W&T

www.WuT.de

Programmierhandbuch

Eigene Anwendungen programmieren

Web-IO Digital 4.0

gültig für folgende Web-IO 4.0 Modelle:

#57737 Web-IO 4.0 Digital 2xIn, 2xOut

#57734 Web-IO 4.0 Digital 12xIn, 6xRelais

#57730 Web-IO 4.0 Digital 12xIn, 12xOut

#57731 Web-IO 4.0 Dig. 12xIn, 12xOut, 1xRS232

Release 10/2016

© 07/2016 by Wiesemann und Theis GmbH

Microsoft, MS-DOS, Windows, Winsock und Visual Basic sind eingetragene Warenzeichen der Microsoft Corporation

Irrtum und Änderung vorbehalten:

Da wir Fehler machen können, darf keine unserer Aussagen ungeprüft verwendet werden. Bitte melden Sie uns alle Ihnen bekannt gewordenen Irrtümer oder Missverständlichkeiten, damit wir diese so schnell wie möglich erkennen und beseitigen können.

Führen Sie Arbeiten an bzw. mit W&T Produkten nur aus, wenn sie hier beschrieben sind und Sie die Anleitung vollständig gelesen und verstanden haben. Eigenmächtiges Handeln kann Gefahren verursachen. Wir haften nicht für die Folgen eigenmächtigen Handelns. Fragen Sie im Zweifel lieber noch einmal bei uns bzw. bei Ihrem Händler nach!

Inhalt

Überblick 6
Zugriff aus der eigenen Anwendung 6

TCP/IP-Sockets - Kommandostrings 7
TCP-Server 10
UDP-Peer 11
Kommunikation zwischen Web-IO und Anwendung 11
Abfrage der Inputs 11
Abfrage der Outputs 14
Schalten der Outputs 14
Abfrage der Counter 16
Löschen der Counter 17
At hoc Lesen von Inputs, Outputs und Countern 19
Abfrage von Datum und Uhrzeit 19
Einstellen von Datum und Uhrzeit 19
Auslesen des Fehlerspeichers 20

TCP/IP-Sockets - Binary 25
TCP-Server 25
TCP-Client 26
UDP-Peer 26
BINARY - Die IO-Strukturen 27
Definition der IO-Strukturen 27
Passwort geschützter Zugang 29
Unautorisierter Zugriffsversuch 29
Die Struktur Authrequired 29
Die Login-Prozedur 29
Die Struktur KeyRequest 30
Die Struktur Key 30
Die Struktur Login 31
Die AuthOK Struktur 31
Die AuthFail Struktur 32
IO-Zugriffe mit Binär-Strukturen 32
Die IO-Struktur Read Register 32
Die IO-Struktur WriteRegister 33
Die IO-Struktur SetBit 33

Die IO-Struktur RegisterRequest 34
Die IO-Struktur RegisterState 35
Die IO-Struktur ReadCounter 36
Die IO-Struktur ReadClearCounter 36
Die IO-Struktur Counter 37
Die IO-Struktur ReadAllCounter 37
Die IO-Struktur ReadClearAllCounter 38
Die IO-Struktur AllCounte 38
Die IO-Struktur ClearCounter 39
Gerätestatus über Binär-Strukturen 40
Die IO-Struktur ReadDiagnosis 40
Die IO-Struktur Diagnosis 40
Die IO-Struktur ClearDiagnosis 41
Die IO-Struktur Options 42

HTTP-Request 43
Verwendung von W&T Tags 52

REST - Representational State Transfer 54
Lesender Zugriff 54
Verändernder Zugriff 57

Modbus TCP - standardisierter Zugriff 60
Modbus TCP Kommunikation 60
Function Code 0x01 Read Coils 63
Function Code 0x02 Read Discrete Inputs 64
Function Code 0x03 Read Holding Registers 65
Function Code 0x04 Read Input Registers 67
Function Code 0x05 Write Single Coil 67
Function Code 0x06 Write Single Register 69
Function Code 0x0F Write Multiple Coils 69
Function Code 0x10 Write Multiple Registers 72
Function Code 0x07 Read Exception State 73
Modbus-Adressbereiche für Web-IO Digital 75
Virtuelle Register für Browser-Interaktion .....
Modbus Speicheraufteilung .....
Bit-Bereich: .....
16- und 32-Bit-Bereich: .....

Überblick

Zugriff aus der eigenen Anwendung

Neben den zahlreichen standardisierten Zugriffsmöglichkeiten bietet das Web-IO auch die Option, es aus einer eigenen Anwendung anzusprechen.

Das kann über TCP/IP-Sockets aus den gängigen Hochsprachen erfolgen - es ist aber auch möglich, gängige Web-Techniken wie AJAX oder PHP zu nutzen, um mit dem Web-IO zu kommunizieren.

Für den individuellen Zugriff bietet das Web-IO mehrere Zugänge:

- Kommandostrings ASCII
- Binärstrukturen BINARY
- HTTP-Requests REST
- HTTP-Requests JavaScript/AJAX

Zahlreiche Programmierbeispiele in diversen Programmiersprachen finden Sie unter *http://wut.de* im Bereich *Web-IO Digital*. Unter *Weitere Informationen zu dieser Produktgruppe* folgen Sie dem Link *Tools* und dort dem Register *Programmierbeispiele allgemein.*

TCP/IP-Sockets - Kommandostrings

Durch den Austausch einfacher Kommandostrings können die Inputs und Counter gelesen bzw. die Outputs gesetzt werden.

Je nach Konfiguration arbeitet das Web-IO in diesem Modus als TCP-Server oder als UDP-Peer.

Der Datenaustausch zwischen Anwendung und Web-IO geschieht dabei mittels einfachen Kommandostrings.

Die Kommandostrings bestehen aus:

- der einleitenden Sequenz: `GET /`
- dem eigentlichen Kommando: `<command>`
- dem Trenner: `?`
- und einem oder mehreren Parametern:
  `<parameter1>&<...`

Die Reihenfolge der Parameter ist vorgegeben und kann nicht frei gewählt werden.

Alle Kommandostrings enden immer mit `&`

Beim Umgang mit den Kommandostrings muss die Groß-/Kleinschreibung beachtet werden.

Es darf kein CR LF (0x0d 0x0a) ans Zeilenende angefügt werden.

Die Antwort des Web-IO besteht aus einem Bezeichner, der anzeigt was übergeben wird und, mit Semikolon getrennt, den Werten, die ebenfalls mit Semikolon getrennt übergeben werden. Als Abschluss wird ein Null-Byte angehängt (0x00).

```
<bezeichner>;<wert>[;<wert2>;<wertx>]+(0x00)
```

Wenn das Web-IO als Server für TCP-Sockets konfiguriert ist, kann bei gesetzter Option *Header voranstellen* vor der eigent-

lichen Antwort die IP-Adresse und der Device-Name gesendet werden.

```
<ip-adresse>;<device-name>;<bezeichner>;<wert>[;<wert2>;<wertx>]+(0x00)
```

Darüber hinaus wird die Antwort bei gesetzter Option *Antworten HTTP konform* in einen HTTP-Rahmen eingefügt und erst dann versendet.

In der folgenden Liste finden Sie einen Überblick aller unterstützen Kommandos mit den zugehörigen Parametern:

Kommandos	Parameter	Beschreibung
GET /input**x**	?PW=**password**& An Stelle von **password,** muss das Administrator- oder Configpasswort eingesetzt werden. Wurde kein Passwort vergeben, wird "PW=&" angegeben. ***(gilt für alle Kommandos!)***	Anforderung des Inputstatus **x** kann ein Wert zwischen 0-11 sein und gibt den Input an. Die Rückmeldung des Web-IO ist ein String beginnend mit "input**x;**" gefolgt vom Input-Status: ON = Signal am Input und OFF = kein Signal am Input Wird **x** komplett weggelassen, gibt das Web-IO ein den Input-Signalen entsprechendes Bitmuster in hexadezimaler Schreibweise zurück.
GET /counter**x**	?PW=**password**&	Anforderung des Counterwertes **x** kann ein Wert zwischen 0-11 sein und gibt den Input an. Die Rückmeldung des Web-IO ist ein String beginnend mit "counter**x**;". Der Zählerstand des gewählten Counters wird in dezimaler Schreibweise angefügt.
GET /counter	?PW=**password**&	Anforderung aller Counter-Werte Die Rückmeldung des Web-IO ist ein String beginnend mit "counter;". Die Zählerstände werden mit Semikolon getrennt in dezimaler Schreibweise angefügt.
GET /output**x**	?PW=**password**&	Anforderung des Outputstatus **x** kann ein Wert zwischen 0-11 sein und gibt den Output an. Die Rückmeldung des Web-IO ist ein String beginnend mit "output**x;**" gefolgt vom Output-Status: ON = Signal am Output und OFF = kein Signal am Output Wird **x** komplett weggelassen, gibt das Web-IO ein den Output-Signalen entsprechendes Bitmuster in hexadezimaler Schreibweise zurück.
GET /outputaccess**x**	?PW=**password**& [Mask=XXXX&] State=**ON**/**OFF**/**YYYY**& [NA=ON&] ON: Output = 1, OFF: Output = 0, TOGGLE: Output wechselt Zustand XXXX: Hex-Wert zwischen 0000 und 0FFF entsprechend der Bits die gesetzt werden sollen YYYY: Hex-Wert zwischen 0000 und 0FFF entsprechend dem Ausgangsbitmuster.	Setzen eines oder mehrerer Outputs **x** kann ein Wert zwischen 0-11 sein und gibt den Output an, der gesetzt werden soll. Die Rückmeldung des Web-IO ist ein String beginnend mit "output;" gefolgt von einem den Output-Signalen entsprechenden Bitmuster in hexadezimaler Schreibweise. Die Angabe von Mask ist optional. Wird Mask nicht mitgesendet, gilt das outputaccess Kommando für alle Outputs. Mit NA=ON wird optional erreicht, dass vom Web-IO keine Antwort auf das outputaccess Kommando gesendet wird.
GET /counterclear**x**	?PW=**password**& [Set=**value**&] value: Zählervoreinstellung, Wert zwischen 0 und 2 Milliarden	Setzt den Zählerstand eines Counters. Wird der Parameter Set nicht mitgesendet, ist die Voreinstellung 0. **x** kann ein Wert zwischen 0-11 sein und gibt den Input an, dessen Counter zurückgesetzt werden soll. Die Rückmeldung des Web-IO ist ein String beginnend mit "counter**x**;". Der neue Zählerstand des gewählten Counters wird in dezimaler Schreibweise angefügt. Wird **x** nicht angegeben, werden alle Counter gesetzt.
GET /allout	?PW=**password**&	Sammelanforderung der Input, Output-Zustände und aller Counterwerte. Das Web-IO antwortet mit einem String folgenden Aufbaus: input;0xxx;output;0xxx;counter;n0;n1;n2,... 0xxx entspricht dabei dem Status der Inputs bzw. Outputs in hexadezimaler Schreibweise. n0,n1 usw. enthalten die Zählestände in dezimaler Darstellung

Kommandos	Parameter	Beschreibung
`GET /time`	`?PW=`**`password`**`&`	Gibt die systemzeit des Web-IO im Format: `DD.MM.YYYY,hh:mm:ss` zurück. D=Tag, M=Monat, Y=Jahr, h=Stunde, m=Minute, s=Sekunde
`GET /settime`	`?PW=`**`password`**`&` `time=`**`DD.MM.YYYY,hh:mm:ss`**`&`	Setzt die Systemzeit des Web-IO auf den mit **`time`** übergebenen Wert
`GET /diagnosis`	`?PW=`**`password`**`&`	Fordert den Status des Diagnosespeichers an. Das Web-IO gibt zurück: `diagnosis;0000;00000000;00000000,00000000` der vierstellige Wert gibt die Anzahl der gespeicherten Meldungen an. Bei den drei achtstelligen hexadezimalen Werten steht jedes gesetzte Bit stellvertretend für eine der 92 moglichen Meldungen.
`GET /diagnosis`**`x`**	`?PW=`**`password`**`&`	Mit **`x`** wird der Index für eine der aktuell gespeicherten Meldungen angegeben. Als Rückgabe sendet das Web-IO den entsprechenden Meldungstext. **`x`** darf nicht größer sein als die Anzahl der aktuell anliegenden Meldungen.
`GET /diaglist`**`x`**	`?PW=`**`password`**`&`	Gibt die Meldungen zu den einzelnen Meldungsbits zurück (max. 92)
`GET /diagclear`	`?PW=`**`password`**`&`	Löscht den Meldungsspeicher
`GET /errorclear`	`?PW=`**`password`**`&`	Löscht Load-Fehler und gibt die betroffenen Outputs wieder frei. Siehe Kapitel *Diagnose und Test*

TCP-Server

Um das Web-IO über ASCII-Sockets als TCP-Server anzusprechen, aktivieren Sie *TCP ASCII-Sockets* unter *Kommunikationswege >> Socket Zugriff*. Geben Sie an, auf welchem *TCP-Port* das Web-IO Verbindungen entgegennehmen soll. Das Web-IO kann zeitgleich bis zu acht TCP-Verbindungen über den angegebenen Port bereitstellen (wird auch der UDP-ASCII-Zugang genutzt sind es nur 7) jeder weitere Verbindungsversuch wird abgewiesen.

