TCP/IP-Sockets - Binary

Neben dem Zugriff mit Kommandostrings bietet das Web-IO auch vier Socket-Zugänge für binären Datenaustausch.

Alle vier Zugänge arbeiten unabhängig von einander. Unter *Kommunikationswege >> Socket-API* können Sie die Binary-Zugänge aktivieren und festlegen, in welchem der drei folgenden Modi gearbeitet werden soll:

- TCP-Server
- TCP-Client
- UDP-Peer

Für die verschiedenen Funktionen wie Lesen der Inputs, Setzen der Outputs usw. definiert das Web-IO binäre Strukturen. Der Zugriff erfolgt ausschließlich durch Austausch dieser Strukturen.

TCP-Server

Wählen Sie unter *Kommunikationswege >> Socket-API* im gewünschten Binary-Bereich als *Socket-Type BINARYx TCP-Server* und bestimmen Sie auf welchem *lokalen Port* die Verbindung entgegen genommen werden soll.

Anders als in der Betriebsart TCP Server für Kommandostrings, in der sich bis zu 8 Clients auf den gemeinsamen Server-Port verbinden können, lässt das Web-IO für den jeweiligen Binary-Zugang nur eine Verbindung zu. Solange ein Client verbunden ist, werden alle weiteren Verbindungsversuche abgewiesen.

Bei Bedarf kann der Binary-Zugang mit einem Passwortschutz versehen werden.

Bei gesetzten Input-Triggern meldet das Web-IO bei bestehender Verbindung von sich aus Veränderungen an den ausgewählten Inputs an die Anwendung.

Sollen auch die Outputs über den Binary-Zugang schaltbar sein, muss das unter *Outputs für Binaryx-Sockets freigeben* aktiviert werden.

TCP-Client

Wenn im gewählten Binary Bereich als *Socket-Type BINARYx TCP-Client* gewählt ist,

baut das Web-IO bei Änderung an einem der unter *Input-Trigger* ausgewählten Inputs eine Verbindung zu einer TCP-Server-Applikation auf.

Geben Sie unter *Server-IP* die IP-Adresse oder den Host-Namen des Ziel-Servers ein und legen Sie unter *BINARYx Server Port* vor, auf welchen TCP-Port die Verbindung aufgebaut werden soll.

Im Normalfall sollte der *BINARYx lokale Port* auf *AUTO* konfiguriert sein. Damit zählt der Absende-Port mit jeder Verbindung dynamisch hoch, was für die Überwindung von Firewalls hilfreich ist. Alternativ kann aber auch mit einer willkürlich gewählten, festen Absende-Port gearbeitet werden.

Unter *Inaktiv Timeout* können Sie bestimmen, nach welcher Zeit ohne Aktivität an den Inputs eine Verbindung getrennt werden soll. Der Wert *0* bedeutet, das die Verbindung solange geöffnet bleibt, bis sie von der Gegenseite getrennt wird.

Sollen auch die Outputs über den Binary-Zugang schaltbar sein, muss das unter *Outputs für Binaryx-Sockets freigeben* aktiviert werden.

UDP-Peer

Wenn im gewählten Binary Bereich als *Socket-Type BINARYx UDP-Peer* gewählt ist, öffnet das Web-IO den unter *UDP lokaler Port* eingestellten UDP Port für den Empfang von UDP-Datagrammen.

Alle vom Web-IO gesendeten Datagramme gehen an die unter *Remote-Peer-IP* eingetragene IP-Adresse auf den als *UDP entfernter Port* konfigurierten UDP-Port.

Bei Änderung an den als *Input-Trigger* definierten Inputs werden entsprechende Datagramme an den konfigurierten Remote-Peer gesendet.

Sollen auch die Outputs über den Binary-Zugang schaltbar sein, muss das unter *Outputs für Binaryx-Sockets freigeben* aktiviert werden.

BINARY - Die IO-Strukturen

Unabhängig davon, welcher BINARY Socket Type gewählt wurde - für die Kommunikation zwischen der Anwendung und dem Web-IO, gibt es eine übersichtliche Menge binärer Strukturen (Variablenfeldern).

Für folgende Funktionen werden solche Strukturen angeboten:

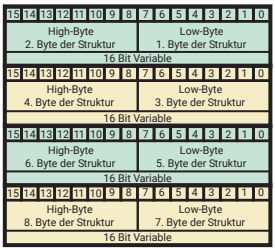
- Passwort-Anforderung und -Übergabe
- Lesen der Inputs
- Lesen der Inputs und Outputs
- Setzen der Outputs
- Lesen der Counter
- Zurücksetzen der Counter
- Parametrieren der zyklischen und automatischen Benachrichtigung bei Zustandsänderung

Das Anwenderprogramm nutzt die einfach zu handhabende Socket-Schnittstelle (Windows: WinSock, UNIX, Linux: Berkley Sockets), um die Daten in Form dieser IO-Strukturen mit dem Web-IO über das Netzwerk per TCP/IP auszutauschen.

