HomeMatic

Homematic Script-Dokumentation

Teil 1: Sprachbeschreibung

Version 2.3

Inhaltsverzeichnis

1 Einleitung 4

2 Allgemeines 5

2.1 Schreibweisen im Script 5

2.2 Kommentare im Script 5

3 Script-Variablen 6

3.1 Variablen-Bezeichnungen 6

3.2 Deklaration und Initialisierung von Variablen 6

3.3 Dynamische Typbindung 7

4 Operatoren 8

4.1 Rangfolge 9

4.2 Besonderheiten bei komplexen Ausdrücken 9

5 Kontrollstrukturen 10

5.1 Bedingungen – if 10

5.2 Schleifen – while 10

5.3 Iteratoren – foreach 11

5.4 Abbruch – quit 12

6 Primitive Datentypen 13

6.1 Allgemeine Methoden 13

6.1.1 Datentyp bestimmen – VarType / Type 13

6.1.2 In Zeichenkette umwandeln – ToString 14

6.1.3 In Ganzzahl umwandeln – ToInteger 15

6.1.4 In Zeitpunkt umwandeln – ToTime 15

6.2 Boolesche Werte – boolean 16

6.3 Vorzeichenbehaftete Ganzzahlen – integer 16

6.4 Gleitkommazahlen – real 16

6.5 Zeitpunkte – time 16

6.5.1 Zugriff auf einzelne Elemente 17

6.5.2 Formatierte Ausgabe – Format 18

6.6 Zeichenketten – string 20
6.6.1 Escape-Sequenzen 20
6.6.2 In Gleitkommazahl umwandeln – ToFloat 20
6.6.3 Länge bestimmen – Length 21
6.6.4 Teilzeichenfolge ermitteln – Substr 21
6.6.5 Ermitteln einer Teilzeichenfolge – Find / Contains / StartsWith / EndsWith 21
6.6.6 Auf Iteration vorbereiten – Split 22
6.6.7 Listenelement ermitteln – StrValueByIndex 22
6.6.8 URI konforme Stringkodierung – UriEncode/UriDecode (ab 2.29.22) 23
6.6.9 UTF8 / Latin Stringkodierung – ToUTF8 / ToLatin (ab 2.29.22) 23
6.6.10 Konvertierung in Groß-/Kleinschreibung – ToUpper / ToLower (ab 2.29.22) 23
6.6.11 Trimmen von Zeichenketten – Trim / LTrim / RTrim (ab 2.29.22) 24
6.6.12 Ersetzen von Teilzeilenfolgen – Replace (ab 2.29.22) 24
7 Scriptumgebung 25
7.1 Selbstbezug - $this$ 25
7.2 Quelle - src 25
8 Mathematische Funktionen 26
8.1 Funktionen 26
8.2 Konstanten 27
9 Zusatzfunktionen 29
9.1 Zufallszahlen 29

1 Einleitung

Mit der Homematic Zentrale ist es möglich, als Reaktion auf ein Ereignis ein Script auszuführen. Hier wird eine eigene Scriptsprache verwendet, die im Folgenden als Homematic Script bezeichnet wird.

Homematic Script wird bei der Homematic Zentrale verwendet. Mit dieser Scriptsprache kann auf die Logikschicht der Homematic Zentrale zugegriffen werden. Dadurch lassen sich z.B. angelernte Homematic Komponenten bedienen.

Ziel der Homematic Script Dokumentation ist es, dem Anwender einen Einblick in die Programmierung von Scripten zu geben, die mittels Programmen von der Homematic Zentrale benutzt werden können.

Die Homematic Script Dokumentation besteht aus den folgenden 4 Dokumenten:

- Teil 1: Sprachbeschreibung
  Beschreibt die Scriptsprache Homematic Script.

- Teil 2: Objektmodell
  Beschreibt auszugsweise das Objektmodell, welches der Homematic Zentrale zugrunde liegt.

- Teil 3: Beispiele
  Eine Sammlung von Beispielen, welche den Umgang mit Homematic Script praxisnah illustrieren.

- Teil 4: Datenpunkte
  Gibt einen Überblick über die verfügbaren Datenpunkte der Homematic Geräte.

Bei diesem Dokument handelt es sich um Teil 1, die Homematic Sprachbeschreibung. Hier wird die Syntax der Scriptsprache beschrieben und auf allgemeine Besonderheiten im Umgang mit der Sprache hingewiesen.

Abschließend wird in diesem Dokument auf die Umgebung eingegangen, in welcher die Scripte innerhalb der Homematic Zentrale ablaufen.