Empfängt das Web-IO innerhalb von 30 Sekunden nach Verbindungsaufbau kein gültiges Kommando, schließt es die Verbindung und gibt den Zugang wieder frei. In gleicher Weise verhält sich das Web-IO, wenn ein fehlerhaftes oder unbekanntes Kommando empfangen wird.

Um die Outputs über ASCII-Sockets schalten zu können, muss *Outputs für ASCII-Sockets freigeben* aktiviert werden.

UDP-Peer

Im Gegensatz zu TCP gibt es bei der Kommunikation über das UDP-Protokoll keine feste Verbindung zwischen dem Web-IO und dem anderen Kommunikationspartner. Der Austausch von Daten findet über Datagramme statt.

Um das Web-IO über ASCII-Sockets als UDP-Peer anzusprechen, aktivieren Sie *UDP ASCII-Sockets* unter *Kommunikationswege >> Socket Zugriff.* Geben Sie unter *Lokaler UDP Port* an, auf welchem *UDP-Port* das Web-IO UDP-Datagramme entgegennehmen soll.

Im Normalfall sollte als *Remote UDP Port* eine 0 eingetragen sein, damit das Web-IO Antwort-Datagramme an den UDP-Port zurücksendet, von dem die Anfrage kam. Alternativ kann unter *Remote UDP Port* ein fester Port vorgegeben werden, an das Web-IO alle Datagramme versendet.

Die UDP-Kommunikation findet immer im Polling-Betrieb statt. Das bedeutet, die Anwendung fordert mit Hilfe der Kommandostrings die gewünschten Werte an, bzw. setzt die Outputs.

Um die Outputs über UDP ASCII-Sockets schalten zu können, muss *Outputs für UDP-Sockets freigeben* aktiviert werden.

Kommunikation zwischen Web-IO und Anwendung

Abfrage der Inputs

Das Lesen der Inputs geschieht im Regelfall im Polling-Verfahren. Das bedeutet: die Client-Anwendung fordert mit Hilfe der Kommandostrings die gewünschten Werte an

Dabei kann entweder gezielt der Zustand eines bestimmten Inputs angefordert werden oder es erfolgt eine Anfrage für alle Inputs.

Soll nur ein Input abgefragt werden, wird dem Schlüsselwort `input` die Nummer des Inputs angefügt. Für die Abfrage aller

Inputs wird auf die Indizierung verzichtet.

Wenn das Web-IO durch ein Passwort geschützt ist, wird dieses mit `?` getrennt mit dem Statement `PW=` angehängt.

Der Kommandostring wird mit `&` beendet.

Wenn kein Passwort vergeben wurde folgt hinter `PW=` direkt das `&`.

Beispiel für die Abfrage von Input1 bei gesetztem Passwort "blau":

```
GET  /input1?PW=blau&
```

Das Web-IO sendet daraufhin zurück:

```
input1;ON+(0x00)
```

wenn der Input mit einem gültigen Signal angesteuert wird oder:

```
input1;OFF+(0x00)
```

wenn kein Signal anliegt. Alle Antworten des Web-IO enden mit einem NullByte (0x00).

Für die Abfrage aller Inputs sieht das Kommando so aus:

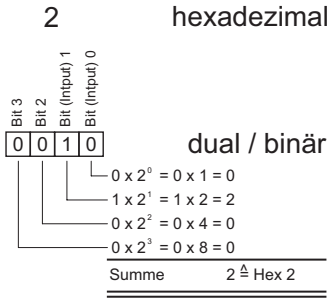
```
GET /input?PW=blau&
```

Das Web-IO antwortet mit dem Schlüsselwort `input` und einer je nach Web-IO-Type ein- oder vierstelligen Hexadezimalzahl. Die hexadezimale Zahl entspricht dem Bitmuster, das sich aus den gesetzten bzw. nicht gesetzten Inputs ergibt.

Beispiel 2fach Web-IO mit Input 1 = ON:

```
input;2+(0x00)
```

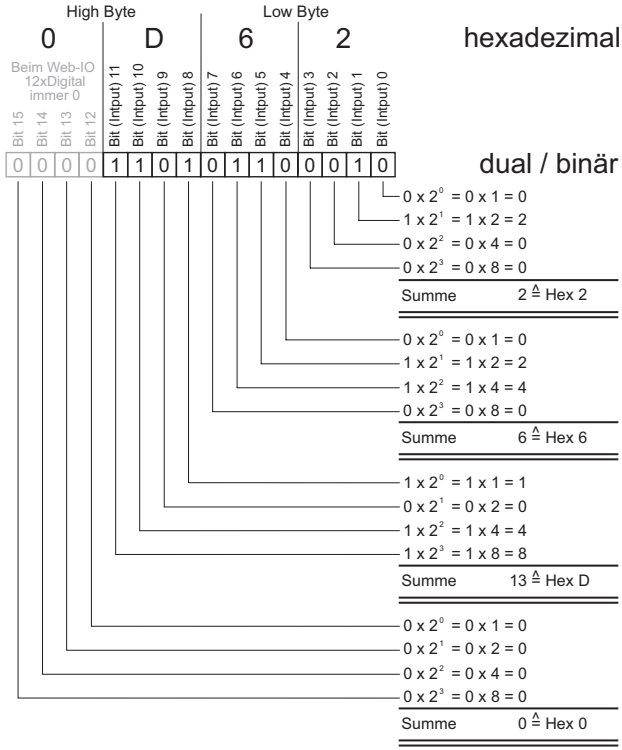
Der Wert ergibt sich folgendermaßen:



Beispiel 12fach Web-IO mit Inputs 1,5,6,8,10 und 11 = ON:

```
input;0D62 +(0x00)
```

Der Wert ergibt sich folgendermaßen:



Event-gesteuerte Benachrichtigung (nur TCP)

Einige Anwendungen machen es notwendig, dass Änderungen an bestimmten Inputs automatisch über die bestehende TCP-Verbindung gemeldet werden.

Ein Grund kann die Reduzierung der Datenlast auf dem Netzwerk sein. Werden die Inputs in einem zyklischen Intervall abgefragt, erzeugt das eine kontinuierliche Datenlast. Ferner besteht die Gefahr, dass Änderungen an den Inputs während zweier Zyklen nicht erkannt werden.

Damit das Web-IO von sich aus Signalveränderungen an den Inputs sendet, aktivieren Sie unter *Kommunikationwege >> Socket-Zugriff* die Entsprechenden *Input Trigger*.

Die Meldungen vom Web-IO haben den gleichen Aufbau wie die Antworten bei der Abfrage aller Inputs.

Abfrage der Outputs

der lesende Zugriff auf die Outputs erfolgt in der gleichen Weise wie die Abfrage der Inputs, nur das Schlüsselwort `input` wird durch `output` ersetzt.

Schalten der Outputs

Für das Schalten der Outputs wird das Kommando `outputaccess` verwendet.

Es können entweder ein einzelner Output, alle Outputs auf einmal oder eine Teilmenge der Outputs geschaltet werden.

Schalten eines Outputs

Wie beim lesenden Zugriff, wird mit einer Indizierung gearbeitet. Dem Schlüsselwort `outputaccess` wird die Nummer des Outputs angehängt, der geschaltet werden soll.

Zusätzlich wird mit dem Parameter `State` gearbeitet, für den es in diesem Fall drei mögliche Werte:

- `ON`

- `OFF`
- `TOGGLE` (Zustandswechsel)

Beispiel:

```
GET /outputaccess1?PW=blau&State=ON&
```

schaltet Output 1 auf ON.

Als Antwort sendet das Web-IO z.B.

```
output;0EAC +(0x00)
```

wobei der tatsächlich zurückgegebene Wert davon abhängt, in welchem Zustand die anderen Outputs vor dem Schalten waren.

	Bitweise Darstellung	Hexadezimal
Anfangszustand der Outputs:	0000 1110 1010 0100	output; 0EA4
Output 1:	1	State=ON
Zustand nach dem Schalten:	0000 1110 1010 0110	output; 0EAC

Gleichzeitiges Schalten aller Outputs

Um alle Outputs zeitgleich zu schalten, wird auf die Indizierung verzichtet und mit dem `State`-Parameter die hexadezimale Zahl übergeben, die dem gewünschten Bitmuster der Outputs entspricht.

Wenn z.B. die Outputs 1,6 und 11 auf ON und alle anderen auf OFF geschaltet werden sollen, sieht das Kommando so aus:

```
GET /outputaccess?PW=blau&State=0842&
```

als Antwort sendet das Web-IO

```
output;0842 +(0x00)
```

	Bitweise Darstellung	Hexadezimal
Anfangszustand der Outputs:	0000 1110 1010 0100	output; 0EA4
State:	0000 1000 0100 0010	State=0842
Zustand nach dem Schalten:	0000 1000 0100 0010	output; 0842

Gleichzeitiges Schalten einiger Outputs

Um nur eine Teilmenge der Outputs zu schalten, wird derselbe Kommandoaufbau wie beim Schalten aller Outputs verwendet. Es wird dem State Parameter allerdings der Parameter Mask vorangestellt. Der Parameter Mask legt fest, welche Outputs geschaltet werden und State legt fest in welchen Zustand.

Sollen z.B. die Outputs 1,6 und 11 auf ON geschaltet werden und die Outputs 5 und 10 auf OFF, sieht das Kommando so aus:

```
GET /outputaccess?PW=blau&Mask=0C62&State=0842&
```

als Antwort sendet das Web-IO

```
output;0AC6 +(0x00)
```

Wobei der tatsächlich zurückgegebene Wert davon abhängt, in welchem Zustand die anderen Outputs vor dem Schalten waren.

	Bitweise Darstellung	Hexadezimal
Anfangszustand der Outputs:	0000 1110 1010 0100	output; 0EA4
Mask:	0000 1100 0110 0010	Mask=0C62
State:	0000 1000 0100 0010	State=0842
Zustand nach dem Schalten:	0000 1010 1100 0110	output; 0AC6

Abfrage der Counter

Die Inputs des Web-IO beinhalten einen Zähler, der bei jeder steigenden Flanke, also einem Wechsel von OFF nach ON um eins hochgezählt wird.

Die Abfrage der Zähler erfolgt über das Kommando `counter`. Wie bei den Inputs und Outputs , können entweder die Werte aller Counter oder eines einzelnen Counters abgefragt werden.

Nach Erreichen eines Zählerstandes von 2147483648 erfolgt der Überlauf nach 0

Abfrage eines Counters

Hinter dem Schlüsselwort `counter` wird die Nummer des Inputs angegeben, dessen Counter abgerufen werden soll.

Beispiel für die Abfrage von Counter 0:

```
GET /counter0?PW=blau&
```

als Antwort sendet das Web-IO z.B.

```
counter;3974 +(0x00)
```

Der Zählerwert wird in dezimaler Schreibweise mit Semikolon getrennt hinter dem Schlüsselwort `counter` zurückgegeben.

Abfrage aller Counters

Wird das Kommando counter ohne die Nummer eines Inputs an das Web-IO gesendet, gibt das Web-IO die Zählerständer aller Inputs mit Semikolon getrennt zurück.

Beispiel für die Abfrage aller Counter eines Web-IO mit 12 Inputs:

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

als Antwort sendet das Web-IO z.B.

```
counter;3974;453;99;0;0;984;712;4;334;1076;0;6543 +(0x00)
```

```
Der Zählerwert wird in dezimaler Schreibweise mit Semikolon getrennt hinter dem
Schlüsselwort counter zurückgegeben.
```

Löschen der Counter

Mit dem Kommando `counterclear` können ein bestimmter oder alle Counter auf 0 oder bei bedarf auch auf einen anderen Wert gesetzt werden.