Definition der IO-Strukturen

Um den Inhalt eines Paketes eindeutig identifizieren und auswerten zu können, müssen im BINARY-Modus alle Daten in Form dieser binären Strukturen an das Web-IO gesendet werden.

Alle Strukturen beginnen mit dem gleichen Header, der aus den folgenden 4 WORDs (16bit_Integer) besteht:



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Start_1, Start_2

Start_1 und *Start_2* sind aus Gründen der Kompatibilität zu Vorgängermodellen vorhanden, werden aber nicht benutzt. Beide Werte sind immer 0.

StruktTyp

Der Wert struct_typ identifiziert, um welche Struktur es sich handelt. Sowohl Die PC-Anwendung als auch das Web-IO entscheiden bei Datenempfang anhand des Wertes *struct_typ,* wie die Struktur ausgewertet werden soll.

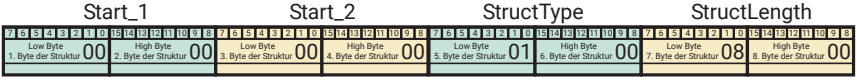
StructLength

length gibt die Gesamtlänge der Struktur in Bytes an, also inklusive der ersten 4 WORDs.

Festlegungen für alle verwendeten Binär-Strukturen

- Ein WORD entspricht 16bit_integer. (ohne Vorzeichen)
- Ein BYTE entspricht einem Byte (8Bit)
- Ein LONG entspricht einem 32bit_integer (ohne Vorzeichen)

Beim Senden und Empfangen gilt für alle Struktur-Variablen: Low-Byte first.



In den Beispielen sind alle Zahlen hexadezimal dargestellt!

Passwort geschützter Zugang

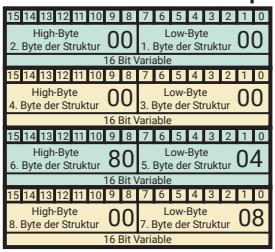
Im TCP-Server Modus kann der Binary-Zugang über eine Passwortabfrage vor unautorisiertem Zugriff geschützt werden.

Wenn unter *Kommunikationswege >> Socket-Zugang > TCP/UDP-Sockets BINARY-Mode* im Bereich *BINARYx TCP-Server der* Punkt *Passwortgeschützter Zugriff* aktiviert ist, muss eine mehrstufige Login-Prozedur durchlaufen werden.

Unautorisierter Zugriffsversuch

Versucht eine Client-Anwendung über den mit Passwort geschützten Binary-Zugang die Kommunikation mit dem Web-IO zu beginnen, sendet das Web-IO die Struktur *AuthRequired*.

Die Struktur Authrequired



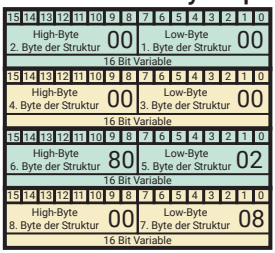
Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Wird die *AuthRequired* Struktur empfangen muss die Client-Anwendung mit der Login-Prozedur beginnen.

Die Login-Prozedur

Um bei gesetztem Passwortschutz Zugriff auf die IOs zu bekommen, muss die Client-Anwendung beim Web-IO einen Schlüssel anfordern. Das erfolgt über senden der Struktur *KeyRequest*

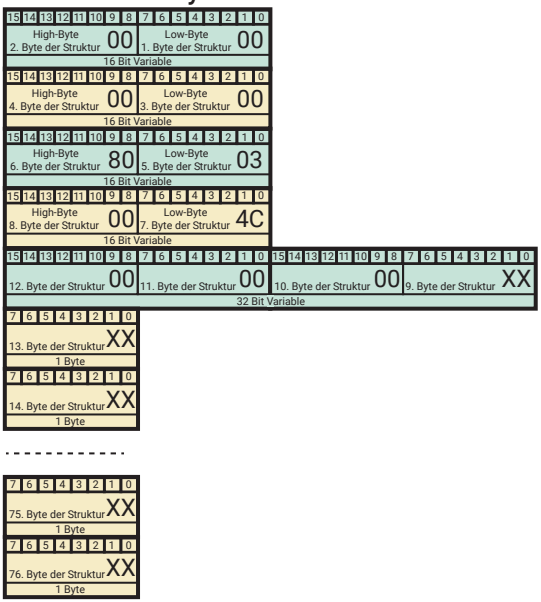
Die Struktur KeyRequest



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Das Web-IO generiert bei Empfang der Struktur *KeyRequest* einen maximal 64 Byte langen Schlüssel, der eingebettet in die Struktur *Key* an das Web-IO gesendet wird.

