24.2.4.1 HTTPClient – Projekte

In den folgenden Kapiteln werden Ihnen zwei HTTP-Client-Projekte ausführlich vorgestellt.

Mit Hilfe der Klasse HTTPClient können Webservices genutzt werden, wobei Anfragen an einen HTTP-Server gesendet und dessen Antwort empfangen werden. Für dieses Einsatzgebiet finden Sie in der Gambas-Dokumentation und in der Software-Farm geeignete Projekte.

Eine gute Beschreibung für einen Webservice finden Sie auf www.bitfactory.io/de/blog: *Webservices ermöglichen es, dass Maschinen oder Anwendungen miteinander kommunizieren können (M2M-Kommunikation). Sie bilden die Schnittstelle zwischen Client und Server und ermöglichen den automatischen Datenaustausch zwischen diesen Softwaresystemen. Ihren Dienst stellen sie über das Internet zur Verfügung, wo Softwareanwendungen sie ansprechen können.*

Ein Webservice als Dienst ist über einen eindeutigen Uniform Resource Identifier (URI) ansprechbar – ähnlich der URL einer Website. Als Protokoll wird vorwiegend HTTPS (Hypertext Transfer Protocol Secure) eingesetzt, um den Datenaustausch abzusichern. Der Zugriff auf eine Ressource erfolgt mit den Methoden Get, Post und Put. Als Daten-Format werden häufig das XML- und das JSON-Format verwendet.

24.2.4.1.1 Webservice

Um unterschiedliche Webservices erfolgreich zu nutzen, könnte die folgende Liste mit Stichpunkten hilfreich sein:

- Daten-Quelle ermitteln → URI
- Daten-Anforderung definieren → Syntax der Anfrage (Request-Query)
- Daten-Format erkunden
- Daten anfordern (GetData)
- Daten speichern (StoreData) → Option:File
- Daten auswählen (SelectData)
- Daten aufbereiten (EditData) → Formatierung
- Daten anzeigen (ShowData) → Option Virtualisierung

Die beiden Projekte werden den einzelnen Punkten dieser Liste folgend umgesetzt.

24.2.4.1.2 Projekt 1 – Wetterdaten

Im ersten Projekt wird der Webservice 'Wetterdaten' von https://openweathermap.org eingesetzt. Unter der Rubrik 'Pricing' finden Sie auch das kostenlose Angebot. Um das zu nutzen, müssen Sie auf der Webseite mit 'Get API key' ein Konto anlegen. Im Gegenzug erhalten Sie einen API-Key, mit dem Sie aktuelle Wetterdaten nach den angegebenen Bedingungen vom Server anfordern können. Die Syntax der Anfrage (Query-String) wird unter https://openweathermap.org/current gut beschrieben.

Für den Ort 'Osterburg' kann nach den o.a. Informationen diese API-Anfrage so eingesetzt werden:

```
https://api.openweathermap.org/data/2.5/weather?q=Osterburg,de&lang=de&units=metric&APPID=apikey
```

- Ort: Osterburg, Land: Deutschland
- Textausgaben in deutscher Sprache
- Metrische Einheiten: m/s, mm, °C, hPa, UTC; relative Angaben in %

und es gilt für die Get()-Methode:

- Ein Query-String nach dem URL wird mit einem ?-Zeichen eingeleitet.
- Jeder Parameter besteht aus einem Name-Wert-Paar ("Name=Wert") und
- mehrere Parameter werden mit einem &-Zeichen verbunden.

In einer Konsole erhält man zum Beispiel als Antwort auf die o.a. Anfrage diese Wetterdaten:

```
$ curl 'https://api.openweathermap.org/data/2.5/weather?q=Osterburg,de&lang=de&units=metric&APPID=APIKEY'
{"coord":{"lon":11.7667,"lat":52.7833},"weather":[{"id":804,"main":"Clouds","description":"Bedeckt",
"icon":"04d"}],"base":"stations","main":{"temp":6.74,"feels_like":3.71,"temp_min":6.72,"temp_max":8.45,
```

```
"pressure":994,"humidity":70,"sea_level":994,"grnd_level":992},"visibility":10000,"wind":{"speed":4.57,
"deg":264,"gust":8.52},"clouds":{"all":100},"dt":1649438025,"sys":{"type":2,"id":2000066,"country":"DE",
sunrise":1649392251,"sunset":1649440704},"timezone":7200,"id":2856639,"name":"Osterburg","cod":200}
```


Bei den Projekten kann für eine Daten-Sichtung auch die Anwendung eines Web-Browsers empfohlen werden. Dafür braucht man dann weder eine Kommandozeile, noch ein Gambas-Programm mit einem Python-Tool und das Ausgabe-Format ist frei wählbar:

