Topic:

Web-IO programming with ASCII sockets

Query and set IO-signals using readable commands

First we show the basic sequence and a brief configuration guide for accessing our Web-IO products using ASCII sockets over TCP or UDP. The page is subdivided as follows:

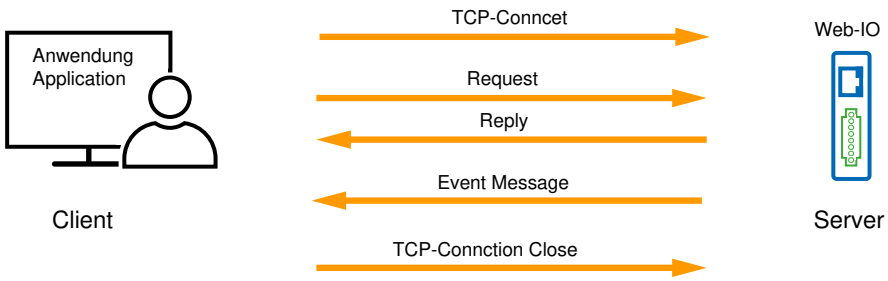
- Access via TCP and configuration of a Web-IO
- Access via UDP and configuration of a Web-IO
- Structure of the ASCII commands

It makes no difference which programming language is used, since the basic principles remain the same. Detailed examples for various high level languages are linked to below.

Access via TCP

Access via TCP

Similar to a telephone conversation, with TCP you must first open a connection before information can be exchanged. The client opens the connection, which is then accepted by the server. In applications for Web-IO with ASCII sockets the application always assumes the role of client and the Web-IO is the server.



The Web-IO also has a TCP port opened for accepting a connection (port 42280 unless otherwise configured). The client, i.e. the application, opens a connection to the IP address of the server to this port. The client also opens a TCP port (dynamically alternating) on which it can receive data sent by the server.

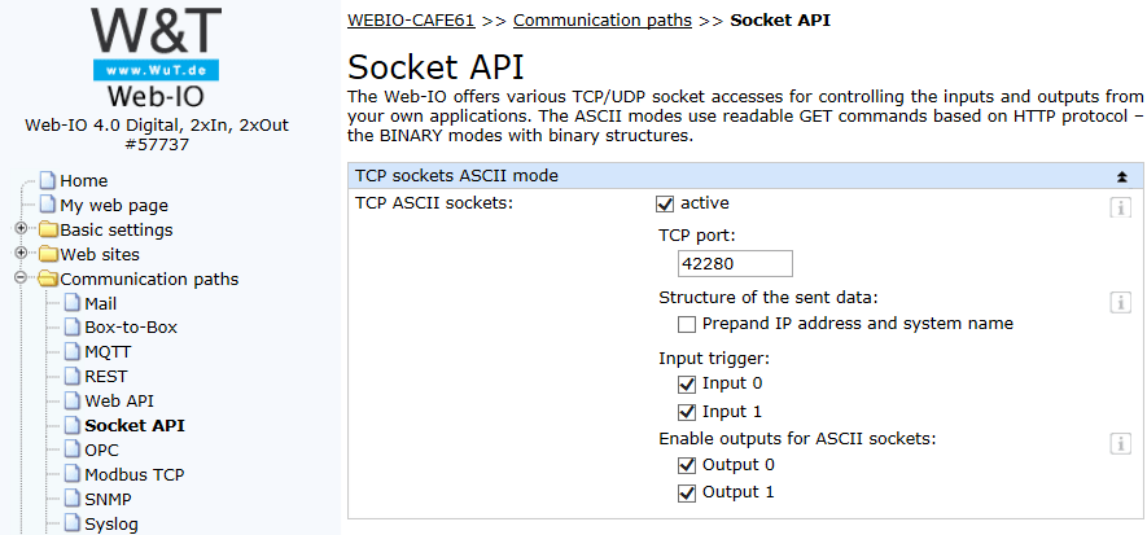
As soon as there is a connection, the application can send commands to the Web-IO (Request) which the Web-IO then replies to and executes (Reply). If so configured the Web-IO can indicate the status of the input without being requested by the application (Event Message).

