IEC-Terminal Script Professional

Dokument	Dokumentation
Version	0.5
Status	in arbeit
Ausgabe vom	13.08.2015
Autor	F. Scherer
Datum erstellt	15.06.2010

ZETA Engineering AG, Zug
Industrielle HW- und SW-Entwicklung
Tel: 041 450 22 11
Fax: 041 783 20 01
mail@zeta-eng.ch
www.zeta-eng.ch

Inhaltsverzeichnis

1 Allgemein

1.1 Zweck

Dokumentiert die IEC-Script-Sprache, welche in der Applikation „ZETA Engineering AG – IECTerminal Professional" integriert ist.

1.2 Änderungen

Version	Datum	Author	Kommentar
0.1	31.03.2009	F.Scherer	Dokument Erstellung
0.2	22.07.2009	F.Scherer	Befehl TICKS zugefügt
0.3	11.08.2009	F.Scherer	SCRIPT.EXCEPTIONABORT eingefügt
0.4	15.06.2010	F.Scherer	Befehl INPUT zugefügt
0.5	13.08.2015	F.Scherer	New GET param DEVICEIDENT

1.3 Referenzierte Dokumente

Ref.	Doc-Number	Title	Author / Phone-Nr
[1]	IEC 62056-21	Standard Electricity Metering – Data exchange for meter reading, tarif and load control	CEI IEC

1.4 Begriffe, Definitionen, Abkürzungen

Abkürzung	Definition	Erklärung
Nibble	Nibble	Ein Nibble umfasst 4 Bits (Halbbyte)
HEX	Hexadecimal	Umfasst Zeichen 0-9, A-F für Wertebereich 0 – 15
STRING	Zeichenkette	

2 IEC-Script

2.1 Script-Editor

Zum editieren der Scripts werden Komponenten von ScintillaNet und Scintilla verwendet.

2.1.1 Lizenz ScintillaNet

2.1.2 Lizenz Scintilla

2.2 Reservierte Wörter

2.2.1 Kontroll-Strukturen

VAR, IF, THEN, ELSE, WHILE, DO, END

2.2.2 Boolsche Operatoren

AND, OR, NOT

2.2.3 Vordefinierte Variablen (nur lesbar)

NULL, TRUE, FALSE, SP, CR, LF

2.2.3.1 Variablen Methoden

LENGTH, FIND, CONCAT, SUBSTR, ASCHEX, HEXASC, HEXDEC, DECHEX

2.2.4 Kommando

MSG , WAIT, TICKS, SCRIPT

2.2.4.1 Kommando Methoden

ABORT, EXIT, RUN

2.2.4.2 Kommando Eigenschaften

TEXT, TIME, NEXT, CURR, EXCEPTIONABORT, ERROR

2.2.5 IEC Kommando

BREAK, PROGMODE, GET, SET, READOUT, R1, W1, R2, W2, E2, R5, W5, R6

2.2.5.1 IEC Kommando Methoden

RUN (nicht für GET und SET)

2.2.5.2 IEC Kommando Eigenschaften

ADDR, AUTOLOADSYMBOLFILE, BACKUPADDRESS, BAUDRATESWITCHTIME, CODE, COUNT, DATA, DATABITS, DEFAULT, DEVADDR, ECHO, ERROR, FIRSTBYTETIMEOUT, INIBAUD, INTERBYTETIMEOUT, INTERFACEPOWER, MAXBAUD, PARITY, PASSWORDP1, PASSWORDP1V, PASSWORDP2, PASSWORDP2V, PASSWORDW5, PASSWORDW5V, PORT, PROTOCOLMODE, RESPONSETIME, SECURITYLEVEL,SHORTREADCOUNT, SOFTPARITY, SUBVERSION, SYMBOLFILE, SYMBOLFILEPATH, TAGS, TEAKEY, TEAKEYV, TRACE, TRIALS

2.3 Datentypen

IEC Script unterscheidet intern zwei Datentypen: Zahlen und Strings. Gespeichert werden die Daten aber immer als String. Vor arithmetischen Operationen wird geprüft, ob sich die Inhalte der beteiligten Variable in Zahlen konvertieren lassen. Ist dies nicht der Fall, wird ein Fehler ausgelöst (Ausnahme + operator).

Der Zahlenbereich ist: $\pm 5.0 \times 10^{-324}$ bis $\pm 1.7 \times 10^{308}$ 15-16 Stellen genau

2.4 Vordefinierte Werte

Vordefinierte Werte sind intern definierte Variablen. Sie sind generell nur lesbar.

Name	Wert	Beschreibung
NULL	""	Leer-String
TRUE	"True"	Boolscher Wert WAHR
FALSE	"False"	Boolscher Wert FALSCH

SP	" "	Leerzeichen (Space)
CR	"\r"	Wagenrücklauf
LF	"\n"	Neue Zeile

2.5 Boolsche Werte

Folgende Werte werden als logisch <WAHR> interpretiert:

Vordefinierte Variable TRUE (nur lesbar).