Löschen eines Counters

Beispiel für das Löschen von Counter 0:

```
GET /counterclear0?PW=blau&
```

als Antwort sendet das Web-IO

```
counter0;0 +(0x00)
```

Voreinstellen eines Counters

Um einen Counter auf einen bestimmten Wert voreinzustellen wird der optionale Parameter `Set=<value>&` angehängt

Beispiel für das setzen von Counter 0 auf den Wert 12345:

```
GET /counterclear0?PW=blau&Set=12345&
```

als Antwort sendet das Web-IO

```
counter0;12345 +(0x00)
```

Löschen aller Counter

Beispiel für das Löschen aller Counter bei einem Web-IO mit 12 Inputs:

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

als Antwort sendet das Web-IO

```
counter0;0;0;0;0;0;0;0;0;0;0;0;0 +(0x00)
```

Voreinstellen aller Counters

Um alle Counter auf einen bestimmten Wert voreinzustellen wird ebenfalls der optionale Parameter `Set=<value>&` angehängt. Es ist nicht möglich, in einem Kommando alle Counter auf jeweils individuelle Zählerstände voreinzustellen.

Beispiel für das setzen aller Counter auf den Wert 12345 bei einem Web-IO mit 12 Inputs:

```
GET /counterclear?PW=blau&Set=12345&
```

als Antwort sendet das Web-IO

```
counter0;12345;12345;12345;12345;12345;12345;
```

```
12345;12345;12345;12345;12345;12345 +(0x00)
```

At hoc Lesen von Inputs, Outputs und Countern

Mit dem Kommando `allout` kann mit nur einer Abfrage das komplette Prozessabbild des Web-IO abgerufen werden

Beispiel:

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

als Antwort sendet das Web-IO mit 12 Inputs und 12 Outputs z.B.

```
input;0C3B;output0842counter;3974;453;99;0;0;984;
712;4;334;1076;0;6543 +(0x00)
```

Abfrage von Datum und Uhrzeit

Mit dem Kommando `time` können das eingestellte Datum und die Uhrzeit des Web-IO abgefragt werden

Auf die Anfrage

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

antwortet das Web-IO z.B. mit

```
01.08.2016,15:14:59
```

Einstellen von Datum und Uhrzeit

Die Systemzeit des Web-IO kann mit dem Kommando `settime` eingestellt werden. Dazu werden über den Parameter `time` die gewünschten Einstellungen im Format

```
time=DD.MM.YYYY,hh:mm:ss
```

übergeben

Beispiel:

```
GET /settime?PW=&time= 01.08.2016,15:14:59
```

das Web-IO antwortet mit

```
01.08.2016,15:14:59
```

Auslesen des Fehlerspeichers

Erkennt das Web-IO bei der Abwicklung seiner Aufgaben einen oder mehrere Fehler, werden diese gespeichert und sind über den Navigationsbaum unter *Diagnose* einsehbar.

In einigen Anwendungen ist es gewünscht, das Fehlermanagement nicht manuell durch Ansehen einer Webseite abzuwickeln, sondern Fehler automatisiert in einem Programm auszuwerten.

Hierzu stellt das Web-IO einige Kommandostrings zur Verfügung, mit denen gleichermaßen per TCP oder UDP gearbeitet werden kann.

Erstellen einer Liste möglicher Fehlermeldungen

Das Web-IO kann maximal 127 verschiedene Fehlertypen verwalten. Die tatsächliche Anzahl kann innerhalb dieser Grenze je nach Firmware-Stand variieren.

Um einen Überblick über die möglichen Fehler zu bekommen, kann das Kommando

```
GET /diaglistx?PW=password&
```

an das Web-IO gesendet werden.

Für *x* wird ein Wert zwischen 0 und 127 eingesetzt. Anstelle von *password* setzen Sie das Administrator- oder Operator-Passwort ein.

Das Web-IO beantwortet die Anfrage mit der entsprechenden Fehlermeldung.

```
diaglistx;errortext
```

Beispiel:

Sendet die Anwendung zu Web-IO (Passwort = „wut“)

```
GET /diaglist2?PW=wut&
```

antwortet das Web-IO mit

```
diaglist2;Formatfehler in der DNS Anfrage
```

Mit einer *for next* Schleife können über das `diaglist` Kommando alle möglichen Fehlermeldungen ausgelesen und als statische Liste abgespeichert werden. Im Binary-Modus zum Beispiel kann nur ermittelt werden wie viele Fehler aktuell vorliegen. Zusätzlich werden die zugehörigen Fehlernummern übergeben. Mit Hilfe der statischen Fehlerliste kann dann ein lesbarer Fehler ausgegeben werden.

Direktes Auswerten der aktuellen Fehler

Um den aktuellen Fehlerstatus des Web-IO zu bekommen, wird das Kommando

```
GET /diagnosis?PW=password&
```

verwendet.

Das Web-IO gibt daraufhin zurück

```
diagnosis;iiii;aaaaaaaa;bbbbbbbb;cccccccc;dddddddd
```

mit `iiii` gibt das Web-IO die Anzahl der aktuell vorliegenden Fehler in vierstelliger, hexadezimaler Schreibweise zurück.

`aaaaaaaa;bbbbbbbb;cccccccc;dddddddd` können für normale Standardanwendungen außer Acht gelassen werden.

Für den Programmierprofi: es handelt sich bei `aaaaaaaa;bbbbbbbb;cccccccc;dddddddd` um vier 32Bit Werte in jeweils achtstelliger hexadezimaler Schreibweise. Die damit wiedergegebenen

92 Bit stehen stellvertretend für die 128 möglichen Fehlertypen. Mit Hilfe der statischen Fehlerliste (siehe `GET /diaglist...`) können die einzelnen Bits dekodiert werden. Dabei wird das niederwertigste Bit rechts im d Bereich angezeigt und das höchstwertige links im a Bereich.

Um die aktuellen Fehler als Fehlertext zu bekommen, stellt das Web-IO das Kommando

```
GET /diagnosis x?PW=&
```

zur Verfügung.

x ist hierbei der Index für den Fehler in dezimaler Schreibweise beginnend bei 1.

Die Antwort vom Web-IO sieht dann so aus

```
diagnosis x;errortext
```

Im Gegensatz zu den Fehlertexten die mit `diaglist` aus dem Web-IO gelesen werden, enthält der Fehlertext bei `diagnosis` ggf. auch dynamische Elemente wie z.B. IP-Adressen oder Portnummern.

Ein Beispiel:

Die Anwendung sendet:

```
GET /diagnosis?PW=wut&
```

Das Web-IO antwortet:

```
diagnosis;0005;00400040;008A0000;00000000
```

Die Anwendung sendet:

```
GET /diagnosis1?PW=wut&
```

Das Web-IO antwortet:

```
diagnosis1;Mail Server antwortet nicht.
```

Die Anwendung sendet:

```
GET /diagnosis2?PW=wut&
```

Das Web-IO antwortet:

```
diagnosis2;Das Versenden der Mail wurde abgebrochen und wird wiederholt.
```

Die Anwendung sendet:

```
GET /diagnosis3?PW=wut&
```

Das Web-IO antwortet:

```
diagnosis3;Ziel IP-Adresse unbekannt: 172.16.232.8.
```

Die Anwendung sendet:

```
GET /diagnosis4?PW=wut&
```

Das Web-IO antwortet:

```
diagnosis4;Watchdog Timer abgelaufen!
```

Die Anwendung sendet:

```
GET /diagnosis5?PW=wut&
```

Das Web-IO antwortet:

```
diagnosis5;TCP Client Alarm: Server nicht erreicht.
```

Löschen des Fehlerspeichers

Die Liste der zur Laufzeit aufgetretenen Fehler bleibt im Web-IO gespeichert, auch wenn der Fehler ggf. zum Zeitpunkt der Abfrage nicht mehr vorliegt.

Um den Fehlerspeicher zu löschen wird das Kommando

```
GET /diagclear?PW=password&
```

verwendet.

Das Web-IO antwortet mit

```
diagnosis;0000;00000000;00000000
```

wenn keine aktuellen Fehler mehr anliegen.

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-Zugriff* 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-Zugriff* 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:

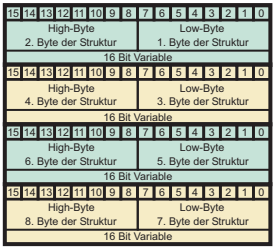
- Passwort-Anforderung und -Übergabe
- Lesen der Inputs
- Lesen der Inputs und Outputs
- Setzen der Outputs
- Lesen der Counter
- Zurücksetzen der Counter
- Parametrieren der zyklischen und automatischen Benachrichtigung bei Zustandsänderung

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	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) 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.

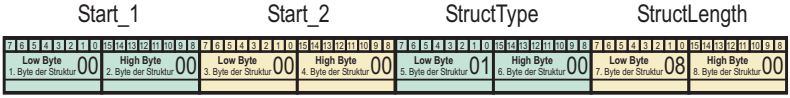
StructLength

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.

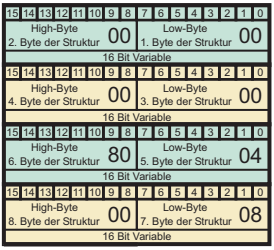
Wenn unter *Kommunikationswege >> Socket-Zugang > TCP/UDP-Sockets BINARY-Mode* im Bereich *BINARYx TCP-Server der* Punkt *Passwortgeschützter Zugriff* aktiviert ist, muss eine mehrstufige Login-Prozedur durchlaufen werden.

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

Die AuthRequired Struktur



Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes

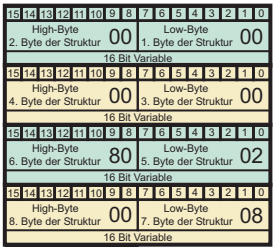
Wird die *AuthRequired* Struktur empfangen muss die Client-Anwendung mit der Login-Prozedur beginnen.

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 *Key-Request*

Die Struktur KeyRequest

Die KeyRequest Struktur

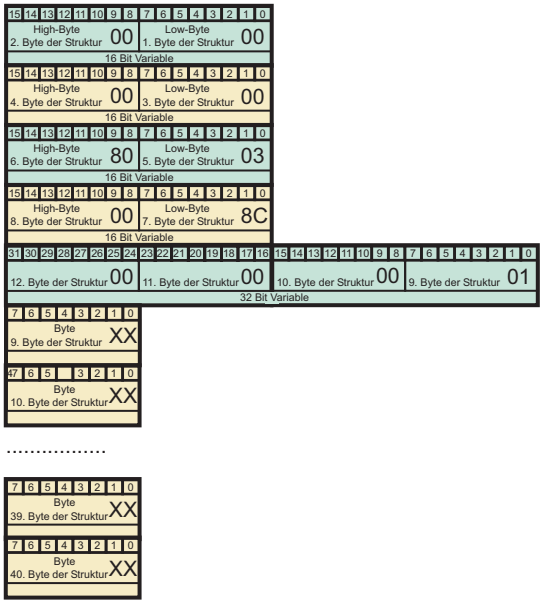


Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes

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

Die Key Struktur



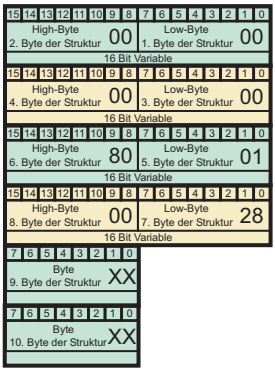
Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes
KeyLength	Longword (32 Bit) Anzahl der tatsächlich genutzten KeyBytes
KeyBytes	Byte-Array (8 Bit) immer 64 Byte auch wenn der tasächliche Key kürzer ist

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

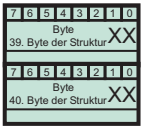
Die Struktur Login

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.

Die Login Struktur



.....

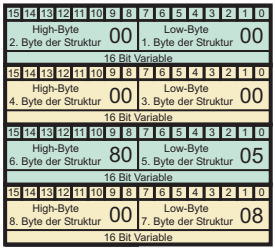


Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes
MD5-Hash	Byte-Array (8 Bit) 32 Byte MD5-Hash von Password + Key

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

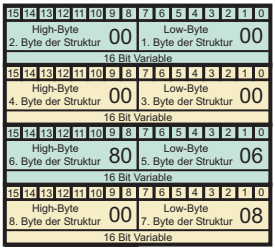
Die AuthOK Struktur



Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes

Die AuthFail Struktur

Die AuthFail Struktur



Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes

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

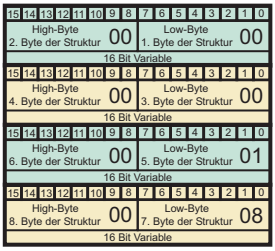
IO-Zugriffe mit Binär-Strukturen

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*.