2 Allgemeines

Programme innerhalb der Homematic Zentrale werden verwendet, um auf bestimmte Ereignisse, z.B. auf das Auslösen eines Kanals oder einer Zeitsteuerung, zu reagieren. Eine solche Reaktion kann z.B. die Änderung eines Kanalzustandes oder auch einer Systemvariable sein.

Üblicherweise werden Programme in der Zentrale auf grafische Weise erstellt. Es kann jedoch zu Situationen kommen, in denen eine grafische Darstellung nicht zweckmäßig ist. Vor allem wenn die Komplexität der Reaktionen zunimmt, ist eine alternative Darstellungsform wünschenswert.

Aus diesem Grund ist es möglich, als Reaktion auf ein Ereignis ein Script auszuführen – das Homematic Script. Homematic Script erlaubt den Zugriff auf die gesamte Logikschicht der Homematic Zentrale. So lassen sich sämtliche angelernten Geräte über Homematic Script fernsteuern. Es ist jedoch nicht möglich, Geräte über Homematic Script anzulernen, zu löschen oder in umfassender Weise zu konfigurieren.

2.1 Schreibweisen im Script

Homematic Script ist Case-Sensitive, d.h. die Scriptsprache unterscheidet zwischen Groß- und Kleinschreibung. Die Bezeichnungen „test“, „Test“ und „TEST“ stehen für drei verschiedene Symbole.

2.2 Kommentare im Script

Kommentare werden in Homematic Script durch ein Anführungszeichen („!“) eingeleitet. Sämtliche Kommentare sind einzeilig; sie beginnen mit dem Ausführungszeichen und gehen bis zum Ende der aktuellen Zeile.

Beispiel:

```
! Dies ist ein Kommentar.
string MyString = "Hallo Welt!"; ! Dies ist ebenfalls ein Kommentar.
```

3 Script-Variablen

Homematic Script kennt eine Menge von primitiven Datentypen, welche in der folgenden Tabelle dargestellt sind:

Datentyp	Wertebereich	Kurzbeschreibung
boolean	{true, false}	Boolescher Wert
integer	$[-2^{31}; 2^{31-1}]$	Vorzeichenbehaftete Ganzzahl
real	$\pm 1{,}7*10^{\pm 308}$, 15 signifikante Dezimalstellen	Gleitkommazahl
string	ISO-8859-1, null-terminiert	Zeichenkette
time	[01.01.1970; 01.01.2037], Sekundengenau	Datum und Uhrzeit
var		Untypisierte Variable
object		Referenztyp

Die hier genannten Datentypen werden im Abschnitt 0 „Primitive Datentypen“ eingehend beschrieben.

Die Typen „var“ und „object“ stellen besondere Typen dar. Während „var“ eine noch untypisierte Variable kennzeichnet (siehe Abschnitt 3.3 „Dynamische Typbindung“), handelt es sich bei den Variablen vom Typ „object“ um Referenzen innerhalb des Homematic Objektmodells.

Das Homematic Objektmodell ist in der Homematic Script Dokumentation Teil 2: Objektmodell beschrieben.

3.1 Variablen-Bezeichnungen

Für die Bezeichnung von Variablen können die Buchstaben des englischen Alphabets, Ziffernzeichen und Unterstrich „_“ verwendet werden. Ein Variablename darf allerdings nicht mit einer Ziffer beginnen.

3.2 Deklaration und Initialisierung von Variablen

Variablen können prinzipiell an jeder Stelle deklariert werden. Dabei sind sie ab der Deklaration bis zum Ende des Scripts sichtbar. Eine Deklaration kann wahlweise mit oder ohne Wertzuweisung erfolgen. Falls bei der Deklaration keine Initialisierung stattfindet, wird der betreffenden Variable ein Standardwert zugewiesen.

Beispiel:

```
integer i;        ! Deklaration ohne Initialisierung
integer j = 1;    ! Deklaration mit Initialisierung
```

3.3 Dynamische Typbindung

Homematic Script verwendet durchgehend eine dynamische Typbindung, d.h. Variablen sind nicht durch die Deklaration an einen festen Datentyp gebunden sondern können während der Ausführung ihren Datentyp wechseln. Praktisch kann sich der Typ einer Variablen mit jeder Zuweisung ändern.

Beispiel:

```
integer myVar = 1;                          ! myVar ist eine Ganzzahl
myVar = true; !                             myVar ist ein boolescher Wert
myVar = "Hallo Welt!";                      ! myVar ist eine Zeichenkette
myVar = 1.0;                                ! myVar ist eine Gleitkommazahl
myVar = @2001-01-01 00:00:00@;              ! myVar ist ein Zeitpunkt
```

Die Regel, nach der jede Zuweisung den Typ einer Variablen ändern kann, gilt auch für die Deklaration einer Variablen mit gleichzeitiger Initialisierung.