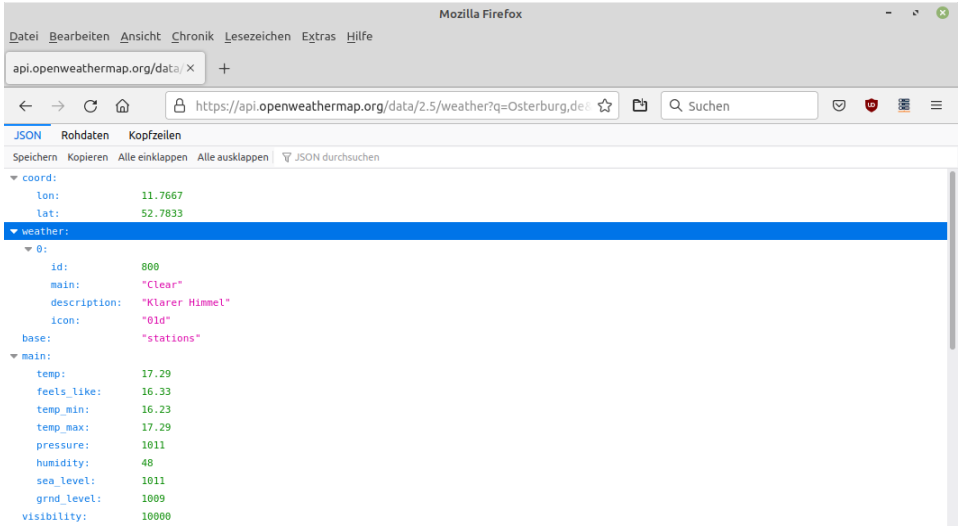

Abbildung 24.2.4.1.1: Anzeige ausgewählter (formatierter) Wetterdaten in einem Web-Browser

Kommentar:

Es fällt in beiden Anzeigen sofort auf, dass die Wetterdaten im JSON-Format – dem *Standardformat* für Wetterdaten von openweathermap.org – ausgeliefert werden. Wenn man diese in geeigneter Weise formatiert, so kann man besser erkennen, welche Daten in der Antwort enthalten sind und festlegen, welche Daten ausgewählt und bearbeitet werden sollen:


```
{
  "coord": {
    "lon": 11,7667,
    "lat": 52,7833
  },
  "weather": [
    {
      "id": 804,
      "main": "Clouds",
      "description": "Bedeckt",
      "icon": "04d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 6,74,
    "feels_like": 3,71,
    "temp_min": 6,72,
    "temp_max": 8,45,
    "pressure": 994,
    "humidity": 70,
    "sea_level": 994,
    "grnd_level": 992
  },
  "visibility": 10000,
  "wind": {
    "speed": 4,57,
    "deg": 264,
    "gust": 8,52
  },
  "clouds": {
    "all": 100
  },
  "dt": 1649438025,
  "sys": {
    "type": 2,
    "id": 2000066,
    "country": "DE",
```

```
    "sunrise": 1649392251,
    "sunset": 1649440704
  },
  "timezone": 7200,
  "id": 2856639,
  "name": "Osterburg",
  "cod": 200
}
```