"TRUE", "ON", "EIN", "YES", "JA"

Folgende Werte werden als logisch <FALSCH> interpretiert:

Vordefinierte Variable FALSE (nur lesbar).

"FALSE", "OFF", "AUS", "NO", "NEIN"

2.6 Variablen

Variablen müssen deklariert werden. Dazu dient das Schlüsselwort „VAR". Variablendeklarationen müssen nicht am Anfang der IEC-Script-Datei stehen. Es reicht sie vor dem ersten Gebrauch zu deklarieren. Gross/Klein-Schreibung wird nicht unterschieden.

Syntax:

```
VAR <Name>, <Name2>, … <NameN>;
```

Beispiel:

```
VAR loop, message;
loop = 5;
message = "ABC";
```

2.7 Ausdrücke

2.7.1 Zuweisungsoperator

Der Zuweisungsoperator ist: [=]

2.7.2 Arithmetische Operatoren

Es werden die vier Grundrechenarten unterstützt: [+], [-], [*], [/]).

Beispiel:

```
VAR A, B;

A = -1.8;
B = -0.8;

MSG.TEXT = "A(" + A + ") + B(" + B + ") = " + (A+B); MSG.RUN;
MSG.TEXT = "A(" + A + ") - B(" + B + ") = " + (A-B); MSG.RUN;
MSG.TEXT = "A(" + A + ") * B(" + B + ") = " + (A*B); MSG.RUN;
MSG.TEXT = "A(" + A + ") / B(" + B + ") = " + (A/B); MSG.RUN;
```

Ausgabe:

```
A(-1.8) + B(-0.8) = -2.6
A(-1.8) - B(-0.8) = -1
A(-1.8) * B(-0.8) = 1.44
A(-1.8) / B(-0.8) = 2.25
```

2.7.3 Boolsche Operatoren

Unterstützt sind folgende boolschen Operatoren: AND OR NOT

2.7.4 Vergleichsoperatoren

Folgende vier Vergleichsoperatoren sind definiert: [=], [<>], [>=], [<=]

Beispiel:

```
VAR A, B, C;

A = TRUE;
B = TRUE;
C = FALSE;

MSG.TEXT = "-- AND --"; MSG.RUN;
IF A=TRUE  AND B=TRUE  THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=FALSE AND B=TRUE  THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=TRUE  AND B=FALSE THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=FALSE AND B=FALSE THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;

MSG.TEXT = "-- NAND --"; MSG.RUN;
IF NOT A=TRUE  AND B=TRUE  THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF NOT A=FALSE AND B=TRUE  THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF NOT A=TRUE  AND B=FALSE THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF NOT A=FALSE AND B=FALSE THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;

MSG.TEXT = "-- OR --"; MSG.RUN;
IF A=TRUE  OR B=TRUE  THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=FALSE OR B=TRUE  THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=TRUE  OR B=FALSE THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=FALSE OR B=FALSE THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;

MSG.TEXT = "-- NOR --"; MSG.RUN;
IF NOT A=TRUE  OR B=TRUE  THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF NOT A=FALSE OR B=TRUE  THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF NOT A=TRUE  OR B=FALSE THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF NOT A=FALSE OR B=FALSE THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;

MSG.TEXT = "-- AND OR --"; MSG.RUN;
IF A=TRUE OR (B=TRUE AND C=FALSE) THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=FALSE OR (B=TRUE AND C=FALSE) THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=TRUE OR (B=FALSE AND C=FALSE) THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A=FALSE OR (B=FALSE AND C=FALSE) THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;

A = 100;
B = 50;
C = 1;
MSG.TEXT = "-- Zahlen gleich unglech --"; MSG.RUN;
IF A=100 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A<>100 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
MSG.TEXT = "-- Zahlen groesser kleiner --"; MSG.RUN;
IF A>99 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A<101 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A<99 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A>101 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
MSG.TEXT = "-- Zahlen groesser gleich kleiner unglech --"; MSG.RUN;
IF A>100 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A>=100 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
IF A<100 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
```

```
IF A<=100 THEN MSG.TEXT = TRUE; ELSE MSG.TEXT = FALSE; END; MSG.RUN;
```

Ausgabe:

```
-- AND ---
True
False
False
False
-- NAND -
False
True
True
True
-- OR -
True
True
True
False
-- NOR -
False
False
False
True
-- AND OR -
True
True
True
False
-- Zahlen gleich unglech -
True
False
-- Zahlen groesser kleiner -
True
True
False
False
-- Zahlen groesser gleich kleiner unglech -
False
True
False
True
```