Die ReadRegister Struktur



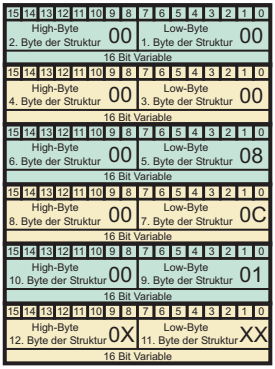
Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes

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.

Die WriteRegister Struktur



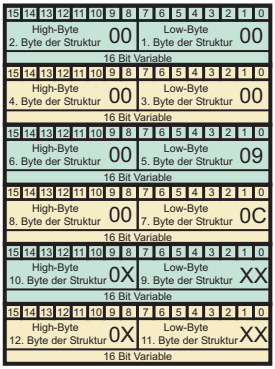
Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes
Amount	Word (16 Bit) Immer = 1
Value	Word (16 Bit) Binäre Entsprechung des Inputstatus

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.

Die SetBit Struktur



Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes
Mask	Word (16 Bit) Die Bits, die verändert werden sollen = 1
Value	Word (16 Bit) Binäre Entsprechung des Inputstatus

Beispiel.:

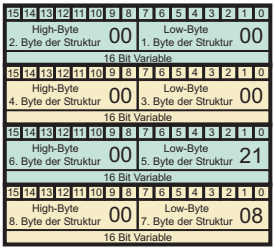
```
set_bits = 0x0124 / value = 0x0104
```

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*

Die RegisterRequest Struktur

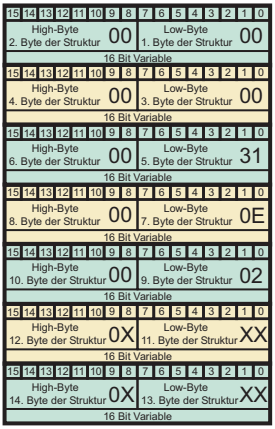


Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes

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.

Die RegisterState Struktur



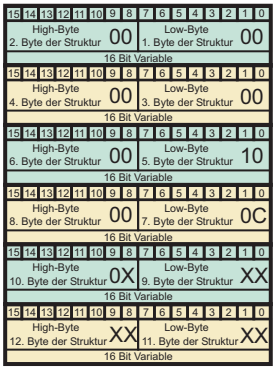
Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes
DriverID	Word (16 Bit) Immer = 2
InputValue	Word (16 Bit) Binäre Entsprechung des Inputstatus
OutputValue	Word (16 Bit) Binäre Entsprechung des Outputstatus

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.

Die SendMode Struktur

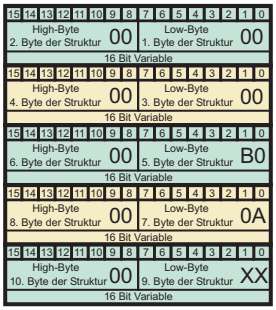


Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes
Mask	Word (16 Bit) Bits für Input Trigger = 1
Interval	Word (16 Bit) Intervall in 100ms für zyklische Input Status Übermittlung

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*.

Die ReadCounter Struktur



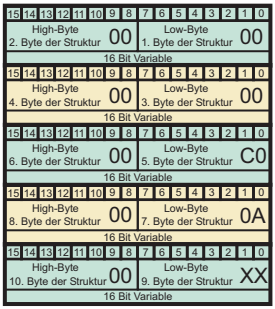
Start_1	Word Immer = 0
Start_2	Word Immer = 0
StructType	Word Strukturtype
StructLength	Word Länge in Bytes
CounterIndex	Word Nummer des Inputs, dessen Counter gelesen werden soll

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_in-*

dex übergeben. Das Web-IO antwortet immer mit der Struktur *Counter*.

Die ReadClearCounter Struktur

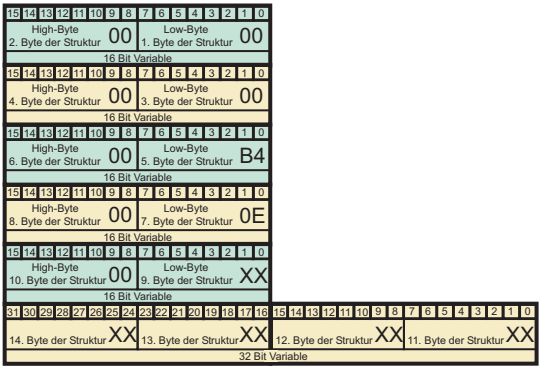


Start_1	Word Immer = 0
Start_2	Word Immer = 0
StructType	Word Strukturtype
StructLength	Word Länge in Bytes
CounterIndex	Word Nummer des Inputs, dessen Counter gelöscht werden soll

Die IO-Struktur Counter

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

Die Counter Struktur

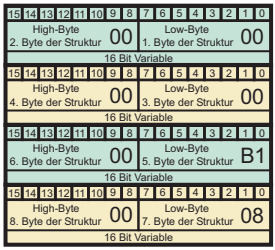


Start_1	Word Immer = 0
Start_2	Word Immer = 0
StructType	Word Strukturtype
StructLength	Word Länge in Bytes
CounterIndex	Word Nummer des Counters
CounterValue	Longword Inputzählerwert 1

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 All-Counter.

Die ReadAllCounter Struktur

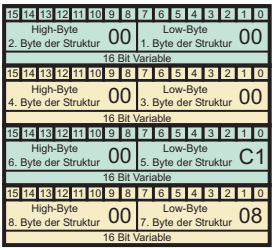


Start_1	Word Immer = 0
Start_2	Word Immer = 0
StructType	Word Strukturtype
StructLength	Word Länge in Bytes

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*.

Die ReadClearAllCounter Struktur

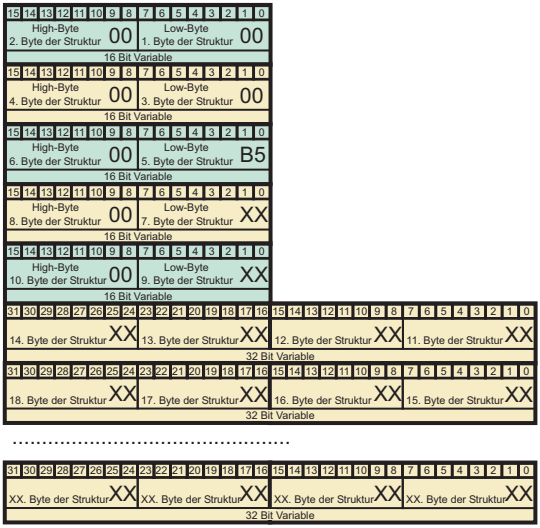


Start_1	Word Immer = 0
Start_2	Word Immer = 0
StructType	Word Strukturtype
StructLength	Word Länge in Bytes

Die IO-Struktur AllCounte

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

Die AllCounter Struktur

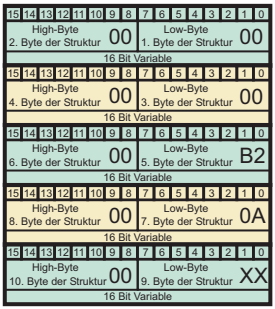


Start_1	Word Immer = 0
Start_2	Word Immer = 0
StructType	Word Strukturtype
StructLength	Word Länge in Bytes
CounterNoOf	Word Anzahl der Counter
CounterValue1	Longword Inputzählerwert 1
CounterValue2	Longword Inputzählerwert 2
CounterValuen	Longword Inputzählerwert n

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.

Die ClearCounter Struktur



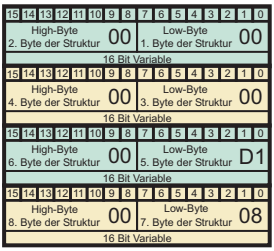
Start_1	Word Immer = 0
Start_2	Word Immer = 0
StructType	Word Strukturtype
StructLength	Word Länge in Bytes
CounterIndex	Word Nummer des Inputs, dessen Counter gelesen werden soll

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

Die ReadDiagnosis Struktur



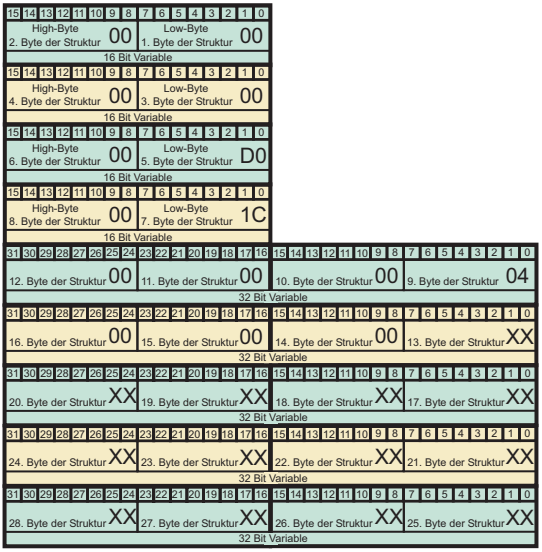
Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes

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.

Die Diagnosis Struktur



Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes
Version	Longword (32 Bit) z.Zt. V4
ErrorCount	Longword (32 Bit) aktuelle Fehlerzahl
ErrorBits1	Longword (32 Bit) binäre Fehler-Kodierung
ErrorBits2	Longword (32 Bit) binäre Fehler-Kodierung
ErrorBits3	Longword (32 Bit) binäre Fehler-Kodierung

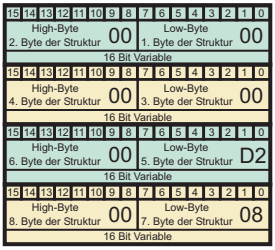
In der Variablen *ErrorCount* wird zurückgegeben, wieviele verschiedene Fehler zur Zeit im Fehlerspeicher sind. Insgesamt unterscheidet das Web-IO bis zu 92 verschiedene Fehlerzustände, wobei jedes gesetzte Bit der beiden Variablen *ErrorBits1* bis *ErrorBits3* für eine Fehlerart steht.

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.

Die ClearDiagnosis Struktur

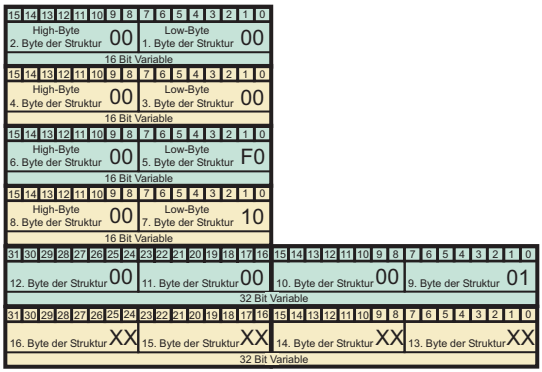


Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes

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.

Die Options Struktur



Start_1	Word (16 Bit) Immer = 0
Start_2	Word (16 Bit) Immer = 0
StructType	Word (16 Bit) Strukturtype
StructLength	Word (16 Bit) Länge in Bytes
Version	Longword (32 Bit) z.Zt. V4
Options	Longword (32 Bit) binäre Options-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.

HTTP-Request

Neben den klassischen Socket-Zugängen kann das Web-IO auch über den HTTP-Zugang direkt mittels HTTP-Requests angesprochen werden

Ab Werk ist dieser Zugang gesperrt und muss zunächst über den Menüzweig *Kommunikationswege >> HTTP/AJAX* aktiviert werden.