Beispiel:

```
integer i = "Hallo Welt!"; ! i ist eine Zeichenkette
```

In diesem Beispiel wird eine Variable i zunächst als Ganzzahl angelegt. Anschließend wird ihr der Wert "Hallo Welt!" zugewiesen. Damit ändert sich der Datentyp und i ist eine Zeichenkette.

4 Operatoren

Die folgende Tabelle zeigt eine Liste von Operatoren, die unter Homematic Script zur Verfügung stehen:

Operator	Zulässige Datentypen	Kurzbeschreibung	Beispiel
=	alle	Zuweisung	var i = 1;
+	integer, real, string, time	Addition / Konkatenation	i = i +1;
-	integer, real, time	Subtraktion	i = i -1;
*	integer, real	Multiplikation	i = i * 10;
/	integer, real	Division	i = i / 10;
==	boolean, integer, real, string, time	Gleichheit	boolean b = (i == 1);
<>, !=	boolean, integer, real, string, time	Ungleichheit	b = (i <> 1);
<	integer, real, string, time	kleiner	b = (i < 1);
<=	integer, real, string, time	kleiner oder gleich	b = (i <= 1);
>	integer, real, string, time	größer	b = (i > 1);
>=	integer, real, string, time	größer oder gleich	b = (i >= 1);
&&	boolean	logisches UND	b = (b && true);
||	boolean	logisches ODER	b = (b || true);
!	boolean	logischen NICHT	b = !b;
&	integer	bitweises UND	i = i & 1;
|	integer	bitweises ODER	i = i | 1;
#	string	Zeichenketten-konkatenation	string s = "Hallo" # "Welt";
.	object	Methodenzugriff.	b = system.IsVar("i");
%	integer, real	Modulo (Ganzzahliger Rest einer Division)	i = i % 3;

4.1 Rangfolge

In Homematic Script existiert keine natürliche Rangfolge von Operatoren. Aus der Mathematik bekannte Regeln wie z.B. „Punktrechnung geht vor Strichrechnung“ finden keine Anwendung. Vielmehr werden Ausdrücke einfach von rechts nach links berechnet. Um die Reihenfolge der Abarbeitung zu beeinflussen, können Klammern verwendet werden.

Beispiel:

```
integer i = 1 + 2 * 3;   ! i = (3 * 2) + 1 = 7
integer j = 3 * 2 + 1;   ! j = (1 + 2) * 3 = 9
integer k = (3 * 2) + 1;          ! k = 1 + (3 * 2) = 7
```

4.2 Besonderheiten bei komplexen Ausdrücken

Bei komplexen Ausdrücken treten im Rahmen von Homematic Script besondere Effekte auf die hier zusammengefasst sind.

Ausdrücke werden von rechts nach links aufgelöst. Da es bei Operatoren keine Rangfolge gibt, können zusätzlich Klammern verwendet werden.

Der Datentyp einer Operation wird jeweils durch den linken Operanden bestimmt. Dies hat ebenfalls Auswirkungen auf Ausdrücke, die mit unterschiedlichen Datentypen arbeiten.

Beispiel:

```
var x = 1 / 10.0; ! x = 0; x ist eine Ganzzahl
var y = 1.0 / 10; ! y = 0.1; y ist eine Gleitkommazahl
```

Dieses Verhalten kann zu Schwierigkeiten führen, wenn Ganzzahlen und Gleitkommazahlen gemischt werden. In einem solchen Fall kann das Umstellen der Operanden eine Lösung sein.

Beispiel:

```
var a = 3 * 2.5; ! a = 6; a ist eine Ganzzahl
var b = 2.5 * 3; ! b = 7.5; b ist eine Gleitkommazahl
```

Um ein Ergebnis bzw. Teilergebnis vom Typ „real“ zu erzwingen, kann 0.0 auf der linken Seite addiert werden.

Beispiel:

```
var c = 0.0 + 3; ! c = 3.0; c ist eine Gleitkommazahl
```

5 Kontrollstrukturen

Um den Programmablauf zu beeinflussen, existieren in Homematic Script verschiedene Kontrollstrukturen die im Folgenden beschreiben sind.