2.8 Kontroll-Strukturen

Zur bedingten Ausführung von Code sind folgende Anweisungen vorhanden:

IF-TEHN-ELSE und WHILE-DO

2.8.1 IF-THEN-ELSE

Syntax:

```
IF BEDINGUNG THEN
  Anweisung1;
  Anweisung2;

  AnweisungN;
ELSE
  Anweisung1;
  Anweisung2;

  AnweisungN;
END;
```

Beispiel:

2.8.2 WHILE-DO

Syntax:

```
WHILE BEDINGUNG DO
  Anweisung1;
  Anweisung2;

  AnweisungN
END;
```

Beispiel:

```
VAR loop, message;
loop = 0;

WHILE loop < 3 DO
 MSG.TEXT = "Var loop contains: " + loop; MSG.RUN;
 loop = loop + 1;
END;
```

2.9 Variablen-Methoden

2.9.1 ASC nach HEX

Gibt den Inhalt der Variable <Varibalenname> interpretiert als ASCII-String als Hex-Nibble-Paar-String aus.

Syntax:

```
<Varibalenname>.ASCHEX;
```

Beispiel:

```
VAR Asc;

Asc = 12345678;
MSG.TEXT = "Var Asc contains: " + Asc; MSG.RUN;
MSG.TEXT = "ASC to HEX      : " + Asc.ASCHEX; MSG.RUN;
```

Ausgabe :

```
Var Asc contains: 12345678
ASC to HEX      : 3132333435363738
```

2.9.2 HEX nach ASC

Gibt den Inhalt der Variable <Varibalenname> interpretiert als Hex-Nibble-Paar-String als ASCII-String aus.

Syntax:

```
<Varibalenname>.HEXASC;
```

Beispiel:

```
VAR Hex;

Hex = 31323334;
MSG.TEXT = "Var Hex contains: " + Hex; MSG.RUN;
MSG.TEXT = "HEX to ASC      : " + Hex.HEXASC; MSG.RUN;
```

Ausgabe :

```
Var Hex contains: 31323334
Hex to ASC      : 1234
```

2.9.3 Länge

Gibt die Länge des Inhaltes der Variable <Varibalenname> aus.

Syntax:

```
<Varibalenname>.LENGTH;
```

Beispiel:

```
VAR Str;

Str = 31323334;
MSG.TEXT = "Var Str contains: " + Str; MSG.RUN;
MSG.TEXT = "LENGTH          : " + Str.LENGTH; MSG.RUN;
```

Ausgabe :

```
Var Str contains: 31323334
LENGTH          : 8
```

2.9.4 DEZ nach HEX

Gibt den Inhalt der Variable <Varibalenname> interpretiert als Dezimalzahl als Hexadezimalzahl aus.

Syntax:

```
<Varibalenname>.DECHEX;
```

Beispiel:

```
VAR Dez;

Dez = 12345678;
MSG.TEXT = "Var Dez contains: " + Dez; MSG.RUN;
MSG.TEXT = "DEC to HEX      : " + Dez.DECHEX; MSG.RUN;
```

Ausgabe :

```
Var Dez contains: 12345678
DEC to HEX      : BC614E
```

2.9.5 HEX nach DEZ

Gibt den Inhalt der Variable <Varibalenname> interpretiert als Hexadezimalzahl als Dezimalzahl aus.

Syntax:

```
<Varibalenname>.HEXDEC;
```

Beispiel:

```
VAR Hex;
```

```
Hex = "BC614E";
MSG.TEXT = "Var Hex contains: " + Hex; MSG.RUN;
MSG.TEXT = "HEX to DEC      : " + Hex.HEXDEC; MSG.RUN;
```

Ausgabe :

```
Var Hex contains: BC614E
HEX to DEC      : 12345678
```

2.9.6 Teil-String

Gibt eine Teilstring des Inhaltes der Variable <Varibalenname> aus.

Syntax:

```
<Varibalenname>.SUBSTR <Start> [, <Länge>];
```

Beispiel:

```
VAR Str, Start, Len;

Str = "ABC123";
Start = 3;
Len = 2;
MSG.TEXT = "Var Str contains      : " + Str; MSG.RUN;
MSG.TEXT = "Var Start contains    : " + Start; MSG.RUN;
MSG.TEXT = "Var Len contains      : " + Len; MSG.RUN;
MSG.TEXT = "SUBSTR <Start>, <Len> : " + Str.SUBSTR Start, Len; MSG.RUN;
MSG.TEXT = "SUBSTR <Start>        : " + Str.SUBSTR Start; MSG.RUN;
MSG.TEXT = "SUBSTR 0,1            : " + Str.SUBSTR 0, 1; MSG.RUN;
```

Ausgabe :

```
Var Str contains      : ABC123
Var Start contains    : 3
Var Len contains      : 2
SUBSTR <Start>, <Len> : 12
SUBSTR <Start>        : 123
SUBSTR 0,1            : A
```

2.9.7 String suchen

Sucht nach einem <String> im Inhalte der Variable < Varibalenname> und gibt die Position zurück, an welcher der <String> gefunden wurde.

