7 HTTP

Nachdem wir Datum-Zeit-Angaben und eine Animation aus dem Internet bezogen haben, wollen wir uns nun dem Download von Webseiten zuwenden. Gewöhnlich können wir unter einer Web-Adresse eine Vielzahl von unterschiedlichen Dateien herunter laden - und nicht nur einen einzigen Informationsblock wie beim Daytime-Server. Wenn wir uns mit einem Browser eine bestimmte Webseite anschauen wollen, müssen wir deswegen in seiner Adresszeile neben der IP-Adresse oder dem entsprechenden Domain-Namen auch den Namen der Datei (inkl. Pfad) eintragen, um die es uns geht. Bei fast allen Browsern reicht es aus, die Web-Adresse über ein /-Zeichen mit dem Dateinamen zu kombinieren, z. B.: "g-heinrichs.de/index.htm" .

Ein Blick hinter die Kulissen: Gibt man eine solche Adress-Dateinamen-Kombination in die Adresszeile eines Browsers ein, so stellt er dieser Kombination meist automatisch die Zeichenkette "http://" voran. In unserem Fall führt das dann zu:

http://g-heinrichs.de/index.htm

Ein solcher Ausdruck wird häufig als **U**niform **R**esource **L**ocator (kurz: **URL**) bezeichnet. "http://" verweist hier auf das benutzte Protokoll HTTP, das wir jetzt behandeln werden.

HTTP-Grundlagen

Bislang reichte es aus, wenn wir mit dem Server eine Verbindung aufgebaut haben. Ohne ihm weitere Informationen zu geben, hat er uns dann die gewünschten Daten (Daytime oder Star-Wars-Animation) zugesendet. Dies war kein Problem, weil in allen diesen Fällen der Server keine anderen Aufgaben erfüllen konnte. Dies entspricht in dem Modell unserer Elektronik GmbH der Abteilung Dienstleistung (555), die auch nur eine einzige Aufgabe ausführen kann.

Nun wollen wir eine bestimmte Webseite von einem Server anfordern; dazu müssen wir ihm den Namen der gewünschten Datei übermitteln. Dabei sind bestimmte Regeln einzuhalten. Das entsprechende Regelsystem bezeichnet man als **Hyperttext Transfer Protocol** (kurz: **HTTP**). Ein **HTTP-Request** (vgl. Kapitel 2) kann z. B. so aussehen (etwas gekürzt):

```
GET /index.htm HTTP/1.1
Host: g-heinrichs.de
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-DE
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)... (hier gekürzt)
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Wenn ein Browser eine Webseite mit Hilfe dieses Protokolls anfordern soll, dann muss er zunächst die in der Adresszeile des Browsers eingegebene URL entsprechend zerlegen: In unserem obigen Beispiel schneidet er aus der URL die Zeichenkette zwischen `http://` und `/` heraus und erhält damit den **HOST-Parameter** (s. zweite Zeile des HTTP-Protokolls). Host entspricht hier dem Domain-Namen des Servers. Was bei unserer URL nun rechts vom Domain-Namen steht, setzt er als **GET-Parameter** ein. In unserem Fall ist der GET-Parameter also `/test.html`. Damit wird die Datei `test.html` aus dem Stamm-Verzeichnis des Servers angefordert. (Will man auf eine Datei aus einem Unterordner des Servers zugreifen, dann muss hier eine entsprechende Pfadangabe gemacht werden.) Zu guter Letzt wird hinter dem GET-Parameter noch die benutzte Protokoll-Version (üblicherweise HTTP/1.1) angegeben. Übrigens: Gibt man in die Adresszeile des Browsers nur den Domain-Namen ein, so hängt dieser automatisch ein /-Zeichen an; man erhält in diesem Fall als GET-Parameter dann das /-Zeichen.

Zusammengefasst: Der HOST-Parameter benennt den Server, an den sich der Request richtet; und der GET-Parameter gibt an, was dieser Server dann an den Client senden soll. Mancher Leser wird sich an dieser Stelle vielleicht fragen, wozu auf der HTTP-Ebene der Host-Name überhaupt benötigt wird, werden die Daten doch mittels der IP-Adresse schon an den richtigen *Server* gelangen. Warum dies aber in vielen Fällen nicht ausreicht, um auch an die gewünschten *Webseiten* zu gelangen, das habe ich schon in Kapitel 2 erläutert.

> *Ein Blick hinter die Kulissen:* GET-Parameter können nicht nur zur Anforderung von Dateien eingesetzt werden. In den nächsten Kapiteln werden wir sie tatsächlich zu verschiedenen anderen Zwecken einsetzen. Neben dem GET-Parameter gibt es auch andere Parameter (z. B. POST-Parameter), mit denen bestimmte Aufträge an den Server übermittelt werden. Da wir sie nicht benutzen werden, wollen wir sie auch nicht weiter verfolgen.

Wenn wir mit unserem TTGO-Modul nun selbst einen HTTP-Request bei einem Web-Server durchführen wollen, benutzen wir der Einfachheit halber direkt den Domain- bzw. Host-Namen und den Dateinamen. Bei dem Request halten wir uns streng an das Schema unseres Beispiels:

Zunächst schreiben wir die GET-Zeile mit dem entsprechenden Parameter, gefolgt von der Angabe des benutzten Protokolls (HTTP/1.1). Darauf folgt in der nächsten Zeile die Angabe des Hosts.Danach können bei Bedarf weitere Informationen angegeben werden, z. B. zum benutzten Browser, zur verwendeten Sprache oder auch, welche Datei-Typen der Client akzeptiert. Diese Angaben werden mit einer Leerzeile abgeschlossen. **Diese Leerzeile ist unbedingt erforderlich und darf daher nicht vergessen werden!**

In unserem Kontext muss also jeder HTTP1.1-Request mindestens aus einer GET-Zeile, einer HOST-Zeile, und einer anschließende Leerzeile bestehen. Im Folgenden werden wir uns mit einem solchen **Minimal-Request** begnügen.

Wenn die Anfrage nun korrekt ist, gibt der Server eine Antwort (**HTTP-Response** genannt); diese beginnt mit einer Statusmeldung **(HTTP-Header**); hier ein (etwas gekürztes) Beispiel:

```
HTTP/1.1 200 OK
Date: Thu, 11 Jun 2020 14:15:28 GMT
Server: Apache
Accept-Ranges: bytes
Content-Length: 1269
Content-Type: text/html
```

Nach einer Leerzeile folgen dann die gewünschten Daten. Wird die angefragte Datei nicht auf dem Server gefunden, wird als Status `404 Not Found` ausgegeben.

Ein einfacher HTTP-Client

Nun sind wir bereit, eine Webseite aus dem Internet herunter zu laden. Dazu habe ich eine einfache Text-Datei mit dem Dateinamen 'hallo.txt' auf dem Server meiner Website 'g-heinrichs.de' gespeichert. (Auf eine HTML-Datei habe ich an dieser Stelle verzichtet, weil unser `print`-Befehl die darin enthaltenen Formatierungsbefehle nicht interpretieren kann. Dazu später mehr!) Für unser HTTP-Client-Programm können wir auf unser Daytime-Programm `sta_client_time_1.py` zurückgreifen. Nur wenige Änderungen sind erforderlich:

Zunächst muss als Server-Adresse `('g-heinrichs.de', 80)` eingegeben werden. Üblicherweise wird die Portnummer 80 für einen HTTP-Dienst verwendet.

Jetzt müssen wir unseren Request senden, sobald die Sockets von Client und Server verbunden worden sind:

```
request = 'GET /hallo.txt HTTP/1.1\r\nHost:g-heinrichs.de\r\n\r\n'
client.send(request)
```

Die Escape-Sequenzen `\r\n` sorgen hier für die erforderlichen Zeilenwechsel und die Leerzeile.

Die Datei `hallo.txt` ist nicht allzu lang: sie besteht aus deutlich weniger 1000 Bytes, wir können es also bei der maximalen Anzahl der Bytes von 1024 belassen:

```
data = client.recv(1024)
print(str(data,'UTF-8'))
```

Führen Sie die Änderungen durch und testen Sie das neue Programm dann aus; schauen Sie sich auch die Statusmeldungen an. Welche Informationen lassen sich ohne weitere Vorkenntnisse verstehen? Testen Sie auch aus, wie sich der Server verhält, wenn der Request fehlerhaft ist.

An einigen Stellen lässt sich das Programm noch etwas verbessern. So kann man die Zeichenkette für den Request einfacher eingeben, indem man `'''` als Anführungszeichen benutzt. In diesem Fall kann man nämlich die Zeichenkette auf mehrere Zeilen verteilen. Dabei werden die im Editor

vorgenommenen Zeilenwechsel automatisch als NL-Steuerzeichen (s. Kapitel 6) in die Zeichenkette übernommen. Auf diese Weise können wir die Zeichenkette für den Request nun übersichtlicher eingeben; allerdings müssen wir dabei die Escape-Sequenz `\r` für das Steuerzeichen CR weiterhin selbst eingeben:

```
request = '''GET /hallo.txt HTTP/1.1\r
Host:g-heinrichs.de\r
\r
'''
```

Schließlich sollte noch berücksichtigt werden, dass es bei der Ausführung des Programms Probleme bei der Herstellung von Verbindungen auftauchen können. Auch sollten alle bestehenden Verbindungen am Ende des Programms wieder gelöst werden. Die Vorgehensweise ist hier die gleiche wie beim Daytime-Programm. Die nötigen Ergänzungen seien dem Leser als Übungsaufgabe überlassen; er kann aber auch auf das Programm `sta_http_client_1.py` zurückgreifen.