Die Struktur Key

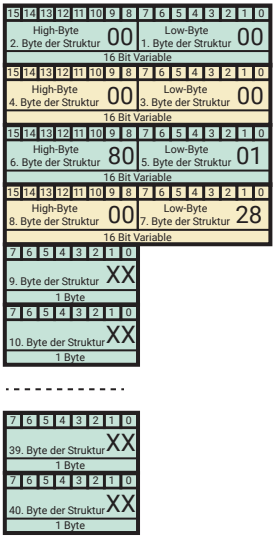


Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
KeyLength	Long Anzahl der tatsächlich genutzten KeyBytes
KeyBytes	Long Byte-Array immer 64 Bytes, auch wenn der tatsächliche Key kürzer ist

Wieviel Byte lang der Schlüssel tatsächlich ist, wird mit der Variablen *KeyLength* übergeben.

Die Struktur Login

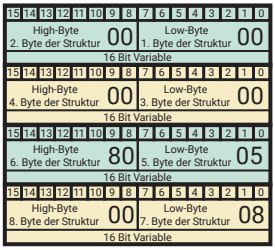
Die Client-Anwendung muss nun aus dem für das Web-IO vergebenen Passwort (Admin oder Benutzer) und dem empfangenen Schlüssel eine MD5-Hash-Summe bilden.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
MD5-Hash	Long Byte-Array 32 Bytes MD5-Hash von Password + Key

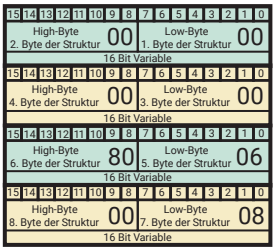
Die MD5 Hash-Summe wird mit der *Login* Struktur an das Web-IO gesendet. Wenn das Web-IO den Login akzeptiert sendet es die Struktur *AuthOK* an die Client-Anwendung und es kann der reguläre Datenaustausch beginnen.

Die AuthOK Struktur



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Die AuthFail Struktur



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

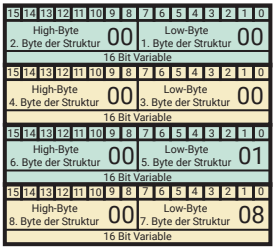
Bei inkorrektem Login sendet das web-IO die *AuthFail* Struktur und schließt die TCP-Verbindung.

Digital-IO-Zugriffe mit Binär-Strukturen

Im folgenden werden die einzelnen Strukturen erläutert und die entsprechenden Werte der Variablen *Start_1, Start_2, struct_typ* und *length*, mit denen jedes Paket beginnt, angegeben.

Die IO-Struktur Read Register

Das Senden dieser Struktur an das Web-IO veranlasst dieses, den Status der Inputs an das Anwenderprogramm zu senden. Das Paket besteht nur aus diesen vier WORDs. Diese Struktur wird nur vom Anwenderprogramm verwendet und das Web-IO reagiert immer mit dem Senden der Struktur *WriteRegister*.

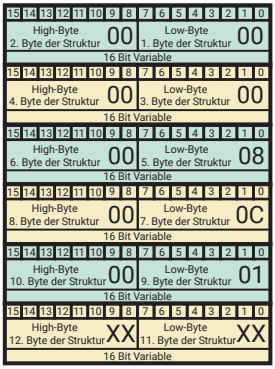


Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Die IO-Struktur WriteRegister

Diese Struktur dient zum Übertragen des Zustandes der Inputs oder Outputs beim Web-IO 12x Digital Input, 12x Digital Output. Sendet das Anwenderprogramm diese Struktur an das Web-IO, setzt das Web-IO die Outputs entsprechend des in *value* übergebenen Wertes.

Sendet das Web-IO diese Struktur an das Anwenderprogramm, hat *value* den dem Input-Status entsprechenden Wert.

