Thema:

Web-IO Programmierung mit Binary-Sockets

IO-Signale mit Binär-Strukturen abfragen und setzen

An dieser Stelle wird zunächst der prinzipielle Ablauf für das Ansprechen unserer Web-IO-Produkte mit Binary-Sockets über TCP oder UDP gezeigt. Die angewandte Programmiersprache spielt dabei keine Rolle, da die Prinzipien übergreifend die gleichen sind. Im Anschluss finden Sie eine Anleitung zur Konfiguration von Web-IOs für den Binary-Zugriff sowie Informationen zu Aufbau und Struktur von Binary-Strukturen.

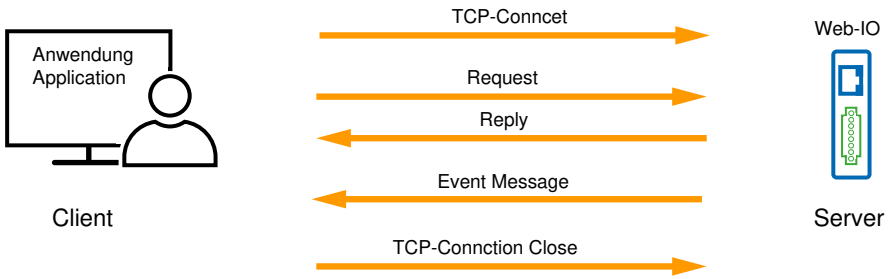
Detaillierte Beispiele für unterschiedliche Hochsprachen sind weiter unten verlinkt.

Zugriff über TCP/UDP

Zugriff über TCP

Ähnlich einem Telefongespräch muss bei TCP zunächst eine Verbindung aufgebaut werden, bevor Informationen ausgetauscht werden können.

Dabei baut der Client die Verbindung auf, die vom Server entgegengenommen wird. Bei Anwendungen für Web-IO mit Binary-Sockets kann das Web-IO je nach Konfiguration entweder als Server oder als Client arbeiten. In den meisten Fällen übernimmt allerdings die Anwendung den Part des Clients und das Web-IO ist der Server.



Das Web-IO als Server
Als Server hat das Web-IO einen TCP-Port für die Verbindungsannahme geöffnet (wenn nicht anders konfiguriert Port 49153). Der Client, also die Anwendung, baut eine Verbindung zur IP-Adresse des Servers zu diesem Port auf. Dazu öffnet auch der Client einen TCP-Port (dynamisch wechselnd), auf dem er Datensendungen vom Server, also vom Web-IO, entgegennehmen kann.

Sobald die Verbindung besteht, kann die Anwendung Daten zum Web-IO senden (Request), die dann vom Web-IO beantwortet bzw. ausgeführt werden (Reply). Bei entsprechender Konfiguration kann das Web-IO ohne Anfrage durch die Anwendung den Status der Inputs übermitteln (Event Message).

Das Schließen der TCP-Verbindung übernimmt im Regelfall die Anwendung. Das Web-IO schließt die Verbindung nur im Fehlerfall.

Im TCP-Binary-Socket-Modus kann das Web-IO zeitgleich auf dem geöffneten TCP-Port genau eine Verbindung entgegennehmen. Der Verbindungsversuch eines weiteren Clients würde abgewiesen (ähnlich besetzt beim Telefon).

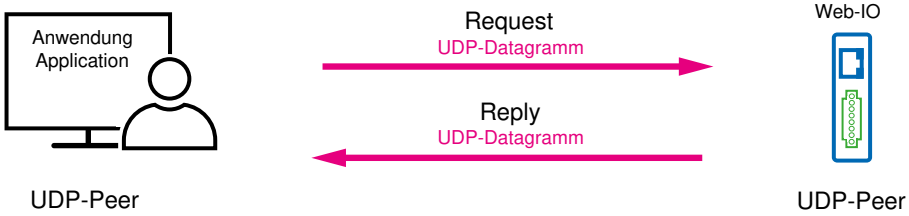
Das Web-IO als Client
Als Client übernimmt das Web-IO den Verbindungsaufbau. Angestoßen durch eine Signaländerung an den Inputs wird die Verbindung zu einem vorkonfigurierten Server hergestellt.

Sobald die Verbindung steht, wird der Status der Inputs übertragen. Solange die Verbindung besteht, können genauso Daten ausgetauscht werden wie in der Betriebsart Web-IO als Server.