Im Projekt werden die rot markierten (Roh-)Daten ausgewählt und für die Anzeige weiter verarbeitet. Es wird der komplette Quelltext angegeben und anschließend kommentiert:

```
[1]  ' Gambas class file
[2]
[3]  Public iTimeStampGMT As Integer
[4]  Public sJSONData As String
[5]  Public sAPIKey As String
[6]  Public sCity As String
[7]  Public hHTTPClientData As HttpClient
[8]  Public hHTTPClientIcon As HttpClient
[9]  Public hJSONCollection As JSONCollection
[10] Public cWind As Collection
[11] Public cClouds As Collection
[12] Public cMain As Collection
[13] Public cSys As Collection
[14] Public avWeather As Variant[]
[15]
[16] '-- Const APIKEY As String = "YOUR_API_KEY"
[17] '-- Const CITY As String = "YOUR_CITY"
[18] '-- Const COUNTRY As String = "YOUR_COUNTRY"
[19]
[20] Public Sub Form_Open()
[21]
[22]     Dim sMessage, sJSONFormatted As String
[23]
[24]     FMain.Title = ("Current weather data")
[25]     MAdditional.CheckNetwork()
[26]
[27]     sJSONData = GetWeatherData(COUNTRY, CITY, APIKEY)
[28] '-- Print "JSON_DATA –»  "; sJSONData
[29]
[30] '-- MAdditional.IsInstalled("python3")
[31] '-- Shell "echo '" & sJSONData & "' | python3 -m json.tool" To sJSONFormatted
[32] '-- Print sJSONFormatted
[33]
[34]     hJSONCollection = JSON.Decode(sJSONData, True)
[35]     If hJSONCollection["cod"] = 401 Then
[36]        sMessage = "Der API-Key ist leer oder nicht aktiviert oder fehlerhaft!<hr>"
[37]        sMessage &= "Sie können einen (kostenlosen) API-Key von<br>"
[38]        sMessage &= "<b>https://home.openweathermap.org/users/sign_up</b>"
[39]        sMessage &= "<br>anfordern, um die Wetter-Daten auslesen zu können!"
[40]        Message.Error(sMessage)
[41]        FMain.Close()
[42]     Else If hJSONCollection["cod"] = 404 Then
[43]        sMessage = "Die angegebene Stadt '" & CITY & "' wurde nicht gefunden!<hr>"
[44]        sMessage &= "‣ Der Name der Stadt ist falsch oder<br>"
[45]        sMessage &= "‣ die Stadt-ID ist falsch oder<br>"
[46]        sMessage &= "‣ das Format der API-Key-Anfrage ist fehlerhaft."
[47]        Message.Error(sMessage)
[48]        FMain.Close()
[49]     Else
[50]        EditAndShowWeatherData()
[51]     Endif
[52]
[53] End
[54]
[55] Public Function GetWeatherData(argCountry As String, argCity As String, argAPIKey As String) As String
[56]
[57]     Dim sURL, sQuery, sURI, sRawData As String
[58]
[59]     sURL = "https://api.openweathermap.org/data/2.5/weather"
[60]     sQuery = "?"
[61]     sQuery &= "q=" & argCity & "," & argCountry
[62]     sQuery &= "&" & "lang=" & argCountry
[63]     sQuery &= "&" & "units=metric"
[64]     sQuery &= "&" & "APPID=" & argAPIKey
[65]     sURI = sURL & sQuery
[66] '-- Print sURI
[67]
[68]     hHTTPClientData = New HttpClient As "HTTPClientData"
[69]     hHTTPClientData.URL = sURI
[70]     hHTTPClientData.Async = False
```

```
[71]     hHTTPClientData.Timeout = 10
[72]
[73]     hHTTPClientData.Get()
[74]
[75]     If Lof(hHTTPClientData) Then sRawData = Read #hHTTPClientData, Lof(hHTTPClientData)
[76]     Return sRawData
[77]
[78] End
[79]
[80] Public Sub EditAndShowWeatherData()
[81]
[82]     Dim iLocalTimeStamp As Integer
[83]     Dim sCityName, sDate, sIconID, sDay, sDateDMY, sTime As String
[84]     Dim dDate As Date
[85]
[86]     sCityName = hJSONCollection["name"]
[87]
[88]     iLocalTimeStamp = hJSONCollection["dt"]
[89]     dDate = TimeStampToDate(iLocalTimeStamp)
[90]     sDay = Format$(dDate, "dddd")
[91]     sDateDMY = Format$(dDate, "d. mmmm yyyy")
[92]     sTime = Format$(dDate, "hh:nn") & " Uhr"
[93]     sDate = sDay & ", " & sDateDMY & " - " & sTime
[94]
[95]     FMain.Text = ("Current weather data") & (" for ") & sCityName & ": " & sDate
[96] '-- ----------------------------------------------------------------------------
[97]
[98]     cWind = hJSONCollection["wind"]
[99]     txbWindSpeed.Text = "   " & Round(cWind["speed"], -1)
[100]      txbWindDirection.Text = "   " & Round(cWind["deg"], 0)
[101]  '-- ----------------------------------------------------------------------------
[102]
[103]      txbTemperature.Text = "   " & Round(hJSONCollection["main"]["temp"], -1)
[104]      txbPressure.Text = "   " & hJSONCollection["main"]["pressure"]
[105]      txbHumidity.Text = "   " & hJSONCollection["main"]["humidity"]
[106]  '-- ----------------------------------------------------------------------------
[107]
[108]      cClouds = hJSONCollection["clouds"]
[109]      txbCloudiness.Text = "   " & cClouds["all"]
[110]  '-- ----------------------------------------------------------------------------
[111]
[112]      cSys = hJSONCollection["sys"]
[113]      txbSunRise.Text = "   " & Format(TimeStampToDate(cSys["sunrise"]), "hh:nn")
[114]      txbSunSet.Text = "   " & Format(TimeStampToDate(cSys["sunset"]), "hh:nn")
[115]  '-- ----------------------------------------------------------------------------
[116]      avWeather = hJSONCollection["weather"]
[117]      lblDescription.Text = "   " & avWeather[0]["description"]
[118]      sIconID = avWeather[0]["icon"]
[119]  '-- Print sIconID
[120]
[121]      GetAndStoreWeatherIcon(sIconID)
[122]
[123]      pboxWeatherIcon.Picture = Picture[Application.Path &/ "WeatherIcon/weathericon.png"]
[124]
[125]  End
[126]
[127]  Public Sub GetAndStoreWeatherIcon(argIconID As String)
[128]
[129]      Dim sURIIcon As String
[130]
[131]      sURIIcon = "https://openweathermap.org/img/w/" & argIconID & ".png"
[132]  '-- sURIIcon = "https://openweathermap.org/img/wn/" & argIconID & "@2x.png" '-- Alternative
[133]
[134]      hHTTPClientIcon = New HttpClient As "HTTPClientIcon"
[135]      hHTTPClientIcon.URL = sURIIcon
[136]      hHTTPClientIcon.Async = False
[137]      hHTTPClientIcon.Timeout = 10
[138]
[139]      hHTTPClientIcon.Get([], Application.Path &/ "WeatherIcon/weathericon.png")
[140]
[141]  End
[142]
[143]  Public Sub HTTPClientData_Error()
[144]      Message.Error("<b><font color='DarkRed'>DATA: ERROR</b></font><hr>" & hHTTPClientData.ErrorText)
[145]  End
[146]
[147]  Public Sub HTTPClientIcon_Error()
[148]      Message.Error("<b><font color='DarkRed'>ICON: ERROR</b></font><hr>" & hHTTPClientIcon.ErrorText)
[149]  End
[150]
[151]  Public Sub Form_Close()
[152]      If hHTTPClientData Then hHTTPClientData.Close()
[153]      If hHTTPClientIcon Then hHTTPClientIcon.Close()
[154]
```

```
[155]      If Exist(Application.Path &/ "WeatherIcon/weathericon.png") Then
[156]         Try Kill Application.Path &/ "WeatherIcon/weathericon.png"
[157]      Endif
[158]
[159]  End
[160]
[161]  Private Function TimeStampToDate(argTimeStamp As Integer) As Date
[162]      Return DateAdd(CDate("1/1/1970"), gb.Second, argTimeStamp)
[163]  End
```