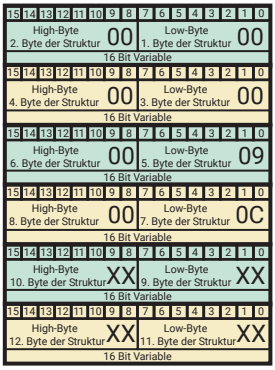


Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Amount	Integer Immer = 1
Value	Integer Binäre Entsprechung des Input-Status

Bei Web-IO mit Relaisausgängen, muss zwischen zwei Schaltvorgängen an einem Output ein Pause von min. 200ms eingehalten werden!

Die IO-Struktur SetBit

Diese Struktur ermöglicht das Setzen einzelner Outputs beim Web-IO. Wird z.B. nicht der gesamte Prozessstatus im Anwenderprogramm abgebildet, können einzelne Outputs gesetzt werden, ohne den Wert der anderen zu verändern. Die Bits 0..11 in *set_bits* und *value* korrespondieren mit den entsprechenden Outputs. Diese Struktur wird nur vom Anwenderprogramm verwendet.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Mask	Integer Bits die verändert werden sollen = 1
Value	Integer Binäre Entsprechung des Input-Status

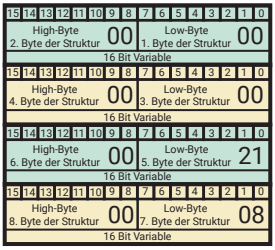
Beispiel.:

```
set_bits = 0x0124 / value = 0x0104
```

Output 2 und Output 8 (Zählweise Output0..1) werden auf ON gesetzt und Output 5 auf OFF. Alle anderen Outputs werden nicht verändert.

Die IO-Struktur RegisterRequest

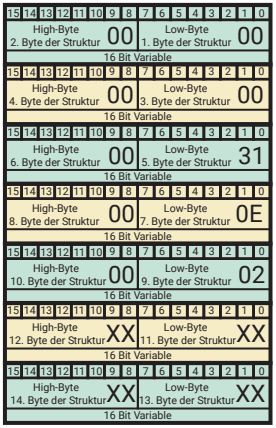
Diese Struktur sendet das Anwenderprogramm an das Web-IO, um den Inhalt von Inputs und Outputs im Überblick lesen zu können. Das Web-IO antwortet immer mit der E/A-Struktur *RegisterState*



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Die IO-Struktur RegisterState

Das Web-IO übermittelt mit dieser Struktur den Inhalt der Inputs und Outputs. Diese Struktur wird nur gesendet, wenn das Anwenderprogramm die Struktur *Register Request* an das Web-IO gesendet hat.

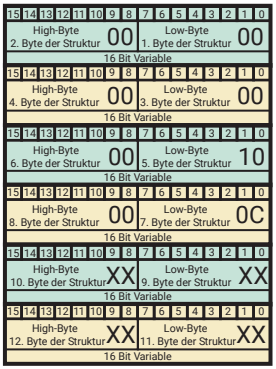


Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
DriverID	Integer Immer = 2
InputValue	Integer Binäre Entsprechung des Input-Status
OutputValue	Integer Binäre Entsprechung des Input-Status

Die IO-Struktur SendMode

Mit dieser Struktur werden die Triggerbedingungen festgelegt, mit denen das Web-IO den Status der Inputs an das Anwenderprogramm sendet. Prinzipiell gibt es drei Möglichkeiten, die jedoch alle miteinander kombinierbar sind

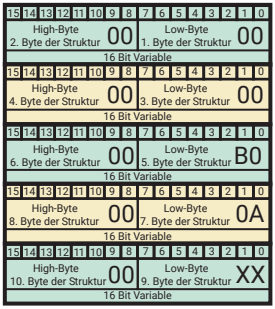
1. Das Anwenderprogramm pollt das Web-IO durch Senden der READ-Struktur.
2. Das Web-IO sendet die WriteRegister - Struktur mit dem Status der Inputs in einem konfigurierbarem Zeitintervall.
3. Das Web-IO sendet die WriteRegister - Struktur mit dem Status der Inputs nach Zustandsänderung der konfigurierten Inputs.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Mask	Integer Bits für Input Trigger = 1
Interval	Integer Intervall in 100ms für Übermittlung der Inputs

Die IO-Struktur ReadCounter

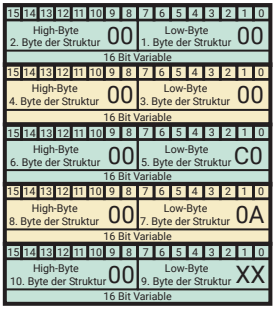
Diese Struktur sendet das Anwenderprogramm an das Web-IO, um den Zählerstand von einem bestimmten Input-Counter anzufordern. Um welchen Input es geht, wird in der Variablen *counter_index* übergeben. Das Web-IO antwortet immer mit der Struktur *Counter*.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Counterindex	Integer Nummer des Inputs