Der Verbindungsabbau wird über einen *Inactivity Timeout*, also ein Zeitlimit, angestoßen. Erfolgt für die eingestellte Zeit keine Änderung an den Inputs, schließt das Web-IO die Verbindung.

Zugriff über UDP

Anders als bei einem Telefongespräch gibt es bei UDP keine Verbindung. Ähnlich wie beim Sprechfunk werden die Informationen einfach gesendet, Client und Server gibt es also bei UDP nicht. Stattdessen spricht man von UDP-Peers, die gleichberechtigt ihre Datagramme versenden.



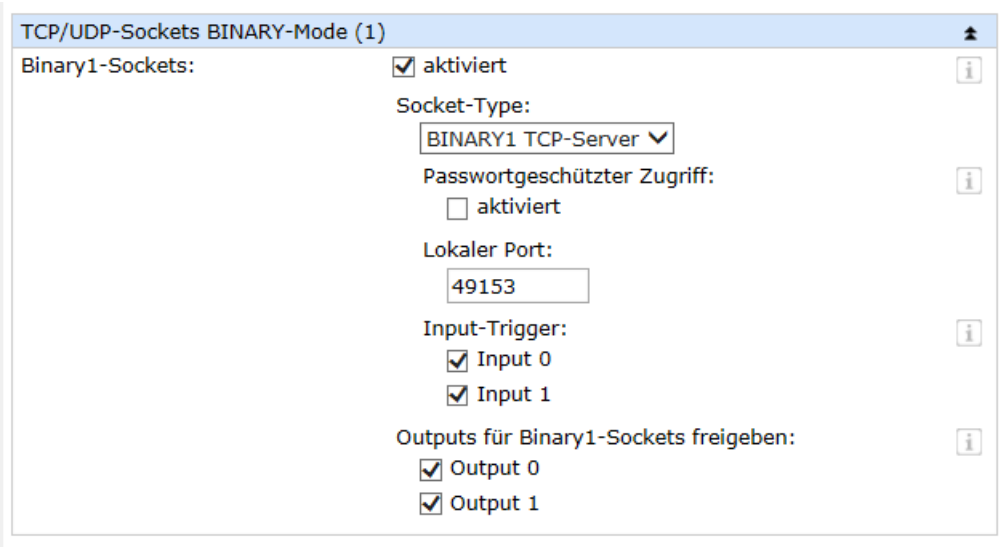
Die Anwendung sendet Daten zum Web-IO (Request), auf die das Web-IO entsprechend reagiert und wenn nötig eine Antwort sendet (Reply). Darüber hinaus sendet das Web-IO bei entsprechender Konfiguration ein Datagramm, wenn sich der Status der Inputs ändert.

Web-IO konfigurieren

Web-IO für Binary-Zugriff konfigurieren

Wählen Sie im Menübaum der Web-Oberfläche *Kommunikationswege >> Socket-API* und aktivieren Sie *Binary1-Sockets*. Die weitere Konfiguration hängt davon ab, welche Art des Socket-Zugangs benötigt wird.

Web-IO als Binary-Server Als *Socket-Type* wählen Sie *BINARY1 TCP-Server*.