Darüber hinaus bietet Das Web-IO die Möglichkeit unter *Visualisierungen >> Meine Webseite,* eine eigene, selbst programmierte Webseite ins Gerät hochzuladen und zu speichern. Die eigene Webseite kann über den Menübaum unter *Meine Webseite* bzw. dem von Ihnen vergebenen Namen aufgerufen werden. Es ist aber auch ein direkter Aufruf über die URL möglich:

```
http://<ip-adresse des Web-IO>/user
```

Die folgende Liste zeigt alle unterstützten HTTP-Requests

Weitere Details zum Zugriff mit Web-Techniken wie AJAX oder PHP finden Sie auf den folgenden Seiten

HTTP-Request	Parameter	Beschreibung
input**x**	?PW=**password**& An Stelle von **password,** muss das Administrator- oder Configpasswort eingesetzt werden. Wurde kein Passwort vergeben, wird "PW=&" angegeben. ***(gilt für alle Kommandos!)***	Anforderung des Inputstatus **x** kann ein Wert zwischen 0-11 sein und gibt den Input an. Die Rückmeldung des Web-IO ist ein String beginnend mit "input**x;**" gefolgt vom Input-Status: ON = Signal am Input und OFF = kein Signal am Input Wird **x** komplett weggelassen, gibt das Web-IO ein den Input-Signalen entsprechendes Bitmuster in hexadezimaler Schreibweise zurück.
counter**x**	?PW=**password**&	Anforderung des Counterwertes **x** kann ein Wert zwischen 0-11 sein und gibt den Input an. Die Rückmeldung des Web-IO ist ein String beginnend mit "counter**x;**". Der Zählerstand des gewählten Counters wird in dezimaler Schreibweise angefügt.
counter	?PW=**password**&	Anforderung aller Counter-Werte Die Rückmeldung des Web-IO ist ein String beginnend mit "counter;". Die Zählerstände werden mit Semikolon getrennt in dezimaler Schreibweise angefügt.
output**x**	?PW=**password**&	Anforderung des Outputstatus **x** kann ein Wert zwischen 0-11 sein und gibt den Output an. Die Rückmeldung des Web-IO ist ein String beginnend mit "output**x;**" gefolgt vom Output-Status: ON = Signal am Output und OFF = kein Signal am Output Wird **x** komplett weggelassen, gibt das Web-IO ein den Output-Signalen entsprechendes Bitmuster in hexadezimaler Schreibweise zurück.
outputaccess**x**	?PW=**password**& [Mask=XXXX&] State=**ON**/**OFF**/**YYYY**& ON: Output = 1, OFF: Output = 0, TOGGLE: Output wechselt Zustand XXXX: Hex-Wert zwischen 0000 und 0FFF entsprechend der Bits die gesetzt werden sollen YYYY: Hex-Wert zwischen 0000 und 0FFF entsprechend dem Ausgangsbitmuster.	Setzen eines oder mehrerer Outputs **x** kann ein Wert zwischen 0-11 sein und gibt den Output an, der gesetzt werden soll. Die Rückmeldung des Web-IO ist ein String beginnend mit "output;" gefolgt von einem den Output-Signalen entsprechenden Bitmuster in hexadezimaler Schreibweise. Die Angabe von Mask ist optional. Wird Mask nicht mitgesendet, gilt das outputaccess Kommando für alle Outputs.
counterclear**x**	?PW=**password**& [Set=**value**&] value: Zählervoreinstellung, Wert zwischen 0 und 2 Milliarden	Setzt den Zählerstand eines Counters. Wird der Parameter Set nicht mitgesendet, ist die Voreinstellung 0. **x** kann ein Wert zwischen 0-11 sein und gibt den Input an, dessen Counter zurückgesetzt werden soll. Die Rückmeldung des Web-IO ist ein String beginnend mit "counter**x**;". Der neue Zählerstand des gewählten Counters wird in dezimaler Schreibweise angefügt. Wird **x** nicht angegeben, werden alle Counter gesetzt.
allout	?PW=**password**&	Sammelanforderung der Input, Output-Zustände und aller Counterwerte. Das Web-IO antwortet mit einem String folgenden Aufbaus: input;0xxx;output;0xxx;counter;n0;n1;n2,... 0xxx entspricht dabei dem Status der Inputs bzw. Outputs in hexadezimaler Schreibweise. n0,n1 usw. enthalten die Zählestände in dezimaler Darstellung

HTTP-Request	Parameter	Beschreibung
`time`	`?PW=`**`password`**`&`	Gibt die systemzeit des Web-IO im Format: `DD.MM.YYYY,hh:mm:ss` zurück. D=Tag, M=Monat, Y=Jahr, h=Stunde, m=Minute, s=Sekunde
`settime`	`?PW=`**`password`**`&` `time=`**`DD.MM.YYYY,hh:mm:ss`**`&`	Setzt die Systemzeit des Web-IO auf den mit **`time`** übergebenen Wert
`diagnosis`	`?PW=`**`password`**`&`	Fordert den Status des Diagnosespeichers an. Das Web-IO gibt zurück: `diagnosis;0000;00000000;00000000,00000000` der vierstellige Wert gibt die Anzahl der gespeicherten Meldungen an. Bei den drei achtstelligen hexadezimalen Werten steht jedes gesetzte Bit stellvertretend für eine der 92 moglichen Meldungen.
`diagnosis`**`x`**	`?PW=`**`password`**`&`	Mit **`x`** wird der Index für eine der aktuell gespeicherten Meldungen angegeben. Als Rückgabe sendet das Web-IO den entsprechenden Meldungstext. **x** darf nicht größer sein als die Anzahl der aktuell anliegenden Meldungen.
`diaglist`**`x`**	`?PW=`**`password`**`&`	Gibt die Meldungen zu den einzelnen Meldungsbits zurück (max. 92)
`diagclear`	`?PW=`**`password`**`&`	Löscht den Meldungsspeicher
`errorclear`	`?PW=`**`password`**`&`	Löscht Load-Fehler und gibt die betroffenen Outputs wieder frei. Siehe Kapitel *Diagnose und Test*
`modbusreg`	`?PW=`**`password`**`&`	Gibt bei aktivierter Modbus-Unterstützung die 128 Byte virtuellen Speicher zurück, die vom Modbus-Master frei beschrieben werden können: `modbus;<1.Byte>;<2.Byte>;.....;<128.Byte>` Es werden alle 128 Byte mit Semikolon getrennt als Hexadezimalzahl übergeben.

Ein Beispiel für die Verwendung von HTTP-Requests

Das folgende Beispiel zeigt, wie mit Hilfe von JavaScript und HTTP-Requests (AJAX-Technik) eine selbstaktualisierende Webseite für das Web-IO aufgebaut werden kann. Hier zunächst der reine HTML-Anteil der Webseite, der quasi als Anzeigegerüst für die AJAX-Technik dient:

```
<!DOCTYPE html PUBLIC «-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
  <title>Web-IO Digital, User</title>
  <style type="text/css">
    * { font-family:arial; }
    table { font-size:14px; }
    .borderLeft { border-left:1px solid #000000; }
    .button { font-size:9px; width:40px; }
    .ce { text-align:center; }
    .description { font-size:18px; text-align:center; }
```

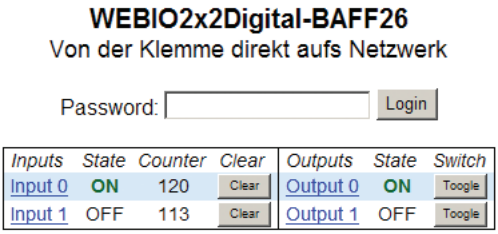
```
    .info { font-size:10px; text-align:center; }
    .italic { font-style:italic; }
    .name { font-size:20px; font-weight:bold; text-align:center }
    .sideSpace { padding-left:5px; padding-right:5px; }
    .table { background-color:#d6e8ff; border-collapse:collapse;
          border:1px solid #000000; }
    .whiteBack { background-color:#ffffff; }
  </style>
  <script language="JavaScript" type="text/javascript">
```

hier steht eigentlich der JavaScript-Teil der Seite, der im Folgenden näher beschrieben wird.

```
  </script>
  </head>
<body onload="CommandLoop();">
<div class="name"><w&t_tags=device_name></div>
<div class="description"><w&t_tags=device_text></div>
<br>
  <form>
    <div align="center">
            <span id="pt"">Password: </span>
        <input id="pw" type="password" name="tfPassword" maxlength="31"
size="20"">
            <input  id="lb" type="button" value="Login"
onclick="setPassword()">
       </div>
  </form>
<table align="center" class="table">
  <tr class="italic whiteBack">
    <td class="sideSpace">Inputs</td>
    <td class="sideSpace">State</td>
    <td class="ce sideSpace20">Counter</td>
    <td class="ce">Clear</td>
    <td class="borderLeft sideSpace">Outputs</td>
    <td class="sideSpace">State</td>
    <td class="ce">Switch</td>
  </tr>
  <tr>
    <td class="sideSpace"><w&t_tags=input0></td>
    <td class="ce" id="input0">-</td>
```

```
    <td class="ce" id="counter0">-</td>
    <td class="ce sidespace">
      <input class="button" onclick="clearCounter(0);" type="button"
value="Clear">
    </td>
    <td class="borderLeft sideSpace"><w&t_tags=output0></td>
    <td class="ce" id="output0">-</td>
    <td class="sideSpace">
      <input class="button" onclick="setOutput(0);" type="button" value="Toogle">
    </td>
  </tr>
  <tr class="whiteBack">
    <td class="sideSpace"><w&t_tags=input1></td>
    <td class="ce" id="input1">-</td>
    <td class="ce" id="counter1">-</td>
    <td class="ce sidespace">
      <input class="button" onclick="clearCounter(1);" type="button"
value="Clear">
    </td>
    <td class="borderLeft sideSpace"><w&t_tags=outpu1></td>
    <td class="ce" id="output1">-</td>
    <td class="sideSpace">
      <input class="button" onclick="setOutput(1);" type="button" value="Toogle">
    </td>
  </tr>
</table>
</body>
</html>
```

Im Browser sieht das so aus:



Die nachfolgend gezeigten Funktionen stützen sich auf zwei

Grundtechniken:

- Identifikation und nachträgliche Veränderung eines HTML-Objektes durch eindeutige ID
- Kommunikation mit dem Server (Web-IO) nach Laden der Web-Seite durch HTTP-Requests

Das zur Web-Seite gehörende JavaScript macht sich diese beiden Techniken zunutze.

Zunächst wird mit `maxi` und `maxo` festgelegt, wieviele Inputs und Outputs unterstützt werden sollen. Es wird eine Variable `applicationsstep` für die verschiedenen Programmschritte definiert. Als Abfrageintervall werden 500ms festgelegt und es wird eine Variable für das Passwort angelegt.

```
var maxi = 2;
var maxo = 2;
var applicationstep = 0;
var interval = 500;
var iopassword = ,';
```

Die Funktion `HextoInt` erlaubt die Umrechnung von hexadezimalen Zahlen in Ganzzahlenwerte (wird benötigt, wenn mit mehr als 2 IOs gearbeitet wird, da die Zustandsübergabe dann hexadezimal erfolgt).

```
function HexToInt(HexStr)
{ var TempVal;
  var HexVal=0;
  for( var i=0; i<HexStr.length;i++)
  { if (HexStr.charCodeAt(i) > 57)
    { TempVal = HexStr.charCodeAt(i) - 55;
    }
    else
    { TempVal = HexStr.charCodeAt(i) - 48;
    }
    HexVal=HexVal+TempVal*Math.pow(16, HexStr.length-i-1);
  }
  return HexVal;
}
```

Die Funktion `CommandLoop` setzt die HTTP-Requests zusammen und übergibt sie an die `DataRequest` Funktion.

```
function CommandLoop()
{ var commandstring ='';
  applicationstep++;
  switch(applicationstep)
  { case 1:
        commandstring = 'input?PW=' + iopassword + '&';
    break;
    case 2:
      commandstring = 'output?PW=' + iopassword + '&';
    break;
    case 3:
      commandstring = 'counter?PW=' + iopassword + '&';
      applicationstep = 0;
    break;
  }
  DataRequest(commandstring);
  maintimer = setTimeout(«CommandLoop()", interval);
}
```

Die Funktion `DataRequest` ist das Herzstück dieses JavaScriptes. Sie nimmt den HTTP-Request entgegen und sendet ihn an den Server (das Web-IO). Die Funktion `DataRequest` nimmt aber auch die Antwort des Web-IO entgegen und übergibt sie an die Funktion `updateDisplay`.

```
  function DataRequest(SendString)
  { var xmlHttp;
    if( window.ActiveXObject )       // Internet Explorer
    { xmlHttp = new ActiveXObject( «Microsoft.XMLHTTP" );
    }
    else if(window.XMLHttpRequest )  // Mozilla, Opera und Safari
    { xmlHttp = new XMLHttpRequest();
    }
    if (xmlHttp)
    { xmlHttp.onreadystatechange = function()
      { if (xmlHttp.readyState == 4)
        { if (xmlHttp.status == 200)
          { if (xmlHttp.responseText.length > 0)
```

```
            { updateDisplay(xmlHttp.responseText);
            }
            xmlHttp=null;
          }
        }
      }
      xmlHttp.open(«GET", SendString, true);
      xmlHttp.setRequestHeader(«Connection", «close");
      xmlHttp.setRequestHeader(«If-Modified-Since", «Thu, 1 Jan 1970 00:00:00
GMT");
      xmlHttp.send(null);
    }
  }
```