Die IO-Struktur ReadClearCounter

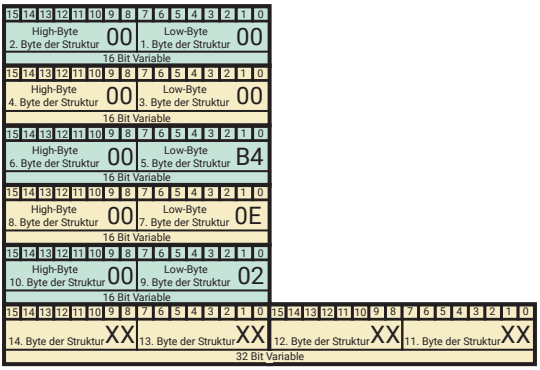
Diese Struktur sendet das Anwenderprogramm an das Web-IO, um den Zählerstand von einem bestimmten Input-Counter anzufordern und den Zähler danach sofort auf 0 zu setzen. Um welchen Input es geht, wird in der Variablen *counter_index* übergeben. Das Web-IO antwortet immer mit der Struktur *Counter*.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Counterindex	Integer Nummer des Inputs

Die IO-Struktur Counter

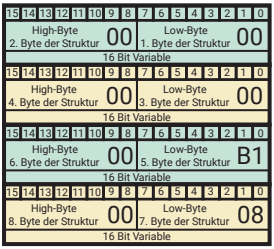
Das Web-IO übermittelt mit dieser Struktur den Zählerstand des in *counter_index* angegebenen Input-Counters.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
CounterIndex	Integer Nummer des Inputs
CounterValue	Long Zählerwert

Die IO-Struktur ReadAllCounter

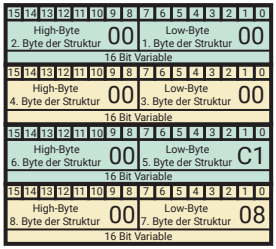
Diese Struktur sendet das Anwenderprogramm an das Web-IO, um die Zählerstände von allen Inputs in einem Datenpaket anzufordern. Das Web-IO antwortet immer mit der Struktur AllCounter.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Die IO-Struktur ReadClearAllCounter

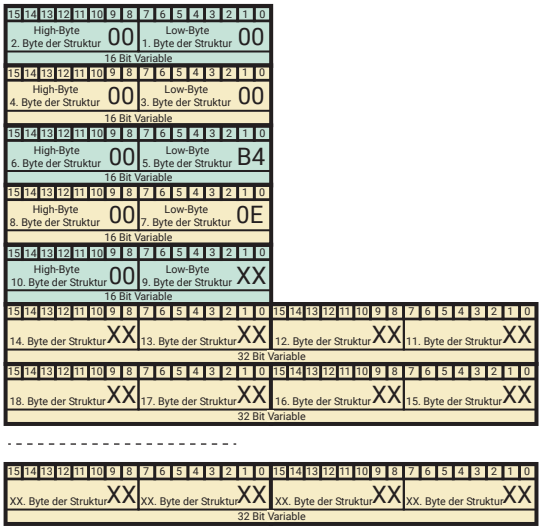
Diese Struktur sendet da Anwenderprogramm an das Web-IO, um die Zählerstände von allen Inputs in einem Datenpaket anzufordern und die Zähler danach sofort auf 0 zu setzen. Das Web-IO antwortet immer mit der Struktur *AllCounter*.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Die IO-Struktur AllCounte

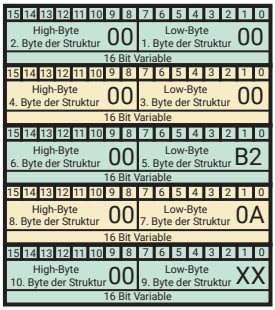
Das Web-IO übermittelt mit dieser Struktur die Zählerstände aller Inputs auf einmal.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
CounterNoOf	Integer Anzahl der Counter
CounterValue1	Long Zählerwert 1
CounterValue2	Long Zählerwert 2
CounterValue*n*	Long Zählerwert n

