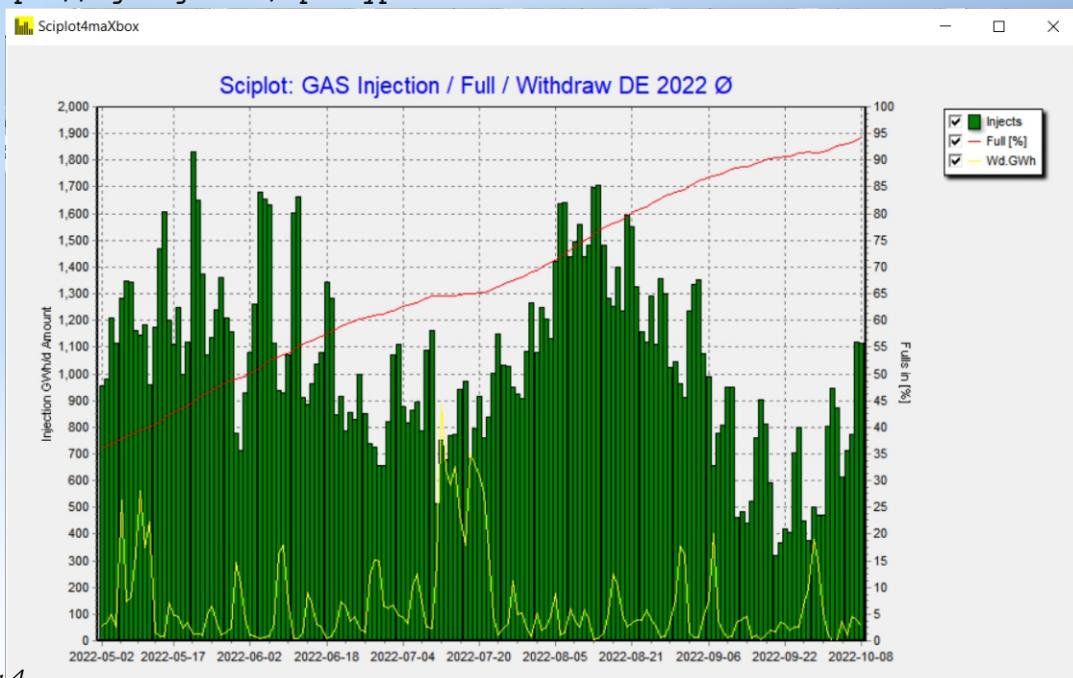


Abbildung 4.



# AGSI DATENSPEICHERUNG ~~maxBox~~ SEITE 6/6

Die Skripte

und Bilder finden Sie unter: <https://github.com/maxkleiner/agsi-data>

## REFERENZEN:

<https://agsi.gie.eu/api>  
[https://www.gie.eu/transparency-platform/GIE\\_API\\_documentation\\_v006.pdf](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://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 und Tool: <https://maxbox4.wordpress.com>

Script Ref: 1154\_energy\_api\_agsi\_plot14.txt

APPENDIX: zeigt eine WinAPIDownload-Klasse aus der 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;
```