In general the application closes the TCP connection. The Web-IO closes the connection only in case of errors, e.g. if the syntax of a command is incorrect.

In TCP-ASCII socket mode the Web-IO can handle maximum four connections at the same time. Any attempt to open a connection by a fifth client is rejected (similar to busy on a telephone).

Configuring Web-IO for TCP access

From the menu tree in the web interface, select *Communication channels >> Socket API* and enable *TCP-ASCII sockets*.



The *TCP port* can if needed be adapted to your own application.

If the Web-IO needs to automatically send the status to the application when the inputs change, check the corresponding boxes under *Input Trigger*.

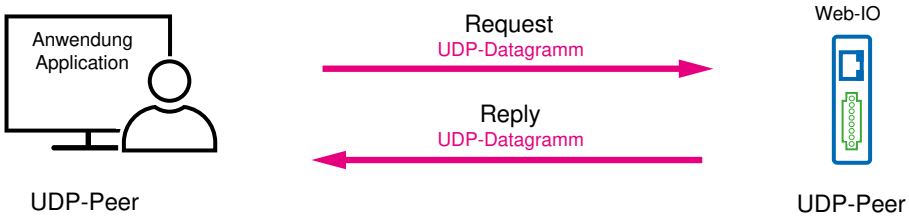
In addition, outputs which need to be switched by the application must be enabled.

Prepending the *IP address and system name* will be covered later.

Access via UDP

Access via UDP

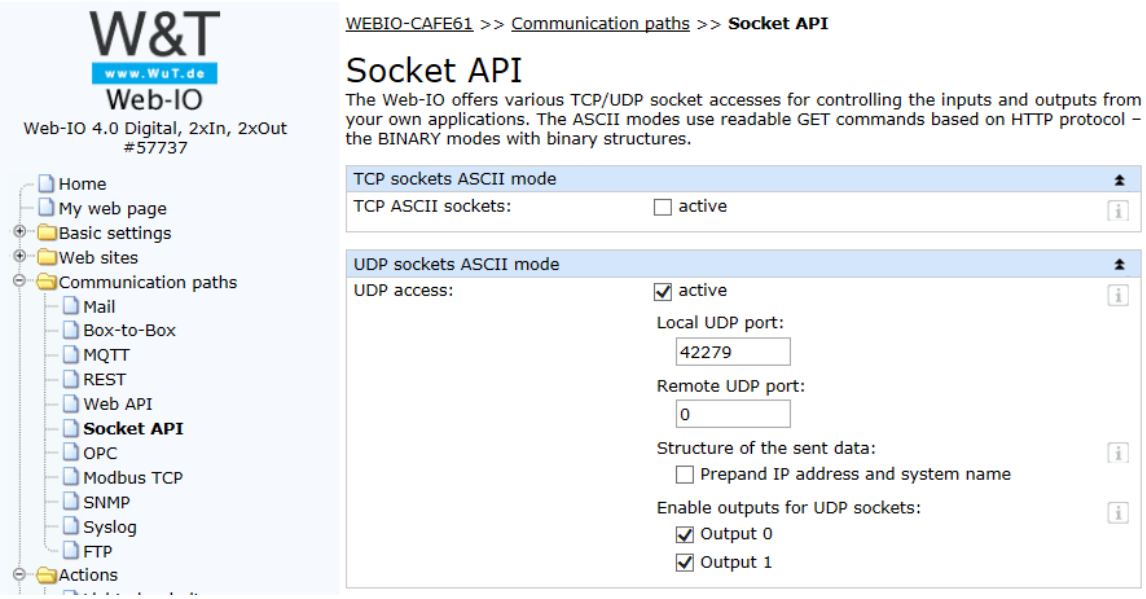
Unlike a telephone conversation, there is no connection when using UDP. Similar to radiotelephony the information is simply sent. There is no client or server, rather there are UDP peers which are all equally authorized to send their datagrams.