Die IO-Struktur ClearCounter

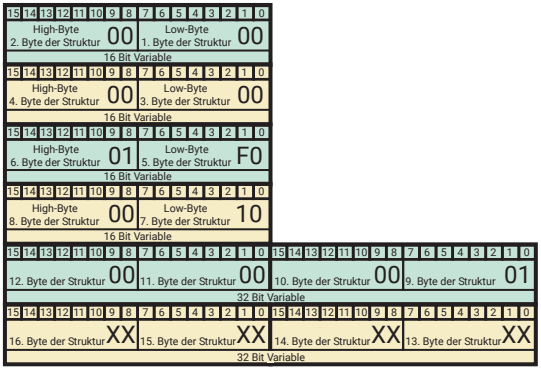
Diese Struktur sendet das Anwenderprogramm an das Web-IO, um den Zählerstand eines bestimmten Input-Counters auf 0 zurückzusetzen. Um welchen Input es geht, wird in der Variablen *counter_index* übergeben.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Counterindex	Integer Nummer des Inputs

Die IO-Struktur Options

Diese Struktur dient dazu, im Web-IO bestimmte Optionen zu setzen. Dazu stehen 32 Bit in der Variablen *options* zur Verfügung.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Version	Long Version z.Zt. 1
Options	Long Binäre Kodierung

In der vorliegenden Version des Web-IO wird nur das Bit 0 der Variablen *options* verwendet.

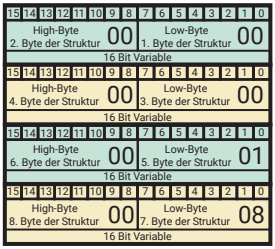
- Bit 0 = 1 //das Web-IO sendet die RegisterState Struktur zurück, wenn ein Output gesetzt wird.
- Bit 0 = 0 //das Web-IO gibt bei Setzen eines Output keine Rückmeldung.

Um in der Rückmeldung zuverlässig den Zustand nach Setzen des/der Output(s) zu bekommen, sollte zwischen zwei Output-verändernden Zugriffen min. eine Zeit 50ms liegen.

Analog-IO-Zugriff mit Binär-Struckturen

Die IO-Struktur ReadRegister

Das Senden dieser Struktur an das Web-IO veranlasst dieses, den Status der Ports an das Anwenderprogramm zu senden.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Das Paket besteht nur aus diesen vier WORDs. Diese Struktur wird nur vom Anwenderprogramm verwendet und das Web-IO reagiert immer mit dem Senden der Struktur AnalogRegisterState.

Die IO-Struktur AnalogRegisterState

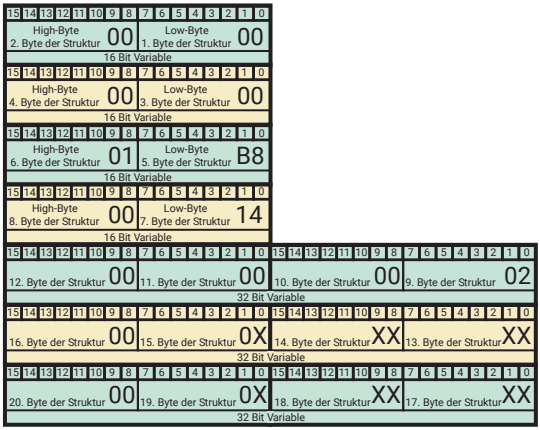
Das Web-IO Analog-In/Out übermittelt mit dieser Struktur den Status der beiden Ports. Diese Struktur wird gesendet, wenn das Anwenderprogramm die Struktur ReadRegister an das Web-IO gesendet hat, oder wenn mit dieser Struktur ein Ausgangswert gesetzt wurde.

Diese Struktur dient auch zur Übertragung der zu setzenden Output-Werte der Ports. Sendet das Anwenderprogramm diese Struktur an das Web-IO, setzt das Web-IO die Outputs entsprechend des in Port 1 und Port 2 übergebenen Wertes.

Hier wird der Wert nicht in der konfigurierten Einheit übermittelt, sondern immer in 1/100.000 vom Maximalwert. Ein Output-Wert von 15,4mA muss als 77000, bzw. 0x012CC8 übertragen werden .

Sendet das Web-IO diese Struktur an das Anwenderprogramm,

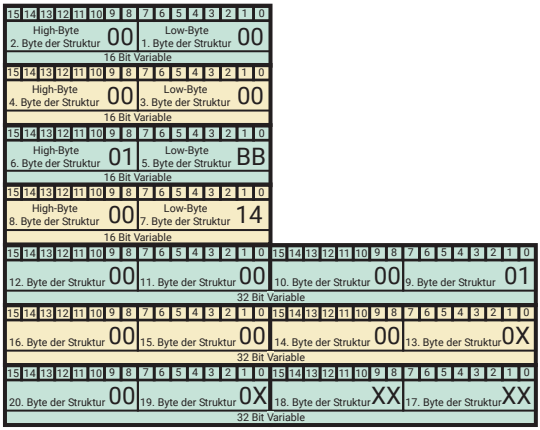
haben Port 1 und Port 2 den dem Eingangs-Status entsprechenden Wert.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Amount	Long Immer 2
Port1Value	Long Wert in 1/100.000
Port2Value	Long Wert in 1/100.000

Die IO-Struktur AnalogSingleRegister

Diese Struktur dient zum Übertragen des Output-Wertes eines einzelnen Ports beim Web-IO Analog-In/Out (Port 1 = 0, Port 2 = 1). Das Verfahren ist identisch wie bei AnalogRegisterState.