5.1 Bedingungen – if

```
if ( <boolescher Ausdruck> ) { <Anweisungsblock> }
[ else { <Anweisungsblock } ] [ elseif ( <boolescher Ausdruck> ) { <Anweisungsblock> } ]
```

Mit Hilfe des Schlüsselwortes „if“ können Anweisungsblöcke bedingt ausgeführt werden. Hinter dem „if“ steht in runden Klammern ein boolescher Ausdruck. Falls dieser Ausdruck den Wert „true“ liefert, wird der Anweisungsblock im Körper der if-Klausel ausgeführt. Die geschweiften Klammern um den Anweisungsblock sind dabei obligatorisch und dürfen nicht weggelassen werden.

Optional kann auf den bedingten Anweisungsblock auch ein „else“-Pfad folgen. Dieser wird ausgeführt, falls die Bedingung den Wert „false“ ergibt. Auch hier sind die geschweiften Klammern obligatorisch.

Seit Firmware 2.29.22 existiert die Möglichkeit auf den bedingten Anweisungsblock alternativ ein „elseif()“-Pfad folgen zu lassen, sodass die darauf folgenden Anweisungen erst ausgeführt werden wenn der boolesche Ausdruck des „elseif()“ Teils den Wert „true“ besitzt.

Beispiel:

```
integer i = 1;
string s;
if (i == 1) { s = "i == 1"; }
elseif (i == 2) { s = “i == 2”; }
else { s = "i != 1 && i != 2"; }
```

5.2 Schleifen – while

```
while ( <boolescher Ausdruck> ) { <Anweisungsblock> }
```

Bei der „while“-Schleife handelt es sich um eine eingangsgeprüfte Schleife, d.h. vor jeder Iteration wird der boolesche Ausdruck geprüft. Der Schleifenkörper wird so lange ausgeführt, bis der Wert des booleschen Ausdrucks „false“ wird.

Die Schleife lässt sich darüber hinaus mittels "break"-Befehl aus dem Schleifenkörper heraus beenden (vgl. Beispiel 2) oder mittels „continue“-Befehl fortsetzen (vgl. Beispiel 3).

Nach spätestens 500000 Iterationen nimmt der Script-Interpreter an, dass die betreffende „while“-Schleife nicht terminiert. In diesem Fall wird die „while“-Schleife ebenfalls verlassen.

Beispiel 1:

```
integer i = 0;
while (true) { i = i + 1; }
! i = 500001
```

Beispiel 2:

```
integer i = 0;
while (true) {
    i = i + 1;
    if( i == 10) { break; }
}
! i = 10
```

Beispiel 3:

```
integer i = 0;
while (i < 3) {
    i = i + 1;
    if(i == 1) { continue; }
    Write(i);
}
! Ausgabe: 23
```

5.3 Iteratoren – foreach

```
foreach (<Indexvariable>, <Liste>) { <Anweisungsblock> }
```

Mit Hilfe der „foreach“-Schleife kann durch eine Liste iteriert werden. Bei der Liste handelt es sich um eine speziell formatierte Zeichenkette. In jeder Iteration wird der Indexvariablen ein Wert aus der Liste zugewiesen. Auf diese Weise wird die komplette Liste durchlaufen.

Die einzelnen Elemente der Indexliste sind durch Tabulatoren voneinander getrennt. Die Indexvariable selbst muss zwingend eine Zeichenkette sein.

Wie die „while“-Schleife, lässt sich auch die „foreach“-Schleife mittels „break“-Befehl beenden oder mittels „continue“-Befehl fortsetzen.

Wie bei while() lässt die Nutzung von foreach() eine Schleife nach max. 500000 Iterationen abbrechen, damit sichergestellt ist, dass eine versehentliche Endlosschleife nicht die komplette Ausführung des Script-Interpreters lahmlegen kann.

Beispiel:

```
string liste       = "a\tb\tc";              ! Liste { "a", "b", "c" }
string ausgabe = "";                         ! Ausgabe
string index;                                ! Indexvariable
foreach (index, liste) {
  ausgabe = index # ausgabe;
}
! ausgabe = "cba";
```

5.4 Abbruch – quit

```
quit;
```

Mit dem Schlüsselwort „quit“ lassen sich Scripte vorzeitig abbrechen.

Beispiel:

```
integer even = 3;
if (even & 1) { quit; } ! "even" ist nicht gerade -> Abbruch
```

6 Primitive Datentypen

Homematic Script kennt eine Reihe von primitiven Datentypen, welche in der folgenden Tabelle zusammengefasst sind:

Datentyp	Kurzbeschreibung
var	Untypisierter Datentyp
boolean	Boolescher Wert
integer	Vorzeichenbehaftete Ganzzahl
real	Gleitkommazahl
string	Zeichenkette
time	Zeitpunkt
object	Objekt des Homematic Objektmodell
idarray	Identifier Array

Primitive Datentypen können als Objekte verstanden werden, d.h. sie besitzen Methoden, über die man auf ihren Inhalt zugreifen kann.