Syntax:

```
<Varibalenname>.FIND <String> [, <Start>];
```

Beispiel:

```
VAR Val, Str, Start;

Val = "ABC123C123";
Str = "C1";
Start = 0;

MSG.TEXT = "Var Val contains      : " + Val; MSG.RUN;
MSG.TEXT = "Var Str contains      : " + Str; MSG.RUN;
MSG.TEXT = "Var Start contains    : " + Start; MSG.RUN;
Start = Val.FIND Str, Start;
MSG.TEXT = "FIND <Str>, <Start>   : " + Start; MSG.RUN;
Start = Start + 1;
```

```
MSG.TEXT = "Var Start contains    : " + Start; MSG.RUN;
Start = Val.FIND Str, Start;
MSG.TEXT = "FIND <Str>, <Start>   : " + Start; MSG.RUN;
MSG.TEXT = "FIND <Str>            : " + Val.FIND Str; MSG.RUN;
```

Ausgabe:

```
Var Val contains      : ABC123C123
Var Str contains      : C1
Var Start contains    : 0
FIND <Str>, <Start>   : 2
Var Start contains    : 3
FIND <Str>, <Start>   : 6
FIND <Str>            : 2
```

2.9.8 Strings verknüpfen

2.9.8.1 Operator +

Verknüpft zwei Strings wenn einer der beiden Strings keine Zahl ist. Sind beide Strings gültige Zahlen, wird eine Addition durchgeführt.

Syntax:

<Varibalenname1> + <Varibalenname2>;

2.9.8.2 CONCAT

Verknüpft den Inhalt zweier Variablen als String.

<Varibalenname1>.CONCAT <Varibalenname2>;

Beispiel:

```
VAR Str1, Str2, Str3;

Str1 = 1;
Str2 = 1;
MSG.TEXT = "Var Str1 contains: " + Str1; MSG.RUN;
MSG.TEXT = "Var Str2 contains: " + Str2; MSG.RUN;
Str3 = Str1 + Str2;
MSG.TEXT = "Str1 + Str2      : " + Str3; MSG.RUN;
Str3 = Str1.CONCAT Str2;
MSG.TEXT = "Str1.CONCAT Str2 : " + Str3; MSG.RUN;
Str1 = "A";
Str2 = 1;
MSG.TEXT = "Var Str1 contains: " + Str1; MSG.RUN;
MSG.TEXT = "Var Str2 contains: " + Str2; MSG.RUN;
Str3 = Str1 + Str2;
MSG.TEXT = "Str1 + Str2      : " + Str3; MSG.RUN;
```

Ausgabe:

```
Var Str1 contains: 1
Var Str2 contains: 1
Str1 + Str2      : 2
Str1.CONCAT Str2 : 11
Var Str1 contains: A
Var Str2 contains: 1
Str1 + Str2      : A1
```

3 Allgemeine Befehle

3.1 MSG

Gibt eine Meldung aus.

Eigenschaften	Standard Wert	Beschreibung
TEXT	""	Meldungstext

Methoden		Beschreibung
RUN		Setzt Meldung ab.

Syntax :

```
MSG.TEXT = <String>;  // weist den Meldungstext zu
MSG.RUN;                      // führt aus
```

Beispiel:

```
VAR Hex;

Hex = "BC614E";
MSG.TEXT = "Var Hex contains: " + Hex; MSG.RUN;
MSG.TEXT = "HEX to DEC      : " + Hex.HEXDEC; MSG.RUN;
```

Ausgabe :

```
Var Hex contains: BC614E
HEX to DEC      : 12345678
```

3.2 SCRIPT

Definiert Script-Eigenschaften und Methoden.

Eigenschaften	Standard Wert	Beschreibung
ERROR	""	Benutzer Fehlermeldung
TRACE	FALSE	Schaltet Datei-Trace ein oder aus
EXCEPTIONABORT	TRUE	Wenn auf TRUE gesetzt wird das Script bei einer Ausnahme abgebrochen.

Methoden		Beschreibung
ABORT		Bricht Ausführung mit Ausgabe von ERROR ab.
EXIT		Beendet die Ausführung mit Status OK.