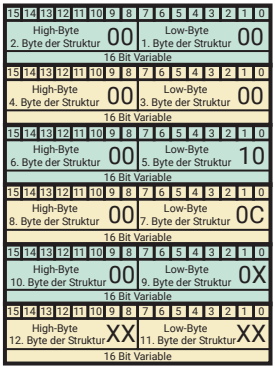


Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Amount	Long Immer 2
Channel	Long Kanal-Nr. (0 / 1)
PortValue	Long Wert in 1/100.000

Die IO-Struktur Send Mode

Mit dieser Struktur werden die Triggerbedingungen festgelegt, mit denen das Web-IO Analog-In/Out den Status der Ports an das Anwenderprogramm sendet. Der Trigger kann auf Statusänderungen beider Ports konfiguriert werden. Die jeweilige Hys-

terese für den Trigger muss in der Web-Kofiguration eingestellt werden.



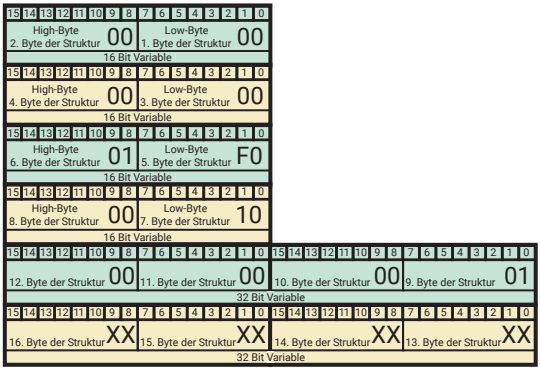
Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Mask	Integer Bits für Input Trigger = 1
Interval	Integer Intervall in 100ms für Übermittlung der Inputs

Als Variable Mask können folgende Kombinationen konfiguriert werden:

	Port 1	Port
0x0000	aus	aus
0x0001	an	aus
0x0002	aus	an
0x0003	an	an

Die IO-Struktur Options

Diese Struktur dient dazu, im Web-IO bestimmte Optionen zu setzen. Dazu stehen 32 Bit in der Variablen *options* zur Verfügung.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Version	Long Version z.Zt. 1
Options	Long Binäre Kodierung

In der vorliegenden Version des Web-IO wird nur das Bit 0 der Variablen *options* verwendet.

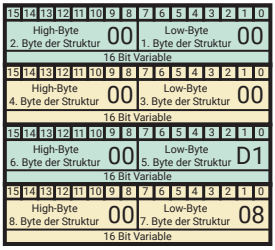
- Bit 0 = 1 //das Web-IO sendet die AnalogRegisterState Struktur zurück, wenn ein Output gesetzt wird.
- Bit 0 = 0 //das Web-IO gibt bei Setzen eines Output keine Rückmeldung.

Um in der Rückmeldung zuverlässig den Zustand nach Setzen des/der Output(s) zu bekommen, sollte zwischen zwei Output-verändernden Zugriffen min. eine Zeit 50ms liegen.

Gerätestatus über Binär-Strukturen

Die IO-Struktur ReadDiagnosis

Stellt das Web-IO einen Kommunikations- oder Systemfehler fest, wird dieser auf der HTML-Seite *diag* aufgelistet und kann über den Browser ausgelesen werden. Da sich das Fehlermanagement via Browser für programmgesteuerte Applikationen nicht immer anbietet, kann der Fehlerstatus des Web-IO über die Struktur *ReadDiagnosis* abgefragt werden