Das Ergebnis kann sich sehen lassen:

Abbildung 24.2.4.1.2: Anzeige ausgewählter Wetterdaten

Kommentar:

- In den Zeilen 16 bis 18 werden die erforderlichen Parameter als Konstanten definiert.
- In der Zeile 25 wird überprüft, ob ein Zugang zum Internet besteht – sonst wird das Programm beendet.
- In der Zeile 27 werden die Wetterdaten abgerufen (Funktion 'GetWeatherData(...)') und in der Variablen sJSONData abgespeichert.
- Sie können sich diese (Roh-)Daten in der Zeile 28 ausgeben lassen. In den Zeilen 30 bis 32 können Sie die Daten, die ja im JSON-Format vorliegen, formatiert ausgeben.
- In den Zeilen 34 bis 48 werden Fehler – definiert in den Ereignissen HTTPClientData_Error() und HTTPClientIcon_Error() – abgefangen und mit einer Fehlermeldung angezeigt oder die Ausgabe der Wetterdaten angeschoben:

Abbildung 24.2.4.1.3: Anzeige Fehler 1

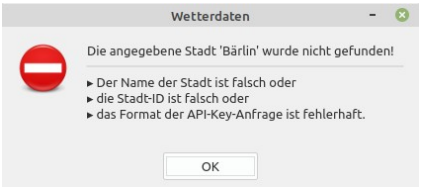

Abbildung 24.2.4.1.4: Anzeige Fehler 2

- In den Zeilen 80 bis 125 werden die Daten editiert, in geeigneter Weise – je nach Datentyp – formatiert und angezeigt.
- In der Zeile 121 wird ein passendes Icon über die Funktion 'GetAndStoreWeatherIcon(...)' angefordert und in einer Bilddatei gespeichert. Das aktuelle Icon wird zum Programmende sicher gelöscht (Zeile 157 bis 161)!
- Die Anzeige des Icons geschieht in der Zeile 123.
- Beachten Sie die Get()-Methode in der Zeile 139, der ein *leerer* Header übergeben wird und ein Datei-Pfad für die temporäre Speicherung der aktuellen Bilddatei.