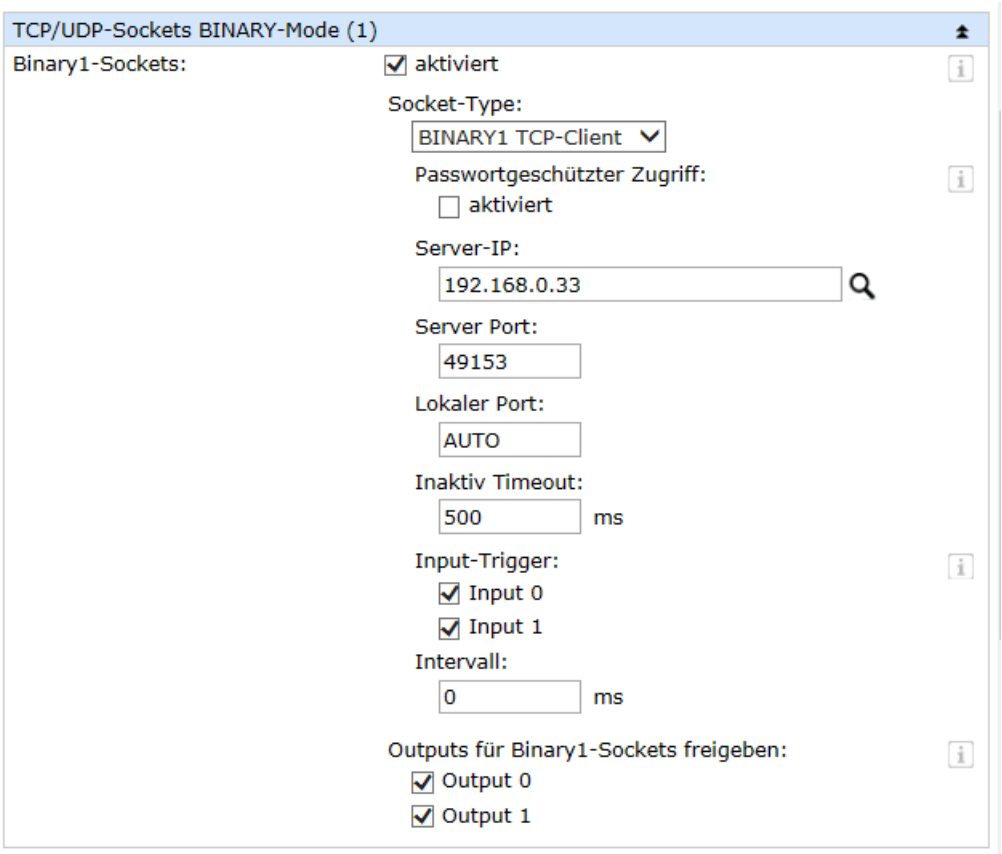


Der lokale Port kann bei Bedarf an die eigene Anwendung angepasst werden.

Wenn das Web-IO bei Änderung an den Inputs den Zustand automatisch an die Anwendung senden soll, setzen Sie die entsprechenden Häkchen unter *Input-Trigger*.

Außerdem müssen Outputs, die über die Anwendung geschaltet werden sollen, freigegeben werden.

Web-IO als Binary-Client Als *Socket-Type* wählen Sie *BINARY1 TCP-Client*.



Als *Server-IP* wird die IP-Adresse des Netzteilnehmers eingetragen, auf dem die Server-Applikation aktiv ist, mit der sich das Web-IO verbinden soll.

Der *Server Port* ist der Port, auf dem die Server-Applikation die Verbindung entgegen nimmt. Dieser Port kann bei Bedarf angepasst werden.

Der lokale Port sollte normalerweise auf *AUTO* stehenbleiben, damit er dynamisch vergeben wird. Wenn die Infrastruktur es verlangt, kann aber auch ein spezifischer Port eingetragen werden.

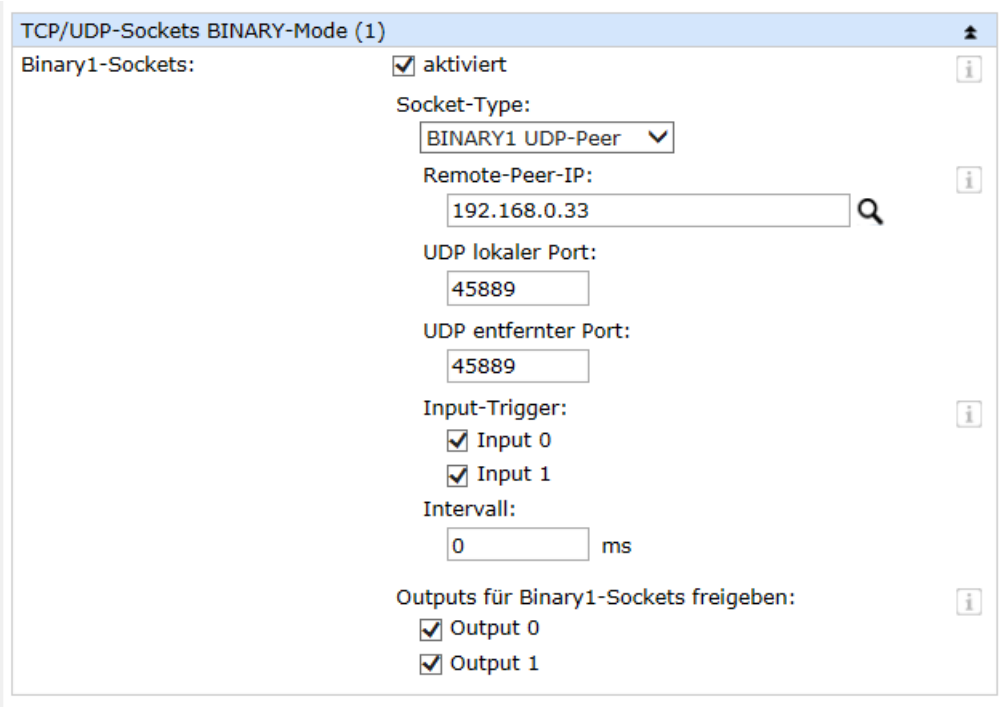
Inputs, die bei Statusänderung einen Verbindungsaufbau auslösen sollen, müssen unter *Input-Trigger* aktiviert werden. Nach