Die Funktion `updateDisplay` wertet die Antwort des Web-IO aus und passt die Anzeige im Browser entsprechend an. Dazu wird geprüft, ob sich die Antwort des Web-IO auf Inputs, Outputs oder Counter bezieht. Über `document.getElementById(ID)` identifiziert JavaScript die zu ändernden Objekte und passt deren Eigenschaften dem aktuellen IO-Status an.

```
function updateDisplay(ReceiveStr)
{ var HexVal;
  var state;
  var ReceiveData = ReceiveStr.split(';')
  // Display Intput state
  if (ReceiveData[ReceiveData.length - 2].substring(0, 1) == 'i')
  { HexVal = HexToInt(ReceiveData[ReceiveData.length - 1]);
    for (var i = 0; i < 2; i++)
    { state = false;
      if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
      { state = true;
      }
      document.getElementById('input'+i).firstChild.data =( !state ) ?
'OFF':'ON';
      document.getElementById('input'+i).style.color
                              = ( !state ) ? '#000000' : '#006600';
      document.getElementById('input'+i).style.fontWeight
                                = ( !state ) ? 'normal' : 'bold';
    }
  }
```

```
  // Display Output state
  if (ReceiveData[ReceiveData.length - 2].substring(0, 1) == 'o')
  { HexVal = HexToInt(ReceiveData[ReceiveData.length - 1]);
    for (var i = 0; i < 2; i++)
    { state = false;
      if ((HexVal & Math.pow(2, i)) == Math.pow(2, i))
      { state = true;
      }
      document.getElementById('output'+i).firstChild.data = ( !state ) ? 'OFF' :
'ON';
      document.getElementById('output'+i).style.color
                                = ( !state ) ? '#000000' : '#006600';
      document.getElementById('output'+i).style.fontWeight
                                  = ( !state ) ? 'normal' : 'bold';
    }
  }
  //Display Counter
  if (ReceiveData.length - maxi - 1 >= 0)
  { if (ReceiveData[ReceiveData.length - maxi - 1].substring(0, 1) == 'c')
    { for (var i = 0; i < maxi; i++)
      { document.getElementById('counter' + i).innerHTML
                           = ReceiveData[ReceiveData.length - maxi + i]
      }
    }
  }
  //Display cleared Counter
  if (ReceiveData[ReceiveData.length - 2].substring(0, 1) == 'c')
  { document.getElementById('counter'
  + ReceiveData[ReceiveData.length - 2].substring(7, ReceiveData[ReceiveData.
length -      2].length)).innerHTML = ReceiveData[ReceiveData.length - 1];
  }
}
```

Die Funktion `setOutput` sendet das entsprechende Kommando zum toggeln des gewählten Output an die Funktion `DataRequest`.

```
function setOutput(iNr)
{ var commandstring = 'outputaccess'+iNr+'?PW='+iopassword+'&State=TOGGLE&';
  DataRequest(commandstring);
}
```

Die Funktion `clearCounter` sendet das entsprechende Kommando zum Löschen des Zählerstandes des gewählten Input an die Funktion `DataRequest`.

```
function clearCounter(iNr)
{  DataRequest('counterclear'+iNr+'?PW='+iopassword+'&');
}
```

Die Funktion `setPassword` übernimmt das eingegebene Passwort und schreibt es in die Variable `iopassword`, die Bestandteil der Kommandostrings ist.

```
function setPassword()
{  iopassword = document.getElementById('pw').value;
   document.getElementById('pw').value = '';
}
```

Die auf HTTP-Requests gestützte AJAX-Technik kann nur eingesetzt werden, wenn die Webseite aus dem Web-IO geladen wird, das auch die Requests empfangen soll.

Die hier gezeigten HTTP-Requests lassen sich gleichermaßen mit anderen Web-Techniken wie z.B. PHP nutzen.

Verwendung von W&T Tags

Die Beschriftung von Inputs, Outputs und einigem mehr kann im Web-IO kundenspezifisch angepasst werden. Siehe hierzu im Menübaum des Web-IO *Grundeinstellungen >> Inputs/Outputs* bzw. *Grundeinstellungen >> Sprache*.

Eine entsprechende Anzeige im Browser wird durch die Verwendung von W&T-Tags erreicht. W&T-Tags sind Platzhalter, die vom Web-IO bei der Übergabe der Webseite an den Browser durch die hinterlegten Bezeichnungen ersetzt werden. So kann erreicht werden, das ein und die gleiche Webseite in verschiedenen Web-IOs individuell anders aussieht.

Diese Tags setzen sich aus `<w&t_tags=` und dem eigentlichen Funktionsaufruf zusammen.

```
<w&t_tags=time>
```

zum Beispiel zeigt im Browser die aktuelle Systemzeit nebst Datum des Web-IO an.

Folgende W&T-Tags stehen zur Verfügung:

W&T-Tag	Beschreibung/Funktion
<w&t_tags=device_name>	Fügt den für das Web-IO vergebenen Namen in die Webseite ein.
<w&t_tags=device_text>	Fügt die für das Web-IO hinterlegte Beschreibung in die Webseite ein.
<w&t_tags=location>	Fügt den für Location hinterlegten Eintrag in die Webseite ein.
<w&t_tags=contact>	Fügt den für Contact hinterlegten Eintrag in die Webseite ein.
<w&t_tags=input*x*>	Fügt die für Input Nr. *x* festgelegte Bezeichnung ein.*x* kann sich dabei zwischen 0-11 bewegen und gibt an auf welchen Input sich der Aufruf bezieht. Das gilt auch für die im Folgenden beschriebenen Output- und Counter-Aufrufe.
<w&t_tags=i*x*>	Gibt den Zustand (ON/OFF) des x entsprechenden Inputs wieder.
<w&t_tags=c*x*>	Fügt den Zählerstand des Counters für Input *x* in die Webseite ein.
<w&t_tags=output*x*>	Fügt die für Output Nr. *x* festgelegte Bezeichnung ein
<w&t_tags=o*x*>	Gibt den Zustand (ON/OFF) des *x* entsprechenden Outputs wieder. Bei Login mit Config- oder Administratorrechten, ist die Zustandsangabe mit einem Hyperlink ausgestattet. Durch Klick auf diesen Link wechselt der entsprechende Output seinen Zustand und die Webseite wird aktualisiert
<w&t_tags=time>	Fügt die Systemzeit und das Datum des Web-IO in die Webseite ein

REST - Representational State Transfer

Mit REST (Representational State Transfer) stellen die Web-IO einen weiteren, web-basierenden Kommunikationsweg zur Verfügung.

Die Kommunikation erfolgt über Web-IO spezifische HTTP-Requests über den unter *Grundeinstellungen » Netzwerk » Browser-Zugang* eingetragenen HTTP- bzw. HTTPS-Port.

Um via REST Daten austauschen zu können, muss der Zugriff zunächst über *Kommunikationswege » Rest* aktiviert werden.

Wenn der REST-Zugang gegen unberechtigten Zugang geschützt werden soll, haben Sie die Möglichkeit, die Digest-Authentifizierung zu aktivieren. Die Requests müssen dann als User „admin“ mit dem Administratorpasswort oder als User „operator“ mit dem Benutzerpasswort erfolgen.

Es kann darüber hinaus festgelegt werden, ob die Outputs über REST geschaltet werden dürfen.

Lesender Zugriff
Für lesende Zugriffe verwendet REST das HTTP-GET-Kommando.

Dabei unterstützt das Web-IO für Antworten auf REST-Anfragen drei Formate:

- JSON
- XML
- Text

In welchem Format geantwortet wird, kann über die Anfrage bestimmt werden. Mit

```
http://<ip-adresse>/rest/json
```

kann z.B. das gesamte Prozessabbild des Web-IO im JSON-Format abgerufen werden. Die Antwort sieht dann so aus:

```
{
  «info» :
  {
    «request» : « / rest / json»,
    «time» : «2016 - 09 - 09,
    09 : 42 : 54»,
    «ip» : «10.40.22.227»,
    «devicename» : «WEBIO - CAFE27»
  },
  «iostate» :
  {
    «input» : [
      {
        «number» : 0,
        «state» : 0
      },
      {
        «number» : 1,
        «state» : 0
      }
    ],
    «output» : [
      {
        «number» : 0,
        «state» : 0
      },
      {
        «number» : 1,
        «state» : 0
      }
    ],
    «counter» : [
      {
        «number» : 0,
        «state» : 0
      },
```

```
      {
        «number» : 1,
        «state» : 0
      }
    ]
  },
  «system» :
  {
    «time» :
    {
      «time» : «2016 - 09 - 09,
      09 : 42 : 54»
    },
    «diagnosis» : [
      {
        «time» : «06.09.2016 09 : 42 : 54»,
        «msg» : «Gerätestatus : OK»
      }
    ],
    «diagarchive» : [
      {
        «time» : «06.09.2016 09 : 42 : 54»,
        «msg» : «Gerätestatus : OK»
      }
    ]
  }
}
```

Um nur einzelne Bereiche oder Punkte abzufragen, kann die Anfrage detaillierter formuliert werden:

```
http://<ip-adresse>/rest/json/iostate/input
```

veranlasst das Web-IO dazu, den Status aller Inputs zurückzugeben:

```
{
  «iostate» :
  {
    «input» : [
      {
```

```
        «number» : 0,
        «state» : 0
      },
      {
        «number» : 1,
        «state» : 0
      }
    ]
  }
}
```

Mit

```
http://<ip-adresse>/rest/json/iostate/input/0
```

kann gezielt der Zustand von Input 0 abgefragt werden

```
{
  «iostate» :
  {
    «input» : [
      {
        «number» : 0,
        «state» : 0
      }
    ]
  }
}
```

Verändernder Zugriff

Bei Zugriffen, die den Schaltzustand der Outputs verändern oder die Counter löschen, wird mit POST gearbeitet.

Um z.B. Output 1 auf ON zu setzen, wird ein POST auf folgende URL gesendet:

```
http://<ip-adresse>/rest/json/iostate/output/1
```

Dabei werden die folgenden Parameter übergeben:

```
PW=<passwort>
Set=ON
```

Das Web-IO antwortet mit

```
{
  «iostate» :
  {
    «output» : [
      {
        «number» : 1,
        «state» : 1
      }
    ]
  }
}
```

Über die gleiche URL kann kann der Output mit dem Parameter Set=OFF ausgeschaltet werden.

Das Löschen von z.B. Counter1 erfolgt über über ein POST auf folgende URL:

```
http://<ip-adresse>/rest/json/iostate/counterclear/1
```

dabei wird folgender Parameter übergeben:

```
PW=<passwort>
```

Das Web-IO antwortet mit

```
{
  «iostate» :
  {
    «counter» : [
      {
        «number» : 1,
        «state» : 0
      }
    ]
  }
}
```

Um die Antworten in einem der anderen Formate zu erhalten, muss einfach das Schlüsselwort `json` gegen `xml` oder `text` ersetzt werden.

Modbus TCP - standardisierter Zugriff

Modbus TCP ist eine Softwareschnittstelle zum adressbasierten Zugriff auf Prozessdaten. Die W&T Web-IOs arbeiten bei entsprechender Konfiguration als Modbus-Server, die in aller Regel von Modbus-Steuerungen (Clients bzw. Master) kontrolliert werden können.

Programmierer, die den Modbus-Zugang in eigenen Applikationen nutzen möchten, finden hier eine detaillierte Beschreibung zu den vom Web-IO unterstützten Register und Funktionscodes.

Über den Modbus TCP Zugang können die Zustände von Outputs, Inputs und Countern gelesen werden. Auch Status- und Systemabfragen sind möglich. Darüber hinaus lassen sich die Outputs über Modbus TCP schalten und die Counter löschen.

Als Besonderheit bietet das Web-IO 64 16-Bit-Register, die frei beschreibbar sind und auf die auch vom Browser aus zugegriffen werden kann.

Unter *Kommunikationswege >> Modbus-TCP* kann der Modbus-Zugang des Web-IO aktiviert werden. Von Hause aus ist der lokale TCP-Port, wie für Modbus-TCP üblich auf 502 voreingestellt.

Outputs die über Modbus-TCP gesteuert werden sollen, können über *Outputs für Modbus/TCP freigeben* aktiviert werden.

Modbus TCP Kommunikation

Modbus TCP ist ein Master/Slave Verfahren, bei dem ein Modbus-Master einen Request zum Slave (Web-IO) sendet und der Slave (Web-IO) mit einen Reply antwortet.

Der Datenaustausch zwischen Modbus-Client und Web-IO erfolgt über Modbus TCP Datenpakete.

Das Web-IO verfügt über einen Speicher, aus dem der Mod-

bus-Master lesen bzw. in den er schreiben kann. Dieser Speicher ist in Bereiche aufgeteilt, die bestimmte Eigenschaften des Web-IO repräsentieren.