Syntax:

```
SCRIPT.TRACE = TRUE | FALSE;          // Trace ein oder ausschalten
SCRIPT.ERROR = <Fehlermeldung>;       // setzt Fehlermeldung
SCRIPT.ABORT;                         // bricht mit Fehlermeldung ab
SCRIPT.EXIT;                          // beendet Ausführung
SCRIPT.EXCEPTIONABORT = TRUE | FALSE;
```

Beispiel:

```
VAR Rout;
SCRIPT.EXCEPTIONABORT = FALSE;
READOUT.DEVADDR = ""; // default
Rout = READOUT.RUN;
If( Rout = FALSE )
  SCRIPT.ERROR = "Fehler beim Readout";
  SCRIPT.ABORT;
END;
```

3.3 TICKS

Ermöglicht Operationen mit dem 1ms Systemzeitgeber. Die Zählung beginnt beim Einschalten des Systems jeweils bei 0 und wird intern als 32-Bit-Ganzzahl gespeichert. Folglich erhöht sich der Wert bis auf (2^31)-1 über ca. 24.9 Tage. Danach wird er auf –(2^31) gesetzt und erhöht sich während der folgenden 24.9 Tagen wieder auf 0.

Eigenschaften	Standard Wert	Beschreibung
CURR	Aktueller Tickwert	Gibt den aktuellen internen Wert des Systemzeitgebers zurück (ms).
NEXT	0	Tickwert (ms), auf welchen mit dem Sleep-Kommando geartet wird. Überschreitet der Wert (2^31)-1, wird er auf –(2^31) + Überlauf gesetzt.

Methoden		Beschreibung
SLEEP		Setzt Ausführung aus bis der Systemzeitgeber intern den Wert von NEXT erreicht hat.

Syntax:

```
t = TICKS.CURR;                       // Aktueller Tickwert abfragen
TICKS.NEXT = <Tickwert>;              // setzt Tickwert
TICKS.SLEEP;                          // Setzt Ausführung aus
```

Beispiel:

```
VAR loop, ticksInterval;

loop = 0;
READOUT.DEVADDR = "";
ticksInterval = 7000;

TICKS.NEXT = TICKS.CURR + ticksInterval;
WHILE loop < 100 DO
 READOUT.RUN;
 loop = loop + 1;
 TICKS.SLEEP;
 TICKS.NEXT = TICKS.NEXT + ticksInterval ;
END;
```

3.4 WAIT

Unterbricht die Ausführung für die angegebene Zeit (ms).

Eigenschaften	Standard Wert	Beschreibung
TIME	500	Wartezeit in ms

Methoden		Beschreibung
RUN		Setzt Ausführung für Wartezeit aus.

Syntax :

```
WAIT.TIME = <MS>;            // weist die Zeit (ms) zu
WAIT.RUN;                    // führt aus
```

3.5 INPUT

Zeigt eine Eingabedialogbox an. Der Dialog besitzt ein Meldungsfenster, ein Eingabefenster und zwei Tasten (Ok, Cancel).

Eigenschaften	Standard Wert	Beschreibung
TEXT		Der Meldungstext für die Eingabebox
DATA		Die eingegebenen Daten

Methoden		Beschreibung
RUN		Zeigt Eingabebox an. OK-Taste gibt TRUE zurück. Cancel-Taste gibt FALSE zurück.

Syntax :

```
INPUT.TEXT = "Enter device number";
IF INPUT.RUN = TRUE THEN
  MSG.TEXT = INPUT.DATA; MSG.RUN;
END;
```

4 IEC-Befehle

4.1 Set und Get

Set setzt Eigenschaften im IEC-Protokoll-treiber. Die Befehle werden unmittelbar ausgeführt. Sie haben keine RUN-Methode.

Get liest den momentanen eingestellten Wert direkt vom IEC-Protokoll-Treiber. SET setzt die Werte im IEC-Protokoll-Treiber.

4.1.1 Passwörter

Wenn der Initialisierungsvektor der Passwörter auf Leerstring gesetzt wird, können die Passwörter (PASSWORDP1, PASSWORDP1, TEAKEY) auch unverschlüsselt gesetzt werden (wird nicht empfohlen).

Syntax:

```
SET.<Eigenschaft> = <Wert>;
GET.<Eigenschaft>;
```

Eigenschaften	Anmerkung	Beschreibung
DEFAULT	Nur Set	Setzt alle Eigenschaften gemäss den Setup-Einstellungen
PORT		Setzt den Kommunikationsport
INIBAUD	300-115200	Setzt die Startbaudrate
MAXBAUD	300-115200	Setzt die maximale Baudrate
ECHO	TRUE,FALSE	Legt fest, ob die Schnittstelle ECHO besitzt
INTERFACEPOWER	TRUE,FALSE	Legt fest, ob die Schnittstelle durch Steuerleitungen gespeist wird
RESPONSETIME	20,200	Legt die Antwortzeit fest (ms)
FIRSTBYTETIMEOUT	1500-	Legt den Timeout für das erste Byte fest (ms)
INTERBYTETIMEOUT	1500	Legt den Timeout für die nachfolgenden Bytes fest (ms)
BAUDRATESWITCHTIME	32	Zusätzliche Wartezeit bevor Baudrate umgeschaltet wird (ms)
DATABITS	7,8	Anzahl Datenbits
PARITY	NONE,EVEN,ODD	Parität
SOFTPARITY	NONE,EVEN,ODD	Parität für "SoftParity"
PROTOCOLMODE	A,B,C,D	IEC-Protokoll-Mode
SECURITYLEVEL	0,1,2,TEA	Sicherheits-Stufe
PASSWORDP1		P1 Passwort
PASSWORDP1V		P1 Passwort Initialisierungsvektor
PASSWORDP2		P2 Passwort
PASSWORDP2V		P2 Passwort Initialisierungsvektor
TEAKEY		TEA Schlüssel
TEAKEYV		TEA Schlüssel Initialisierungsvektor
TRIALS	1-	Anzahl Versuche im Fehlerfall