Zustandekommen der Verbindung werden die Zustände automatisch an die Anwendung gesendet.

Über das Feld *Intervall* kann festgelegt werden, ob und in welchem Zyklus eine Verbindung aufgebaut werden soll.

Zuletzt müssen Outputs, die über die Server-Anwendung geschaltet werden sollen, freigegeben werden.

Web-IO als Binary-UDP-Peer Als *Socket-Type* wählen Sie *BINARY1 UDP-Peer*.



Als *Remote-Peer-IP* wird die IP-Adresse des Netzteilnehmers eingetragen, an den das Web-IO Datagramme versenden soll.

UDP lokaler Port und *UDP entfernter Port* können bei Bedarf an die eigene Anwendung angepasst werden.

Wenn das Web-IO bei Änderung an den Inputs den Zustand automatisch an die Anwendung senden soll, setzen Sie die entsprechenden Häkchen unter *Input-Trigger*.

Über das Feld *Intervall* kann festgelegt werden, ob und in welchem Zyklus Datagramme an die Anwendung gesendet werden sollen.

Outputs, die über die Anwendung geschaltet werden sollen, müssen entsprechend freigegeben werden.

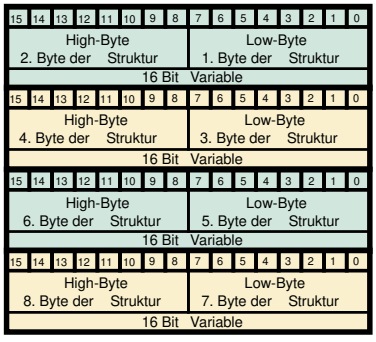
Binary-Strukturen

Aufbau und Struktur der Binary-Strukturen

Unabhängig davon, ob per TCP oder UDP zugegriffen wird - die Strukturen die ausgetauscht werden, sind die gleichen.

Datenstrukturen sind Einzelvariablen, die bündig hintereinander als Byte-Folge im Speicher abgelegt sind. In dieser Anordnung werden sie auch über das Netzwerk versendet. Alle Binär-Strukturen für das Web-IO bestehen aus vier 16-Bit-Werten, denen je nach Aufgabe der Struktur weitere 8-, 16- oder 32-BitWerte folgen können.

Die Struktur *EA-Driver*



Start_1	Integer = 0
Start_2	Integer = 0
Struct Type	Integer
StructLength	Integer Länge/Length in Byte

Start_1 und Start_2 Diese beiden 16-Bit-Variablen haben immer den Wert 0 und bilden als Startsequenz den Anfang jeder Struktur.

StructType Über den Struktur-Type ist codiert, was das Web-IO tun soll bzw. auf was sich seine Anwort bezieht. Strukturen eines bestimmten Types haben einen type-spezifischen Aufbau.

StructLength Gibt die Gesamtlänge einer Struktur in Bytes an.

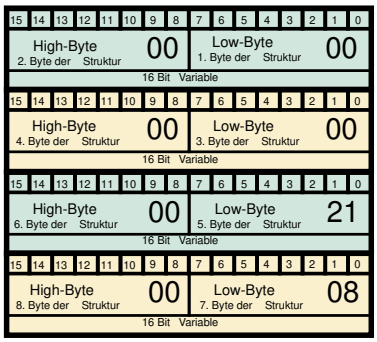
Was auf den ersten Blick kompliziert anmutet, hat einen großen Vorteil: Bestimmte Informationen stehen innerhalb der Struktur immer an der selben Stelle und sind dort sofort auffindbar. Es entfällt also das *Parsen* (Untersuchen) von Zeichenketten auf bestimmte

Inhalte.

