DLMS Script-Sprache

Dokument	Dokumentation
Version	1.2
Status	freigegeben
Ausgabe vom	21.03.2011
Autor	F. Scherer
Datum erstellt	06.05.2010

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 DLMS-Script Sprache, welche im ZETA DLMS-Terminal verwendet wird.

1.2 Änderungen

Version	Datum	Author	Kommentar
1.0	06.05.2010	F.Scherer	Dokument Erstellung
1.1	09.12.2010	F.Scherer	Modulo-Operator nachgeführt
1.2	21.03.2011	F.Scherer	DLMS-Kommando um LN-Zugriffe erweitert

1.3 Referenzierte Dokumente

Ref.	Doc-Number	Title	Author

1.4 Begriffe, Definitionen, Abkürzungen

Abkürzung	Definition	Erklärung
SN	short name (SN) referencing	The attributes and methods of each interface object are mapped to *DLMS named variables (*more suitable for simple devices)
LN	logical name (LN) referencing	Attributes and methods are accessed via the logical name of the object, specifying the indexes of the attributes and/or the methods (more suitable for complex devices)

2 ZETA DLMS-Terminal

ZETA DLMS-Terminal erlaubt die Kommunikation gemäss **DLMS/COSEM** (IEC 62056) mit allen DLMS/COSEM-konformen Geräten:

IEC Mode E (IEC 62056-21)
HDLC (IEC 62056-46)
DLMS Wrapper (IEC 62056-47 Draft)

ZETA DLMS-Terminal ist nicht Bestandteil dieser Beschreibung.

3 DLMS-SCRIPT

3.1 Reservierte Wörter

3.1.1 Kontroll-Strukturen

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

3.1.2 Boolsche Operatoren

AND, OR, NOT

3.1.3 Vordefinierte Variablen (nur lesbar)

NULL, TRUE, FALSE, SP, CR, LF

3.1.3.1 Variablen Methoden

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

3.1.3.2 Kommandos

MSG, WAIT, SCRIPT, DLMS, INPUT

3.1.3.3 Kommando Methoden

ABORT, EXIT, RUN, READ, WRITE

3.1.3.4 Kommando Eigenschaften

TEXT, TIME, DATA, RESULT, SHORTNAME, ERROR, TRACE

3.2 Datentypen

DLMS-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).

3.3 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

3.4 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"

3.5 Variablen

Variablen müssen deklariert werden. Dazu dient das Schlüsselwort "VAR". Variablendeklarationen müssen nicht am Anfang der DLMS-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";
```

3.6 Ausdrücke

3.6.1 Zuweisungsoperator

Der Zuweisungsoperator ist: [=]

3.6.2 Arithmetische Operatoren

Es werden die vier Grundrechenarten und der Modulo-Operator 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;

A=2000;
B=4;
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
A(2000) % B(4) = 0
```

3.6.3 Boolsche Operatoren

Unterstützt sind folgende boolschen Operatoren: [AND], [OR], [NOT]

3.6.4 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 ungleich --"; 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 ungleich --"; 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 ungleich -
True
False
-- Zahlen groesser kleiner -
True
True
False
False
-- Zahlen groesser gleich kleiner ungleich -
False
True
False
True
```

3.7 Kontroll-Strukturen

Zur bedingten Ausführung von Code sind folgende Anweisungen vorhanden:

IF-THEN-ELSE und WHILE-DO

3.7.1 IF-THEN-ELSE

Syntax:

```
IF BEDINGUNG THEN
  Anweisung1;
  Anweisung2;

  AnweisungN;
ELSE
  Anweisung1;
  Anweisung2;

  AnweisungN;
END;
```

3.7.2 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;
```

3.8 Variablen-Methoden

3.8.1 ASC nach HEX

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

Syntax:

```
<Variablenname>.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
```

3.8.2 HEX nach ASC

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

Syntax:

```
<Variablenname>.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
```

3.8.3 Länge

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

Syntax:

```
<Variablenname>.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
```

3.8.4 DEZ nach HEX

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

Syntax:

```
<Variablenname>.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
```

3.8.5 HEX nach DEZ

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

Syntax:

```
<Variablenname>.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
```

3.8.6 Teil-String

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

Syntax:

```
<Variablenname>.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
```

3.8.7 String suchen

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

Syntax:

```
<Variablenname>.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
```

3.8.8 Strings verknüpfen

3.8.8.1 Operator +

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

Syntax:

```
<Variablenname1> + <Variablenname2>;
```

3.8.8.2 CONCAT

Verknüpft den Inhalt zweier Variablen als String.

```
<Variablenname1>.CONCAT <Variablenname2>;
```

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
```

4 Allgemeine Befehle

4.1 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
```

4.2 SCRIPT

Definiert Script-Eigenschaften und Methoden.

Eigenschaften	Standard Wert	Beschreibung
ERROR	""	Benutzer Fehlermeldung
TRACE	Settings	Schaltet Script-Trace ein oder aus

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
```

Beispiel:

```
VAR Rout;

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

4.3 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
```

4.4 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;
```

4.5 DLMS-Befehl

Mit diesem Befehl können Daten mit dem DLMS-Protokoll gelesen und geschrieben werden.

Eigenschaften	Anmerkung	Beschreibung
SHORTNAME		Aufzählung der Attribut-ShortNames (SN)
RESULT		Resultatdaten: Lesen: Nur Nutzdaten (ohne Metadaten) Schreiben: Erfolg der Aktion
DATA		Daten die gelesen wurden oder die zu schreiben sind (mit Metadaten)
LN		Logical-Name als Hex-Nibble-String (CosemAttributeDescriptor, LN)
CLASS		Class-ID (CosemAttributeDescriptor, LN)
ATTRIB		Attribute-ID (CosemAttributeDescriptor, LN)

Methoden		Beschreibung
OPEN		Öffnet Kommunikationskanal und Port
CLOSE		Schliesst Port und Kommunikationskanal
READ		Liest vom Gerät (SN)
WRITE		Schreibt zum Gerät (SN)
GET		Liest vom Gerät (LN)
SET		Schreibt zum Gerät (LN)
ACTION		Führt Methode aus (LN)

Syntax SN:

```
DLMS.OPEN;

DLMS.SHORTNAME = "$1450";
DLMS.READ;
VAR readResult;
readResult = DLMS.RESULT;

DLMS.DATA = "Integer8('01')";
DLMS.WRITE;

IF DLMS.RESULT <> "success" THEN
   DLMS.CLOSE;
   SCRIPT.ERROR = "Write failed";
   SCRIPT.ABORT;
ELSE
   DLMS.DATA = "OctetString[12]('07D0010A010F3B39FF800000')";
   DLMS.WRITE;
END;

DLMS.CLOSE;
```

Syntax LN:

```
DLMS.OPEN;

DLMS.LN = "0000010000FF"; DLMS.CLASS = 8; DLMS.ATTRIB = 2;
DLMS.DATA = "OctetString('07DB011A03091517FF8000FF')";
DLMS.SET;
MSG.TEXT = "Set returns: " + DLMS.RESULT; MSG.RUN;
IF DLMS.RESULT <> "success" THEN
        SCRIPT.ERROR = "DLMS set failure.";
        SCRIPT.ABORT;
END;

DLMS.LN = "0000010000FF"; DLMS.CLASS = 8; DLMS.ATTRIB = 2;
DLMS.GET;
MSG.TEXT = "Get returns: " + DLMS.RESULT; MSG.RUN;

DLMS.CLOSE;
```