Für den Datentyp „var“, „object“ und „idarray“ sind ab Firmware 2.29.18 die Methoden „VarType()“ und „Type()“ wie im Folgenden beschrieben anwendbar.

6.1 Allgemeine Methoden

Jeder primitive Datentyp verfügt über einen Satz von allgemeinen Methoden. Über die Methoden kann der Typ einer Variablen zur Laufzeit bestimmt werden. Außerdem ist es möglich, Werte eines primitiven Datentyps in einen anderen Typ zu konvertieren.

6.1.1 Datentyp bestimmen – VarType / Type

```
integer boolean.VarType();
integer integer.VarType();
integer real.VarType();
integer time.VarType();
integer string.VarType();
```

Mit dieser Methode kann der Datentyp einer Variablen bestimmt werden. Das Ergebnis ist eine Ganzzahl. Die folgende Tabelle liefert eine Zuordnung zwischen den Datentypen und den entsprechenden Zahlenkonstanten:

VarType()	Type() – Datentyp
0	var
1	boolean
2	integer
3	real
4	string
5	time
9	object
10	idarray

```
boolean b;
integer type = b.VarType(); ! type = 1;
```

6.1.2 In Zeichenkette umwandeln – ToString

```
integer boolean.ToString();
integer integer.ToString();
integer real.ToString(p);
integer time.ToString(s);
integer string.ToString(p);
```

Wandelt die Variable in eine Zeichenkette um. Bei Anwendung auf eine Gleitkommazahl findet eine implizite Rundung auf die sechste Nachkommastelle statt.

Bei einer Gleitkommazahl (Typ ‚real'), sowie einer Zeichenkette (Typ ‚string') ist es möglich, unter Angabe des optionalen Parameters ‚p' den momentanen Wert zur nächstliegenden Zahl mit Genauigkeit p zu runden, bevor diese dann in eine Zeichenkette konvertiert wird (äquivalent Round()).

Beispiel:

```
var i = 1.23456;
var s = i.ToString(3); ! s = "1.235"; s ist eine Zeichenkette
var r = s.ToString(1); ! r = "1.2"; r ist eine Zeichenkette
```

Bei der Anwendung von ToString() auf eine Zeitvariable vom Typ ‚time' ist es möglich, einen optionalen Parameter ‚s' anzugeben. Dies entspricht dann einem direkten Aufruf der time.Format() Funktion wie unter Kapitel 6.5.2.

Beispiel:

```
time t = @2008-12-24 18:30:00@;
string sDate = t.ToString("%d.%m.%Y"); ! sDate = "24.12.2008";
```

6.1.3 In Ganzzahl umwandeln – ToInteger

```
integer boolean.ToInteger();
integer integer.ToInteger();
integer real.ToInteger(p);
integer time.ToInteger();
integer string.ToInteger();
```

Wandelt eine Variable in eine Ganzzahl um. Bei Anwendung auf Gleitkommazahlen findet eine Rundung auf eine Genauigkeit auf die sechste Nachkommastelle statt, bevor in eine Ganzzahl konvertiert wird. Zusätzlich kann ein optionaler Parameter p angegeben werden, sodass diese Genauigkeit explizit angegeben werden kann bevor eine Wandlung stattfindet.

Beispiel:

```
var s = "100";
var s = s.ToInteger(); ! i = 100; i ist eine Ganzzahl
```

6.1.4 In Zeitpunkt umwandeln – ToTime

```
integer boolean.ToTime();
integer integer.ToTime();
integer real.ToTime();
integer time.ToTime();
```

```
integer string.ToTime();
```

Wandelt eine Variable in einen Zeitpunkt um.

Beispiel:

```
var i = 1;
var t = i.ToTime(); ! t = @1970-01-01 01:00:01@
```

6.2 Boolesche Werte – boolean

```
boolean bTRUE = true;
boolean bFALSE = false;
```

Boolesche Variablen können die Werte „true“ und „false“ annehmen.

6.3 Vorzeichenbehaftete Ganzzahlen – integer

```
integer i = -123;
```

Vorzeichenbehaftete Ganzzahlen liegen im Bereich von -2^{31} bis 2^{31-1}.