Beispiel: Abfrage von Inputs und Outputs

Mit dem folgenden Beispiel wollen wir zeigen, wie der Status der Inputs und Outputs angefordert werden kann und wie die entsprechende Antwort vom Web-IO aufgebaut ist.

Der Strukturtyp RegisterRequest Die *RegisterRequest*-Struktur entspricht vom Aufbau der Grundstruktur *EA-Driver.* Als *StructType* wird 0x21 hex. (dezimal 33) eingefügt und *StructLength* auf 8 (Byte) gesetzt.



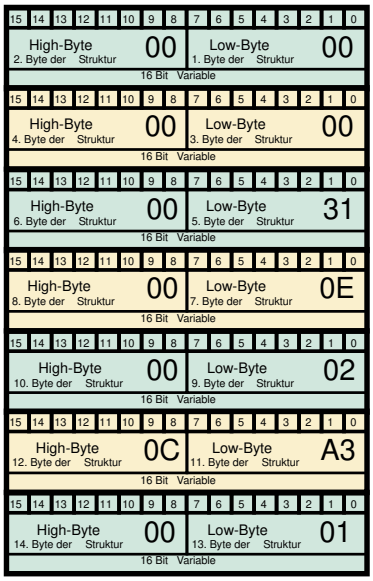
Start_1

Start_2

Struct Type

StructLength

Der Strukturtype RegisterState Als Anwort sendet das Web-IO eine Struktur vom Type *RegisterState*.



Start_1

Start_2

Struct Type

StructLength

DriverID

InputValue

OutputValue

Auch bei der Struktur *RegisterState* entspricht der erste Teil der Grundstruktur *EA-Driver*. Als *StructType* wird 0x31 hex. (dezimal 49) eingefügt und *StructLength* auf 0x0E hex. (dezimal 14 Byte) gesetzt.

Allerdings folgen drei weitere 16-Bit-Variablen. Die Variable *DriverID* hat eigentlich keine Bedeutung mehr, ist aber aus Kompatibilitätsgründen zu älteren Web-IO-Modellen mit dem Wert 2 noch Bestandteil der Struktur.

Die Variablen *InputValue* und *OutputValue* geben die Schaltzustände der Inputs und Outputs wieder. Der eingetragene Wert entsprich dem Bit-Muster der Inputs bzw. Outputs. Die Codierung ist einfach - hier ein Beispiel für die 12 Inputs eines Web-IO #57730:

Im Zustand ON sind die Inputs 0,1,5,7,10,11. Die anderen Inputs sind im Zustand OFF.

Hier also `1100 1010 0011`. In hexadezimaler Schreibweise entspricht das `0xCA3` oder dezimal 3235.

Im oben gezeigten Beispiel ist der Wert für*OutputValue* 1, es ist also nur Output 0 eingeschaltet, also ON. Alle andere Outputs haben den Zustand OFF.

Eine detaillierte Beschreibung zur Kommunikation mit Binary-Strukturen finden Sie in der Binary-Kurzübersicht oder im Programmierhandbuch zum Web-IO.

Beispiele in verschiedenen Programmiersprachen

Visual Basic Net

TCP-Binary-Client
für Web-IO Digital 4.0

Visual C#

TCP-Binary-Client
für Web-IO Digital 4.0

Delphi

TCP-Binary-Client
für Web-IO Digital 4.0

Produkte

Web-IO 4.0 Digital
2xIn, 2xOut

Bei Bedarf auch über PoE zu versorgen

Web-IO 4.0 Digital
12xIn, 12xOut

12x Ausgänge (6-30V),
12x Eingänge (8-30V)

Weitere Web-IOs

Alle W&T Web-IO Digital 24V

W&T
www.WuT.de

Wir sind gerne persönlich für Sie da:

Wiesemann & Theis
GmbH
Porschestr. 12
42279 Wuppertal
Tel.: 0202/2680-110 (Mo-Fr. 8-17
Uhr)
Fax: 0202/2680-265
info@wut.de

Datenschutz