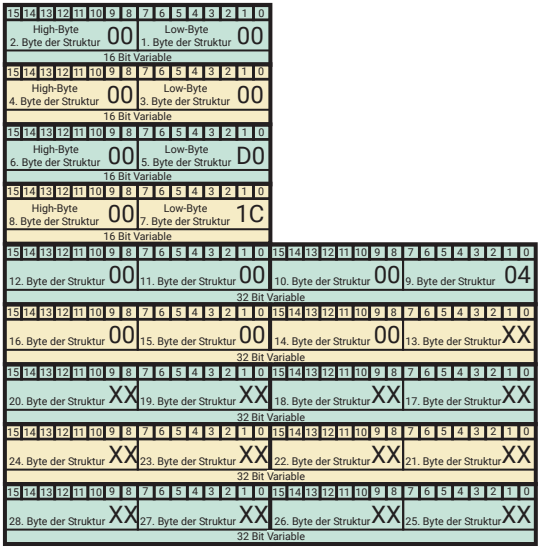


Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Als Antwort sendet das Web-IO eine Struktur vom Typ *Diagnosis*

Die IO-Struktur Diagnosis

Das Web-IO sendet die Struktur Diagnosis als Antwort auf die *ReadDiagnosis* Struktur.



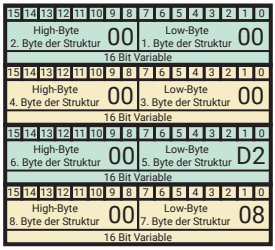
Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Version	Long z. Zt. 4
ErrorCount	Long aktuelle Fehlerzahl
Errorbits1	Long binäre Fehler-Kodierung
Errorbits2	Long binäre Fehler-Kodierung
Errorbits3	Long binäre Fehler-Kodierung

In der Variablen *ErrorCount* wird zurückgegeben, wieviele verschiedene Fehler zur Zeit im Fehlerspeicher sind. Insgesamt unterscheidet das Web-IO bis zu 92 verschiedene Fehlerzustände, wobei jedes gesetzte Bit der beiden Variablen *ErrorBits1* bis *ErrorBits3* für eine Fehlerart steht.

Die genaue textliche Beschreibung kann über den ASCII-Socket Port abgerufen werden. Eine nähere Beschreibung hierzu finden Sie im Kapitel Socketprogrammierung mit Kommandostrings.

Die IO-Struktur ClearDiagnosis

Mit dieser Struktur wird der Fehlerspeicher des Web-IO gelöscht.

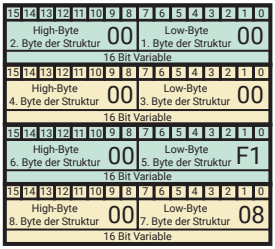


Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Geräteausstattung über Binär-Struktur

Die IO-Struktur InventoryRequest

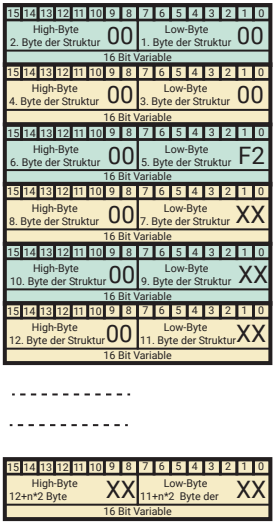
Um die IO-Ausstattung des Web-IO abzufragen wird die InventoryRequest-Struktur zum Web-IO gesendet.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes

Die IO-Struktur Inventor

Empfängt das Web-IO die Struktur InventoryRequest, antwortet darauf mit der Inventory-Struktur.



Start_1	Integer Immer = 0
Start_2	Integer Immer = 0
StructType	Integer Strukturtype
StructLength	Integer Länge in Bytes
Amount	Integer Anzahl der IO-Mode Einträge
IO-Mode	Integer High-Byte = IO-Kanalnummer Low-Byte = IO-Kanalmodus

In der Variablen *Amount* gibt das Web-IO an, wieviele *IO-Mode* einträge dir Struktur enthält.

Jeder Eintrag setzt sich aus der Kanalnummer als High Byte und dem verfügbaren IO-Modus als Low Byte zusammen.

Kann von einem IO-Kanal mehr als ein Wert oder Zustand abgerufen werden, gibt es für jeden Abrufbaren Parameter einen Eintrag (beim digitalen Input z.B. einen Eintrag für den Input-Zustand und einen weiteren für den Counter-Wert.

IO-Mode
Die IO-Modi sind durchnummeriert wie folgt:

1 analoger Input (4..20mA oder 0..10V)
2 digitaler Input
3 digitaler Output (Lese- und Schreibzugriff)
4 Counter
5 digitaler Output (nur Lesezugriff)
6 analoger Output

IO-Kanalnummer
Die IO-Kanalnummer entspricht der Indizierung der einzelnen Inputs oder Outputs. Abhängig vom IO-Type kann die Indizierung bei 0 (z.B. bei digitalen Web-IOs) oder 1 (z.B. bei analogen Web-IOs) beginnen