6.4 Gleitkommazahlen – real

```
real r = 1.0;
real r = "-1.0E-1".ToFloat(); ! -0.1
```

Gleitkommazahlen besitzen 15 signifikante Dezimalstellen. Sie liegen im Wertebereich von $\pm 1{,}7*10^{\pm 308}$.

6.5 Zeitpunkte – time

```
time t = @2008-12-24 18:00:00@;
```

Der Datentyp „time“ bezeichnet Zeitpunkte zwischen dem 01. Januar 1970 und dem 01. Januar 2037. Dabei kann ein Zeitpunkt sekundengenau angegeben werden.

6.5.1 Zugriff auf einzelne Elemente

Über spezielle Methoden kann auf die einzelnen Felder eines Zeitpunktes zugegriffen werden. Die folgende Tabelle zeigt eine Übersicht der verfügbaren Methoden:

Methode	Kurzbeschreibung
integer time.Year();	Ermittelt die Jahreszahl
integer time.Month();	Ermittelt die Monatszahl
integer time.Day();	Ermittelt den Monatstag
integer time.Hour();	Ermittelt die Stunde
integer time.Minute();	Ermittelt die Minute
integer time.Second();	Ermittelt die Sekunde
integer time.Week();	Ermittelt die Wochennummer
integer time.Weekday();	Ermittelt die Nummer des Wochentags
integer time.Yearday();	Ermittelt die Nummer des Tages im Jahr
integer time.IsLocalTime();	Ermittelt, ob der Zeitpunkt in Lokalzeit (1) oder Weltzeit (0) angegeben ist
integer time.IsDST();	Ermittelt, ob der Zeitpunkt in der lokalen Sommerzeit (engl: daylight-saveing time) liegt (1) oder nicht (0).

```
time t                  = @2008-12-24 18:30:00@;

integer year            = t.Year();         ! year = 2008
integer month           = t.Month();        ! month = 12
integer day             = t.Day();          ! day = 24
integer hour            = t.Hour();         ! hour = 18
integer minute          = t.Minute();       ! minute = 30
integer second          = t.Second();       ! second = 0
integer week            = t.Week();         ! week = 51
integer weekday         = t.Weekday();      ! weekday = 4
integer yearday         = t.Yearday();      ! yearday = 359
integer isLocalTime     = t.IsLocalTime();  ! isLocalTime = 1
integer isDST           = t.IsDST();        ! isDST = 0
```

6.5.2 Formatierte Ausgabe – Format

```
string time.Format(string formatString);
```

Formatiert den Zeitpunkt nach den Angaben in „formatString“ und gibt das Ergebnis als Zeichenkette zurück.

Bei dem Parameter „formatString“ handelt es sich um eine Zeichenkette, die Platzhalter für diverse Elemente des Zeitpunktes enthalten kann. Folgende Platzhalter stehen zur Verfügung:

Platzhalter	**Kurzbeschreibung**	**Beispiel (24.12.2008 18:30:00)**
%%	Das Prozent-Zeichen	„%“
%a	Abgekürzter Wochentagsname	Wed
%A	Vollständiger Wochentagsname	Wednesday
%b	Abgekürzter Monatsname	Dec
%B	Vollständiger Monatsname	December
%c	Datum und Uhrzeit	Wed Dec 24 18:30:00 2008
%C	Jahrhundert – 1	20
%d	Monatstag	24
%D	Datum %m/%d%y	12/24/08
%F	Datum %Y-%m-%d	2008-12-24
%h	Abgekürzter Monatsname	Dec
%H	Stunde (24-Stunden-Uhr)	18
%I	Stunde (12-Stunden-Uhr)	06
%j	Nummer des Tages im Jahr	359
%m	Monatsnummer	12
%M	Minute	30
%n	Neue-Zeile-Steuerzeichen	
%p	AM bzw. PM	PM
%r	Uhrzeit (12-Stunden-Uhr)	06:30:00 PM
%S	Sekunde	00
%t	Tabulator-Steuerzeichen	
%T	Uhrzeit (24-Stunden-Uhr)	18:30:00
%u	Wochentag (Montag = 1)	3

%U	Wochennummer (Woche 1 ab dem 1. Sonntag im Januar)	51
%V	Wochennummer (ISO 8601)	52
%w	Wochentag (Sonntag = 0)	3
%W	Wochennummer (Woche 1 ab dem 1. Montag im Januar)	51
%x	Datum	12/24/08
%X	Uhrzeit	18:30:00
%y	Jahreszahl (2 Ziffern)	08
%Y	Jahreszahl (4 Ziffern)	2008
%z	Zeitabstand zu GMT	+0100
%Z	Name der Zeitzone	CET

```
time t = @2008-12-24 18:30:00@;
string sDate = t.Format("%d.%m.%Y"); ! sDate = "24.12.2008";
string sTime = t.Format("%H:%M:%S"); ! sTime = "18:30:00";
```

6.6 Zeichenketten – string

```
string s = "Hallo Welt!";
```

Eine Zeichenkette in Homematic Script ist nach ISO-8859-1 kodiert. Es handelt sich dabei um einen null-terminierten String. Zeichenketten werden wahlweise von einfachen oder doppelten Anführungszeichen umschlossen. Sonder- und Steuerzeichen können über Escape-Sequenzen angegeben werden.