SHORTREADCOUNT	TRUE,FALSE	Anzahlformat bei R1: 1 oder 2 Hex-Nibble-Paare
SYMBOLFILEPATH		Verzeichnis der Symbolfiles
SYMBOLFILE		Symbolfilename
AUTOLOADSYMBOLFILE	TRUE,FALSE	Legt fest, ob das Symbolfile automatisch geladen wird
SUBVERSION	TRUE,FALSE	Legt fest, ob die Software-Unterversion verwendet wird
DEVICEIDENT		Device identifier (read only (GET)), will be set in signon

Beispiel:

```
SET.DEFAULT;
MSG.TEXT = "PORT                  " + GET.PORT; MSG.RUN;
MSG.TEXT = "INIBAUD               " + GET.INIBAUD; MSG.RUN;
MSG.TEXT = "MAXBAUD               " + GET.MAXBAUD; MSG.RUN;

SET.INIBAUD = 2400;
SET.MAXBAUD = 2400;
MSG.TEXT = "INIBAUD               " + GET.INIBAUD; MSG.RUN;
MSG.TEXT = "MAXBAUD               " + GET.MAXBAUD; MSG.RUN;


MSG.TEXT = "ECHO                  " + GET.ECHO; MSG.RUN;
MSG.TEXT = "INTERFACEPOWER        " + GET.INTERFACEPOWER; MSG.RUN;
MSG.TEXT = "RESPONSETIME          " + GET.RESPONSETIME; MSG.RUN;
MSG.TEXT = "FIRSTBYTETIMEOUT      " + GET.FIRSTBYTETIMEOUT; MSG.RUN;
MSG.TEXT = "INTERBYTETIMEOUT      " + GET.INTERBYTETIMEOUT; MSG.RUN;
MSG.TEXT = "BAUDRATESWITCHTIME    " + GET.BAUDRATESWITCHTIME; MSG.RUN;
MSG.TEXT = "DATABITS              " + GET.DATABITS; MSG.RUN;
MSG.TEXT = "PARITY                " + GET.PARITY; MSG.RUN;
MSG.TEXT = "SOFTPARITY            " + GET.SOFTPARITY; MSG.RUN;
MSG.TEXT = "PROTOCOLMODE          " + GET.PROTOCOLMODE; MSG.RUN;
MSG.TEXT = "SECURITYLEVEL         " + GET.SECURITYLEVEL; MSG.RUN;
MSG.TEXT = "PASSWORDP1            " + GET.PASSWORD_P1; MSG.RUN;
MSG.TEXT = "PASSWORDP2            " + GET.PASSWORD_P2; MSG.RUN;
MSG.TEXT = "TEAKEY                " + GET.TEAKEY; MSG.RUN;
MSG.TEXT = "TRIALS                " + GET.TRIALS; MSG.RUN;
MSG.TEXT = "SHORTREADCOUNT        " + GET.SHORTREADCOUNT; MSG.RUN;
MSG.TEXT = "SYMBOLFILEPATH        " + GET.SYMBOLFILEPATH; MSG.RUN;
MSG.TEXT = "SYMBOLFILE            " + GET.SYMBOLFILE; MSG.RUN;
MSG.TEXT = "AUTOLOADSYMBOLFILE    " + GET.AUTOLOADSYMBOLFILE; MSG.RUN;
MSG.TEXT = "SUBVERSION            " + GET.SUBVERSION; MSG.RUN;
```

Ausgabe:

```
PORT                S.ZEH20000
INIBAUD             300
MAXBAUD             9600
INIBAUD             2400
MAXBAUD             2400
ECHO                OFF
INTERFACEPOWER      OFF
RESPONSETIME        200
FIRSTBYTETIMEOUT    4000
INTERBYTETIMEOUT    1500
BAUDRATESWITCHTIME  32
DATABITS            7
PARITY              EVEN
SOFTPARITY          NONE
PROTOCOLMODE        C
SECURITYLEVEL       0
PASSWORDP1
PASSWORDP2
TEAKEY
TRIALS              3
SHORTREADCOUNT      OFF
SYMBOLFILEPATH
SYMBOLFILE
AUTOLOADSYMBOLFILE  OFF
SUBVERSION          OFF
```