The application sends commands to the Web-IO (Request) which the Web-IO then responds to and executes (Reply).

Configure Web-IO for UDP access

From the menu tree in the web interface, select *Communication channels >> Socket API* and enable *UDP Sockets*.



The *TCP port* can if needed be adapted to your own application.

Outputs that need to be switched by the application must be correspondingly enabled.

Prepending the *IP address and system name* will be covered later.

ASCII commands

Structure of the ASCII commands

Regardless of whether you are accessing via TCP or UDP, the commands and replies are the same.

The structure of the commands is based on the original HTTP protocol. Each command begins with the introductory sequence `GET /` followed by the actual command. Upper and lower case must be observed exactly as shown. Following parameters are separated by `?`. With the Web-IO Digital always use the password having the syntax `PW=`*password*`&`. Each parameter set is terminated with `&`, so that two parameter sets are also separated by `&`.

IMPORTANT: *Commands including parameter sets may not be terminated with* CR *(Carriage Return) or* LF *(Line Feed) or both. Many programming languages use the methods* `write` *and* `writeln` *. Do not use the* `writeln` *method, since then* CR *and* LF *will be automatically appended.*

Example: Input query

As an example, here is the command for calling the input switching states. The Web-IO is protected with the password *blue*.

```
GET /input?PW=blau&
```

The reply from the Web-IO looks like this depending on the model:

`input;1` (Web-IO Digital 2xIn, 2xOut)
`input;0001` (Web-IO Digital 12xIn, 12xOut and 12xIn, 6xRelay)

The structure consists of the keyword for the query, separated by a semicolon followed by one or more values. As a terminator the Web-IO adds a *null character*.

Here the reply begins with the keyword `input` and is followed by the value `1` or `0001`, which in hex represents the switching state of all inputs. In this case Input 0 = ON, all other inputs are OFF.

For better understanding of the hexadecimal input coding, here is another example for a Web-IO with 12 inputs:

Inputs 0, 1, 5, 7, 10 and 11 are ON. The other inputs are in the OFF state. This bit pattern results in a dual number where ON bits = 1 and OFF bits = 0.

And so here `1100 1010 0011`.
In hex this corresponds to `CA3` and in four-place representation `0CA3`.
More about hexadecimal number can be found here.

Example: Setting the outputs

The command for setting outputs has a similar structure as the `input` command but has more parameters:

```
GET /outputaccess0?PW=blau&State=ON&
```

The digit behind the keyword `outputaccess` indicates which output to switch - here Output 0. `State` is used to send which state to switch to. Possible values are `ON`,`OFF` and `TOGGLE` (state change).

The reply from the Web-IO looks like this depending on the model:

`output;1` (Web-IO Digital 2xIn, 2xOut)
`output;0001` (Web-IO Digital 12xIn, 12xOut and 12xIn, 6xRelay)

Just as with the input query, the reply sent is the corresponding keyword followed by the bit pattern of the output states plus null byte. The bit pattern always represents the state of all the outputs, regardless of which output is set.

A detailed description of all the commands can be found in the command overview or in the programming manual for the Web-IO.

Examples in various programming languages

Visual Basic.Net

TCP-ASCII client
for Web-IO Digital 4.0

Visual C#

TCP-ASCII client
for Web-IO Digital 4.0

Visual C++

TCP-ASCII client
for Web-IO Digital 4.0

Delphi

TCP-ASCII client
for Web-IO Digital 4.0

Products

Web-IO 4.0 Digital
2xIn, 2xOut

Power via PoE also when needed

Web-IO 4.0 Digital
12xIn, 12xOut

12x outputs (6-30V),
12x inputs (8-30V)

Other Web-IOs

All W&T Web-IO Digital 24V

We are available to you in person:

Wiesemann & Theis GmbH
Porschestr. 12
42279 Wuppertal
Phone: +49 202/2680-110 (Mon.-Fri. 8 a.m. to 5 p.m.)
Fax: +49 202/2680-265
info@wut.de

Data Privacy