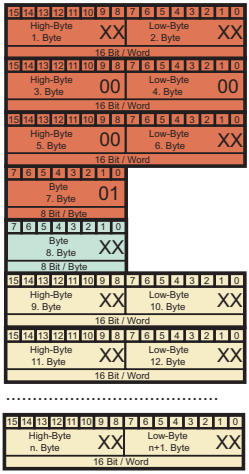
Es gibt Bereiche für den Status der Inputs, Outputs und Counter. Andere Bereiche geben den Systemstatus wieder oder betreffen die Alarme.

Eine Besonderheit des Web-IO ist ein Speicherbereich, der auf der einen Seite per Modbus-Zugriff frei beschreib- und lesbar ist und auf der anderen Seite über das Web-Interface abgerufen werden kann. Bei entsprechender Webseiten-Programmierung kann ein Modbus-System so Prozessdaten im Browser visualisieren.

Eine Liste mit der genauen Speicheraufteilung folgt im weiteren Verlauf dieses Kapitels.

Die Modbus Datenpakete bestehen immer aus einem Header, dem Function Code, der Start Address und weiteren Parametern bzw. Registern.

Aufbau von Modbus TCP Datenpaketen



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Start Address	16 Bit Adresse der zu lesenden/schreibenden Register
Weitere Parameter oder Register	8 Bit, 16 Bit oder 32 Bit

Transaction Identifier

Dient dazu, die Antwort des Web-IO der Anfrage des Clients zuzuordnen. Der Client zählt dazu normalerweise die ID mit jeder

neuen Datensendung um 1 hoch. Das Web-IO gibt immer den empfangenen Wert 1:1 zurück.

Protocol Identifier
Für die Kommunikation mit dem Web-IO ohne Bedeutung und immer 0x0000

Length
Anzahl der Bytes, die nach *Length* gesendet werden (gesamte gesendete Bytes - 6).

Unit identifier
Bei Modbus TCP immer 0x01

Function Code
Über den *Function Code* wird festgelegt, wie auf den Modbus-Speicher des Web-IO zugegriffen werden soll:

- bitweise
- registerweise (16-bitweise)

und welchen Zweck der Request hat:

- 0x01 Read Coils - lesen einzelner Bits
- 0x02 Read Discret Inputs - lesen einzelner Bits
- 0x03 Read Holding Registers- lesen mehrerer Register
- 0x04 Read Input Register - lesen mehrerer Eingangs Reg.
- 0x05 Write Single Coil - Schreiben eines Bits
- 0x06 Write Single Register - Schreiben nur eines Registers
- 0x07 Read Exception State - Fehlerstatus lesen
- 0x0F Write Multiple Coils - schreiben mehrerer Bits
- 0x10 Write Multiple Register - schreiben mehrerer Register

Start Address
Über *Start Address* wird festgelegt, auf welchen Bereich des Modbus-Speichers im Web-IO zugegriffen werden soll. Damit wird gleichzeitig festgelegt, auf welche Eigenschaft (Inputs, Outputs, Counter, Alarme .....) zugegriffen wird.

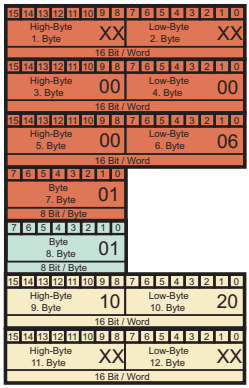
Die beiden Parameter *Function Code* und *Start Address* bestimmen, was der gesendete Request beim Web-IO auslösen soll.

Weitere Parameter und Register

Je nach verwendetem *Function Code* können weitere Parameter bzw. Register folgen.

Function Code 0x01 Read Coils

Der *Function Code* 0x01 ist zum binären Lesen des Status der Web-IO Outputs vorgesehen.



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Start Address	16 Bit Adresse der zu lesenden/schreibenden Register
Quantity of Outputs	16 Bit Anzahl der Bits die gelesen werden sollen

Start Address

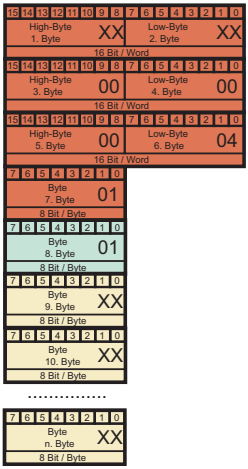
gibt an, ab welcher Speicherstelle Bits (Outputs) gelesen werden sollen.

Quantity of Outputs

Hier wird die Anzahl der zu lesenden Bits (Outputs) übergeben.

Über das Senden eines Datenpaketes mit FC 0x01 können ein oder mehrere boolsche Werte (Output-Zustände also 0/1 bzw. ON/OFF) abgefragt werden.

Die Antwort des Web-IO sieht so aus:



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Byte Count	8 Bit Anzahl der zurückgeegebenen Bytes (pro angefangene 8 Bit je ein Byte)
Output State	8 Bit Die angefragten Bits werden beginnend beim 10.Byte Bit 0 übergeben

Bytes Count

enthält die Anzahl der übertragenen Bytes, welche die angefragten Bits beinhalten. Pro angefangene 8 Bit wird ein Byte gesendet. Das bedeutet: ab 9 Bit werden 2 Bytes gesendet.

Output State

Anzahl an Bytes, wie in *Byte Count* übergeben. Das erste State-Byte (10. Byte) enthält beginnend bei Bit 0 die ersten 8 angefragten Bits (Outputs). Wurden weniger als 8 Bits angefragt, werden die ungenutzten Bits mit 0 übergeben. Bei mehr als 8 Bit geht es mit dem 11. Byte weiter.

Beispiel: Lesen des Status der Outputs 0 und 1

Output 0 ist OFF, Output 1 ist ON

Start Address wird auf 0x1020 gesetzt, *Quantity of Outputs* auf 0x02. (Die Outputs sind ab Adresse 0x1020 abrufbar.)

Das Web-IO antwortet mit *Byte Count* = 1, ein *Output State* = 0x02.

Function Code 0x02 Read Discrete Inputs

Der *Function Code* 0x02 ist zum binären Lesen des Status der Web-IO Inputs vorgesehen.

Der Paketaufbau von Anfrage und Antwort mit *Function Code* 0x02 ist identisch mit dem von Function Code 0x01.

Beispiel: Lesen des Status der Inputs 0 und 1

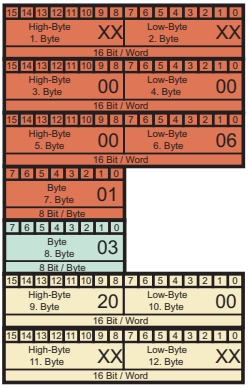
Input 0 ist OFF, Input1 ist ON

Start Address wird auf 1000 gesetzt,*Quantity of Inputs* auf 2. (Die Inputs sind ab Adresse 0x1000 abrufbar.)

Das Web-IO antwortet mit *Byte Count* = 1, *Input State* = 0x02.

Function Code 0x03 Read Holding Registers

Der *Function Code* 0x03 ist zum Lesen von mehreren Registern 16-Bit Werten) vorgesehen. Über FC 0x03 können beim Web-IO je nach verwendeter *Start Address* die Werte von Inputs, Outputs, Counter, usw. abgefragt werden.



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Start Address	16 Bit Adresse der zu lesenden/schreibenden Register
Number of Registers	16 Bit Anzahl der Register die gelesen werden sollen

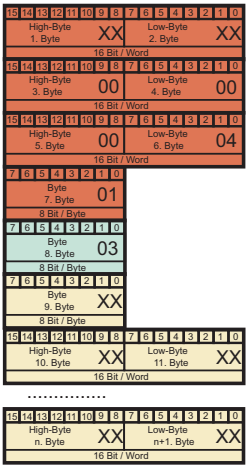
Start Address

gibt an, ab welcher Speicherstelle Register gelesen werden sollen.

Number of Registers

Hier wird die Anzahl der zu lesenden Register übergeben.

Das Web-IO antwortet mit folgendem Paket:



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Byte Count	8 Bit Anzahl der zurückgeegebenen Register-Bytes
Register Value	16 Bit Die angefragten Register werden beginnend beim 10.Byte übergeben

Bytes Count

enthält die Anzahl der als Register übertragenen Bytes (2 Bytes pro 16-Bit Register).

Register Value

Ein oder mehrere 16-Bit Registerwerte. Das erste 16-Bit Register beginnt mit dem High-Byte an der Position des 10. Bytes.

Je nach Start Address (ab 0x5000) werden für ein angefragtes 32-Bit Register zwei 16-Bit Werte - also 4 Bytes - übergeben. Auch in diesem Fall beginnt der Wert mit dem höchsten Byte an Position des 10. Byte und das erste Low-Byte liegt auf Position des 13. Byte des Datenpaketes.

Beispiel:
Lesen des Status der Outputs eines Web-IO 2xDigital

Output 0 ist OFF, Output1 ist ON

Start Address wird auf 0x2002 gesetzt,

Number of Registers auf 0x01

(Die Outputs sind als Register ab Adresse 0x2002 abrufbar.)

Das Web-IO antwortet mit

Byte Count = 0x02,

Register Value = 0x0002.

Function Code 0x04 Read Input Registers

Der *Function Code* 0x04 ist speziell zum Lesen des Status der Web-IO Inputs als 16-Bit Register vorgesehen.

Der Paketaufbau von Anfrage und Antwort mit Function Code 0x04 ist identisch mit dem von Function Code 0x03.

Beispiel:
Lesen des Status der Inputs eines Web-IO 2xDigital

Input 0 ist OFF, Input1 ist ON

Start Address wird auf 0x2000 gesetzt,

Number of Registers auf 0x01

(Die Inputs sind als Register ab Adresse 0x2002 abrufbar.)

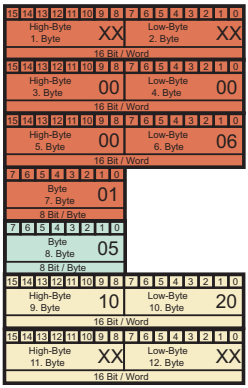
Das Web-IO antwortet mit

Byte Count = 0x02

Register Value = 0x0002

Function Code 0x05 Write Single Coil

Der *Function Code* 0x05 ist zum Setzen eines einzelnen Outputs vorgesehen.



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Output Address	16 Bit Adresse des zu schreibenden Registers
Output Value	16 Bit Wert der in das Register geschrieben werden soll

Output Address

Über *Output Address* wird indirekt festgelegt, welcher Output geschaltet werden soll. Beginnend bei 0x1020 hat jeder Output eine eigene Adresse.

Output Value

Hier wird festgelegt, ob ein Output auf ON oder OFF geschaltet werden soll:

ON = 0xFF00
OFF = 0x0000

Das Web-IO antwortet mit einem exakt gleich aufgebauten Datenpaket.

Beispiel:
Output1 eines Web-IO soll auf ON geschaltet werden

Output Address wird auf 0x1021 gesetzt,

Output Value auf 0xFF00

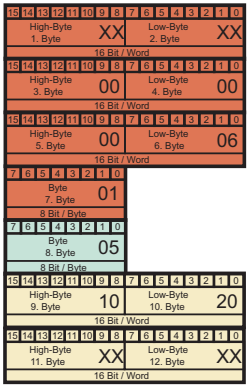
Das Web-IO antwortet mit

Output Address = 1021

Output Value = 0xFF00

Function Code 0x06 Write Single Register

Der *Function Code* 0x06 arbeitet identisch zum Function Code 0x05 und ist zum Setzen eines beliebigen Registers vorgesehen.



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Register Address	16 Bit Adresse des zu schreibenden Registers
Register Value	16 Bit Wert der in das Register geschrieben werden soll

Register Address

Über *Register Address* wird festgelegt, in welche Speicherstelle geschrieben werden soll.

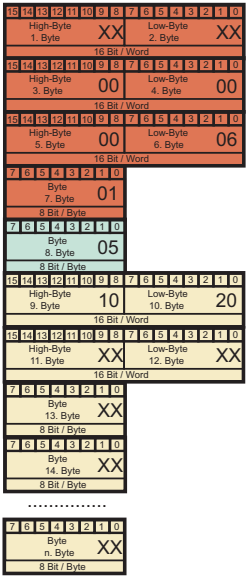
Register Value

ist ein 16-Bit Registerwert, der in den Modbus-Speicher des Web-IO geschrieben wird.

Das Web-IO antwortet mit einem exakt gleich aufgebauten Datenpaket.

Function Code 0x0F Write Multiple Coils

Der Function Code 0x0F ist für das bitweise Setzen von Outputs vorgesehen.



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Start Address	16 Bit Adresse des ersten zu setzenden Outputs
Quantity of Outputs	16 Bit Anzahl der zu setzenden Outputs
Byte Count	8 Bit Anzahl der Bytes nach ByteCount (pro angefangene 8 Bit je ein Byte)
Output State	8 Bit Die Output-Bits werden beginnend beim 14.Byte Bit 0 übergeben

Start Address

gibt an, ab welcher Speicherstelle Output-Bits geschrieben werden sollen.