4.2 Readout

Führt ein Readout aus und gibt das Ergebnis als String zurück.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse

Syntax:

```
READOUT.DEVADDR = <Geräteadresse>;   // Setzt Geräteadresse
READOUT.RUN;                                 // führt aus
```

Beispiel:

```
VAR Rout;

READOUT.DEVADDR = ""; // default
Rout = READOUT.RUN;
READOUT.DEVADDR = ".2";
Rout = READOUT.RUN;
MSG.TEXT = Rout; MSG.RUN;
```

Ausgabe:

```
MSG: 0.2.1();C.90(D.M4MD);C.2.0(00000003);C.0.0(LGZ84416138-2);0.2.0(D44);C.2.1(05-07-13
11:25);C.240.13(D.M4);
```

4.3 Programming Mode

Setzt das Gerät in den Programming Mode. Der Rückgabewert ist <WAHR> bei Erfolg, sonst <FALSCH>.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse

Syntax:

```
PROGMODE.DEVADDR = <Geräteadresse>;          // Setzt Geräteadresse
PROGMODE.RUN;                                        // führt aus
```

Beispiel:

```
VAR Result;

Result = PROGMODE.RUN;
```

4.4 Break

Sendet ein Break-Kommando.

Syntax:

```
BREAK.RUN;
```

4.5 R1

Führt einen unformatierten Lesebefehl (R1) aus und gibt die gelesenen Daten zurück.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse
ADDR	""	IEC-Leseadresse (Hex oder Symbolische Adresse)
COUNT	0	Anzahl zu lesende Bytes

Syntax:

```
R1.DEVADDR = <Geräteadresse>; // Setzt Geräteadresse
R1.ADDR = <IECAdresse>;              // Setzt Leseadresse
R1.COUNT = <Anzahl>;          // Setzt Anzahl zu lesende Bytes
R1.RUN;                       // führt aus
```

Beispiel:

```
VAR R1Res;
R1.ADDR = "1FA0"
R1.COUNT = 22;
R1Res = R1.RUN;
MSG.TEXT = R1Res; MSG.RUN;
```

Ausgabe:

```
MSG: 7D7D7D7D1B777C202020302E322E3400000000010000
```

4.6 R2

Führt einen formatierten Lesebefehl (R2) aus und gibt die gelesenen Daten zurück.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse
CODE	""	R2 Lese-Code
DATA	""	Daten zu R2-Code

Syntax:

```
R2.DEVADDR = <Geräteadresse>; // Setzt Geräteadresse
R2.CODE = <R2Code>;                  // Setzt R2 Lese-Code
R2.DATA = <CodeDaten>;        // Setzt Daten zu Code
R2.RUN;                       // führt aus
```

Beispiel:

```
VAR R2Res;
R2.CODE = "D000";                    // Geräte ID 1 lesen
R2.DATA = "";
R2Res = R2.RUN;
```

```
MSG.TEXT = R2Res; MSG.RUN;
```

Ausgabe:

```
MSG: DEV12345
```

4.7 R5

Führt einen formatierten Lesebefehl (R5) aus und gibt die gelesenen Daten zurück.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse
CODE	""	R5 Lese-Code
DATA	""	Daten zu R5-Code
TAGS	""	Zusätzliche Daten-Merkmale
PASSWORDW5	""	W5 Passwort

Syntax:

```
R5.DEVADDR = <Geräteadresse>; // Setzt Geräteadresse
R5.CODE = <R5Code>;                  // Setzt R5 Lese-Code
R5.DATA = <CodeDaten>;        // Setzt Daten zu Code
R5.TAGS = <R5DatenTag>        // Setzt zusätzliches Datenmerkmal
R5.PASSWORDW5 = <W5Passwort>  // Setzt W5 Passwort
R5.RUN;                       // führt aus
```

Beispiel:

```
VAR R5Res;
R5.CODE = "0.0.0";                   // Geräte ID 1 lesen
R5.DATA = "";
R5.TAGS = "";
R5.PASSWORDW5 = "";
R5Res = R5.RUN;
MSG.TEXT = R5Res; MSG.RUN;
```

Ausgabe:

```
MSG: DEV12345
```

4.8 R6

Führt einen formatierten Lesebefehl (R6) aus und gibt die gelesenen Daten zurück.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse
CODE	""	R5 Lese-Code
DATA	""	Daten zu R5-Code
TAGS	""	Zusätzliche Daten-Merkmale
PASSWORDW5	""	W5 Passwort