Eine Zeichenkette kann als Liste formatiert werden. Dazu werden die einzelnen Listen- elemente durch Tabulatoren voneinander getrennt. Diese Formatierung wird u.a. von der „foreach“-Schleife verwendet.

6.6.1 Escape-Sequenzen

Die folgende Tabelle listet eine Reihe von Escape-Sequenzen auf. Diese können innerhalb von Zeichenketten verwendet werden, um Sonder- und Steuerzeichen zu kodieren.

Escape-Sequenz	**Kurzbeschreibung**
\\	Backslash (\)
\"	Doppeltes Anführungszeichen (")
\'	Einfaches Anführungszeichen (')
\t	Tabulator (ASCII 0x09)
\n	Neue Zeile (ASCII 0x0A)
\r	Wagenrücklauf (ASCII 0x0D)

6.6.2 In Gleitkommazahl umwandeln – ToFloat

```
real string.ToFloat();
```

Wandelt eine Zeichenkette in eine Gleitkommazahl um.

Beispiel:

```
string s = "1.01";
real r1 = s.ToFloat();          ! r1 = 1.01;
real r2 = "0.1".ToFloat();      ! r2 = 0.1;
real r3 = "1E-1".ToFloat();     ! r3 = 0.1;
```

6.6.3 Länge bestimmen – Length

```
integer string.Length();
```

Ermittelt die Anzahl der Zeichen einer Zeichenkette.

Beispiel:

```
string s      = "Hallo\tWelt!";
integer length = s.Length();   ! length = 11
```

6.6.4 Teilzeichenfolge ermitteln – Substr

```
string string.Substr(integer index, integer length);
```

Liefert eine Teilzeichenfolge. Dabei bezeichnet „index“ das erste Zeichen und „length“ die Länge der Teilzeichenfolge. Das erste Zeichen beginnt bei Index 0.

Beispiel:

```
string s      = "Hallo Welt!";
string world  = s.Substr(6, 4); ! world = "Welt"
```

6.6.5 Ermitteln einer Teilzeichenfolge – Find / Contains / StartsWith / EndsWith

```
integer string.Find(string key);
boolean string.Contains(string key);
boolean string.StartsWith(string key);
boolean string.EndsWith(string key);
```

Die Methode Find() ermittelt den ganzzahligen Index des ersten Vorkommens der Teilzeichenfolge „key“ in einer Zeichenkette. Wird „key“ nicht gefunden, wird „-1“ zurückgegeben.

Seit Firmware 2.29.22 existieren Methoden (Contains, StartsWith, EndsWith) die wie Find eine Teilzeichenkettensuche durchführen, jedoch als Resultat statt eines Indexes einen booleschen Wert bei (Nicht-)Auffinden der Teilzeichenkette zurückgeben. Hierbei gibt Contains() einen Wert true/false zurück wenn die angegeben Teilzeichenkette gefunden oder nicht gefunden wurde (gleichzusetzen mit „Find() >= 0“ bzw. „Find() == -1“). StartsWith() und EndsWith() geben jeweils true/false zurück, wenn die Teilzeichenkette entweder am Anfang oder Ende der Zeichenkette gefunden oder nicht gefunden wurde.

Beispiel:

```
string s = "Hallo Welt";
integer World = s.Find("Welt"); ! World = 6
integer world = s.Find("welt"); ! world = -1
boolean bWorld = s.Contains("Welt"); ! bWorld = true
boolean bStart = s.StartsWith("Hallo"); !bStart = true
boolean bEnd = s.EndsWith("Welt"); !bEnd = true
```

6.6.6 Auf Iteration vorbereiten – Split

```
string string.Split(string separator);
```

Erstellt eine Liste. Dabei werden alle Vorkommen von „separator“ durch Tabulatoren ersetzt. Die so entstandene Zeichenfolge kann von der „foreach“-Schleife durchlaufen werden.