Quantity of Outputs

Hier wird die Anzahl der zu schreibenden Bits übergeben.

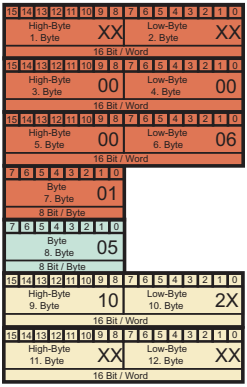
Bytes Count

enthält die Anzahl der übertragenen Bytes welche die zu setzenden Output-Bits beinhalten. Pro angefangene 8 Bit wird ein Byte gesendet. Das bedeutet ab 9 Bit werden 2 Bytes gesendet.

Output State

Anzahl an Bytes, wie in *Byte Count* übergeben. Das erste Byte (14. Byte) enthält beginnend bei Bit 0 die ersten 8 zu setzenden Output-Bits (Outputs). Wurden weniger als 8 Bits (Outputs) angegeben, werden die ungenutzten Bits mit 0 übergeben. Bei mehr als 8 Bits geht es mit dem 15. Byte weiter.

Das Web-IO antwortet mit einem Datenpaket mit folgendem Aufbau:



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Start Address	16 Bit Adresse des ersten zu setzenden Outputs
Quantity of Outputs	16 Bit Anzahl der zu setzenden Outputs

Start Address

gibt an, ab welcher Speicherstelle Output-Bits geschrieben wurden.

Quantity of Outputs

Hier wird die Anzahl der geschriebenen Output-Bits übergeben.

Beispiel:
Output1 eines Web-IO soll auf ON geschaltet werden

Output 0 soll auf OFF geschaltet werden, Output 1 auf ON

Output Address wird auf 1020 gesetzt,

Quantity of Outputs auf 0x02

Byte Count auf 0x01

Es wird ein Byte Output Value mit Inhalt 0x02 übergeben.

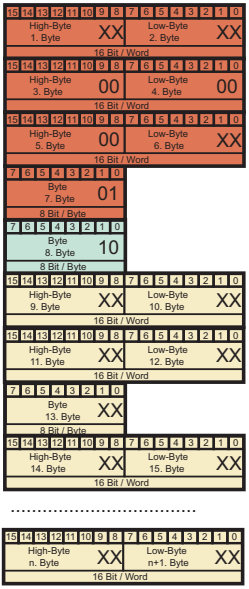
Das Web-IO antwortet mit

Output Address = 1020

Quantity of Outputs = 0x0002

Function Code 0x10 Write Multiple Registers

Der *Function Code* 0x0F ist für das Schreiben mehrerer 16-Bit Registerwerte vorgesehen.



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Start Address	16 Bit Adresse des ersten zu setzenden Outputs
Quantity of Registers	16 Bit Anzahl der zu schreibenden Register
Byte Count	8 Bit Anzahl der Bytes nach ByteCount
Register Value	16 Bit / 32 Bit Die Register werden beginnend beim 14.Byte mit dem ersten Highbyte übergeben

Start Address

gibt an, ab welcher Speicherstelle Register geschrieben werden sollen.

Quantity of Registers

Hier wird die Anzahl der zu schreibenden 16-Bit Register übergeben. Beim Schreiben in den 32-Bit Bereich des Web-IO müssen je 32-Bit Wert zwei 16-Bit Register gerechnet werden.

Bytes Count

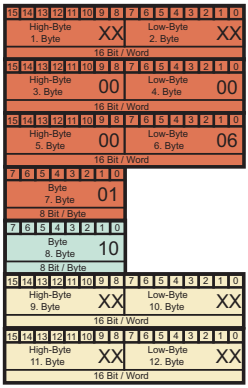
enthält die Anzahl der zu übertragenen Bytes. Für jedes zu sendende 16-Bit Register werden 2 Bytes gerechnet.

Register Value

Hier werden die 16-Bit Register übergeben. Das erste High-Byte wird als 14. Byte in das Datenpaket gesetzt. Das erste Low-Byte als 15. Byte usw.

Beim Schreiben in den 32-Bit Bereich des Web-IO (ab Adresse 0x5000) müssen je 32-Bt Wert zwei 16-Bit Register beginnend mit dem höchsten Byte des 32-Bit Wertes geschrieben werden.

Das Web-IO antwortet mit einem Datenpaket mit folgendem Aufbau:



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Start Address	16 Bit Adresse des ersten zu setzenden Registers
Quantity of Registers	16 Bit Anzahl der geschriebenen Register

Start Address

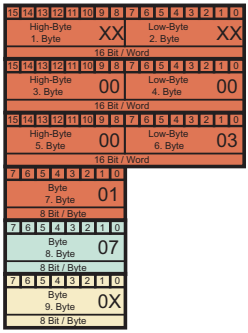
gibt an, ab welcher Speicherstelle die 16-Bit Register geschrieben wurden.

Quantity of Outputs

Hier wird die Anzahl der geschriebenen 16-Bit Register übergeben.

Function Code 0x07 Read Exception State

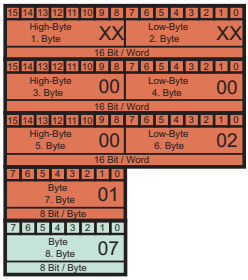
Der *Function Code* 0x07 ist für das Auslesen des Web-IO Fehlerstatus vorgesehen.



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung
Exception Code	8 Bit Web-IO Fehlerstatus

Vom Master gesendete Pakete mit *Function Code* 0x07 bestehen nur aus Modbus-Header und *Function Code.* Es werden keine weiteren Parameter mitgesendet.

Das Web-IO antwortet mit einem Datenpaket mit folgendem Aufbau:



Transaction ID	16 Bit Wird vom Client je Sendung um 1 hochgezählt
Protocol ID	16 Bit Immer = 0x0000
Length	16 Bit Anzahl der Bytes nach dem Header
Unit ID	8 Bit Immer = 0x01
Function Code	8 Bit Bestimmt den Zweck der Datensendung

Exception Code

beim Web-IO spiegelt der Exception Code einzig den Zustand der LEDs *System Error* und *On Error* wieder.

Bit 0 = 1 entspricht *System Error*
Bit 2 = 1 entspricht *ON Error*

Alle anderen Bits haben beim Web-IO keine Funktion und werden mit 0 übergeben.

Fehlerhafte Anfragen vom Master

Sollte der Modbus TCP Master dem Web-IO Anfragen senden, die in Form und Anzahl der Parameter oder hinsichtlich der gewählten Registeradresse für das Web-IO nicht zu verarbeiten sind, sendet das Web-IO ein Modbus Paket zurück, das vom Aufbau her einer Antwort auf *Function Code* 0x07 entspricht.

Das Web-IO addiert in solchen Fällen auf den empfangenen *Function Code* 0x80 auf und setzt diesen Code in das eigene Paket ein.

Der *Exception Code* enthält weitere Details zum Fehler:

0x01	Function Code vom Web-IO nicht unterstützt
0x02	Schreibversuch auf einen Read Only Bereich
0x03	Schreibversuch mit falschen Daten
0x04	Schreiben auf erlaubten Bereich fehlgeschlagen

Modbus-Adressbereiche für Web-IO Digital

Alle Adress-Angaben sind hexadezimal zu verstehen.

Beim Web-IO gibt es verschiedene Modbus-Speicherbereiche:

- Bit-Bereich (ab Adresse 0x1000),
- 16Bit-Bereich (ab Adresse 0x2000)
- 32-Bit-Bereich bzw. 2x16-Bit-Bereich (ab Adresse 0x5000)

Die Adressierung erfolgt im Bit-Bereich bitweise, d.h. 1 Bit benötigt eine Adresse. Im 16-Bit- und im 32-Bit-Bereich findet die Adressierung wortweise (2 Byte) statt.

Die Inputs
finden sich im Bit-Bereich ab Adresse 0x1000

(Beispiel 57637: 0x1000 und 0x1001),
im 16-Bit-Bereich ab 0x2000 im 16-Bit-Bereich ab 0x2000

(Beispiel 57637: 0x2000),
im 32-Bit-Bereich ab 0x5000.

Die Outputs
finden sich im Bit-Bereich ab Adresse 0x1020

(Beispiel 57637: 0x1020 und 0x1021)
im 16-Bit-Bereich ab 0x2002.
im 32-Bit-Bereich ab 0x5002.

Die Alarme
finden sich im Bit-Bereich ab Adresse 0x1040

(Beispiel 57637: 0x1040 und 0x1041),
im 16-Bit-Bereich bei 0x2004.

Alarm-Trigger liegen im Bit-Bereich ab Adresse 0x1800.

Die Counter
finden sich im 32-Bit-Bereich ab Adresse 0x5004

(Beispiel 57637: 0x5004/0x5005 und 0x5006/0x5007).

Counter-Werte schreiben: Counter können auf beliebige Werte gesetzt werden.

Der Exception-Status
findet sich im Bit-Bereich ab Adresse 0x1050,

im 16-Bit-Bereich bei 0x200C (Low Byte).

Alternativ wird der Exception-Status mit dem *Function Code* 0x07 ausgelesen.

Der Konfigurations-Status
findet sich im Bit-Bereich bei Adresse 0x1058,
im 16-Bit-Bereich bei 0x200C (High Byte).

Der Diagnose-Status
(Anzahl Fehler) liegt im 16Bit-Bereich bei 0x2005,
im 32-Bit-Bereich bei 0x5048.

Die Diagnose-Status-Bits
liegen im 16Bit-Bereich ab 0x2006,

im 32-Bit-Bereich ab 0x5048.

Identifizierung des Gerätes
erfolgt über Seriennummer (ab 0x6000) und Mac-Adresse (ab 0x6004).

Virtuelle Register für Browser-Interaktion
Der Speicherbereich, den das Gerät für die Übergabe an Web-Anwendungen zur Verfügung stellt, liegt ab Adresse 0x7000.

Lesen von nicht unterstützen Registern
Beim Lesen von Daten (Speicherbereichen), die nicht für das Gerät definiert wurden, gibt das Gerät „0“ zurück.

Modbus - virtuelle Register
Das Web-IO stellt 64 virtuelle 16-Bit Register zur Verfügung, in die vom Modbus Master beliebige Werte geschrieben werden können (High Byte first). Das Schreiben in diese Register löst im Web-IO keine speziellen Aktionen aus.

Der virtuelle Speicher dient vielmehr zur Übergabe von Modbus- Prozessdaten an Web-Anwendungen.

Über einen HTTP Request

```
modbusreg?PW=<password>&
```

können die 64 Register (128Byte) von Web-Anwendungen abgerufen werden.

Das Web-IO antwortet mit

```
modbus;<High Byte1>;<Low Byte1>;<High Byte2>;
<Low Byte2>;<High Byte3>;......
```

Das heißt, alle 64 Register (128Byte) werden Byte für Byte mit Semikolon getrennt hinter dem Wort „modbus“ ausgegeben.

Über JavaScript und Programmiertechniken wie AJAX kann so

eine Prozessvisualisierung im Browser realisiert werden.

Im einfachsten Fall können die virtuellen Register auf der werksseitig vorhandenen User-Seite des Web-IO tabellarisch angezeigt werden (nur wenn die Betriebsart Modbus im Web-IO aktiviert wurde).

Web-IO User Site
WEBIO-05372A
Von der Klemme direkt aufs Netzwerk

Logout

Inputs				Outputs			
Name	*State*	*Counter*	*Clear*	*Name*	*State*	*Switch*	
Input 0	ON	0	Clear	Output 0	OFF	ON	OFF
Input 1	ON	0	Clear	Output 1	OFF	ON	OFF

Free available Modbus Memory

Addr.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
7000	26 38	FE 12	AF 45	20 16	39 45	FC D3	09 CD	23 FA	16 29	F3 D7	FD 2A	45 9F	35 2F	DA 29	58 39	F2 D1
7010	16 29	F3 D7	FD 2A	45 9F	35 2F	DA 29	58 39	F2 D1	23 4E	FC 56	01 29	56 39	23 E4	F1 AC	CD CD	45 91
7020	23 4E	FC 56	01 29	56 39	23 E4	F1 AC	CD CD	45 91	12 AC	56 FF	92 73	5F 2A	FA DE	34 1D	00 02	1F A1
7030	12 AC	56 FF	92 73	5F 2A	FA DE	34 1D	00 02	1F A1	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00

Show as: ○ 8Bit ◉ 16Bit ○ 32Bit