Syntax:

```
R6.DEVADDR = <Geräteadresse>; // Setzt Geräteadresse
R6.CODE = <R6Code>;                  // Setzt R5 Lese-Code
R6.DATA = <CodeDaten>;        // Setzt Daten zu Code
R6.TAGS = <R6DatenTag>        // Setzt zusätzliches Datenmerkmal
R6.PASSWORDW5 = <W5Passwort>  // Setzt W5 Passwort
R6.RUN;                       // führt aus
```

Beispiel:

```
VAR R6Res;
R6.CODE = "P.98";                    // Log Buch lesen
R6.DATA = ";;";
R6.TAGS = "";
R6.PASSWORDW5 = "";
R6Res = R6.RUN;
MSG.TEXT = R6Res; MSG.RUN;
```

Ausgabe:

```
MSG:
P.98(0090424041739);(0040);();(2);();();(F.F);();(024);(00000000);P.98(0090423101902);(2080);();(2);()
;();(F.F);();(023);(00000000);
```

4.9 W1

Führt einen unformatierten Schreibbefehl (W1) aus. Gibt TRUE zurück, wenn der Schreibvorgang erfolgreich war. Sonst Fehler-Code oder FALSE.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse
ADDR	""	IEC-Schreibadresse (Hex oder Symbolische Adresse)
DATA	""	Zu schreibende Daten in Hex-Nibble-Paaren

Syntax:

```
W1.DEVADDR = <Geräteadresse>; // Setzt Geräteadresse
W1.ADDR = <IECAdresse>;              // Setzt Leseadresse
W1.DATA = <Daten>;                   // Setzt zu schreibende Daten
W1.RUN;                       // führt aus
```

Beispiel:

```
VAR W1Res;
W1.ADDR = "0064";
W1.DATA = "5555AAAA";
W1Res = W1.RUN;
MSG.TEXT = W1Res; MSG.RUN;
```

Ausgabe:

```
MSG: True
```

4.10 W2

Führt einen formatierten Schreibbefehl (W2) aus. Gibt TRUE zurück, wenn der Schreibvorgang erfolgreich war. Sonst Fehler-Code oder FALSE.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse
CODE	""	W2 Schreib-Code
DATA	""	Daten zu W2-Code

Syntax:

```
W2.DEVADDR = <Geräteadresse>; // Setzt Geräteadresse
W2.CODE = <W2Code>;                 // Setzt R2 Lese-Code
W2.DATA = <CodeDaten>;        // Setzt Daten zu Code
W2.RUN;                       // führt aus
```

Beispiel:

```
VAR W2Res;
W2.CODE = "D000";                   // Geräte ID 1 setzen
W2.DATA = "DEV12345";
W2Res = W2.RUN;
MSG.TEXT = W2Res; MSG.RUN;
```

Ausgabe:

```
MSG: True
```

4.11 W5

Führt einen formatierten Schreibbefehl (W5) aus. Gibt TRUE zurück, wenn der Schreibvorgang erfolgreich war. Sonst Fehler-Code oder FALSE.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse
CODE	""	R5 Lese-Code
DATA	""	Daten zu R5-Code
PASSWORDW5	""	W5 Passwort

Syntax:

```
W5.DEVADDR = <Geräteadresse>; // Setzt Geräteadresse
W5.CODE = <R5Code>;                 // Setzt R5 Lese-Code
W5.DATA = <CodeDaten>;        // Setzt Daten zu Code
W5.PASSWORD_W5 = <W5Passwort> // Setzt W5 Passwort
W5.RUN;                       // führt aus
```

Beispiel:

```
VAR W5Res;
```

```
W5.CODE = "0.0.0";                    // Geräte ID 1 setzen
W5.DATA = "ABC12345";
W5.PASSWORDW5 = "xxxxxxxx";
W5Res = W5.RUN;
MSG.TEXT = W5Res; MSG.RUN;
```

Ausgabe:

```
MSG: True
```

4.12 E2

Führt einen formatierten Befehl (E2) aus. Gibt TRUE zurück, wenn der Schreibvorgang erfolgreich war. Sonst Fehler-Code oder FALSE.

Eigenschaften	Standard Wert	Beschreibung
DEVADDR	""	Geräteadresse
CODE	""	E2 Befehls-Code
DATA	""	Daten zu E2-Code

Syntax:

```
E2.DEVADDR = <Geräteadresse>; // Setzt Geräteadresse
E2.CODE = <E2Code>;                   // Setzt E2 Befehls-Code
E2.DATA = <CodeDaten>;        // Setzt Daten zu Code
E2.RUN;                       // führt aus
```

Beispiel:

```
VAR E2Res;
E2.CODE = "0101";                     // Certification Mode einschalten
E2.DATA = "";
E2Res = E2.RUN;
MSG.TEXT = E2Res; MSG.RUN;
E2.CODE = "0102";                     // Certification Mode ausschalten
E2Res = E2.RUN;
MSG.TEXT = E2Res; MSG.RUN;
```

Ausgabe:

```
MSG: True
MSG: True
```