Beispiel:

```
string summanden = "1,2,3";
integer summe          = 0;
string summand;
foreach(summand, summanden.Split(","))
{
  summe = summe + summand.ToInteger();
}
! summe = 6
```

6.6.7 Listenelement ermitteln – StrValueByIndex

```
string string.StrValueByIndex(string separator, integer index);
```

Die Zeichenkette ist eine Liste, deren Elemente durch Separatoren voneinander getrennt sind. Die Methode „StrValueByIndex“ liefert das durch den Index spezifizierte Listenelement. Die Zählung für „index“ beginnt bei 0.

Beispiel:

```
string Rezept   = "Butter,Eier,Mehl,Milch,Zucker";
string ErsteZutat = Rezept.StrValueByIndex(",", 0); ! ErsteZutat = Butter;
```

6.6.8 URI konforme Stringkodierung – UriEncode/UriDecode (ab 2.29.22)

```
string string.UriEncode();
string string.UriDecode();
```

Diese Methoden erlauben das Umkodieren einer Zeichenkette in/von einer RFC 3986 konformen Weise (%xx für nicht URI-konforme Zeichen) um diese dann in einer URL verwenden zu können bzw. von dort kommend in eine normale Zeichenkette zurück zu konvertieren.

Beispiel:

```
string str = " !\"#$%&'()";
string kodiert = str.UriEncode(); ! kodiert = %20%21%22%23%24%25%26%27%28%29
string dekodiert = kodiert.UriDecode(); ! dekodiert = !"#$%&'()
```

6.6.9 UTF8 / Latin Stringkodierung – ToUTF8 / ToLatin (ab 2.29.22)

```
string string.ToUTF8();
string string.ToLatin();
```

Die Methode ToUTF8() erlaubt eine Zeichenkette (die standardmäßig in ISO-8859-1 Kodierung vorliegt) in eine UTF8-kodierte Zeichenkette zu konvertieren. Des Weiteren kann eine Zeichenkette, die bereits im UTF8-Format vorliegt, zurück in eine standardkonforme ISO-8859 Kodierung konvertiert werden.

Beispiel:

```
string str = "Übergrößenträger";
string utf8 = str.ToUTF8(); ! utf8 = "Ã¬bergrÃ¶Ã•entrÃ¤ger“
string latin = utf8.ToLatin(); ! latin= "Übergrößenträger“
```

6.6.10 Konvertierung in Groß-/Kleinschreibung – ToUpper / ToLower (ab 2.29.22)

```
string string.ToUpper();
string string.ToLower();
```

Diese Methoden erlauben eine Zeichenkette in Groß- (ToUpper) bzw. Kleinbuchstaben (ToLower) zu konvertieren.

Beispiel:

```
string str = "AbCdEfGhI";
string upper = str.ToUpper(); ! upper = "ABCDEFGHI“
string lower = str.ToLower(); ! lower = "abcdefghi“
```

6.6.11 Trimmen von Zeichenketten – Trim / LTrim / RTrim (ab 2.29.22)

```
string string.Trim(string chars);
string string.LTrim(string chars);
string string.RTrim(string chars);
```

Diese Methoden erlauben bei einer Zeichenkette die im optionalen Argument „chars“ angegebenen Zeichen jeweils vom Anfang (LTrim), vom Ende (RTrim) oder vom Anfang und Ende (Trim) zu entfernen, bis ein Zeichen gefunden wurde, das **nicht** im angegebenen „chars“ Argument existiert (sogenanntes „trimmen“ von Zeichenketten).

Wird das optionale Argument „chars“ nicht angegeben, entfernen diese Methoden auftauchende Leerzeichen ( ), Form Feeds (\f), Zeilenvorschübe (\n bzw. LF), Wagenrückläufe (\r bzw. CR), Tabulatorzeichen (\t) oder vertikale Tabulatorzeichen (\v) aus der jeweiligen Stelle der Zeichenkette bis ein anderes Zeichen gefunden wurde.

Beispiel:

```
string str = " \tAnfang und Ende \r\n";
string trim = str.Trim();                  ! trim = "Anfang und Ende"
string ltrim = str.LTrim();                ! ltrim = "Anfang und Ende \r\n"
string rtrim = str.RTrim();                ! rtrim = " \tAnfang und Ende"
string trimc = str.Trim(" \t\nAnfang");    ! trimc = "und Ende \r"
```

6.6.12 Ersetzen von Teilzeilenfolgen – Replace (ab 2.29.22)

```
string string.Replace(string src, string dest);
```

Die Methode Replace() erlaubt es in einer Zeichenkette nach einer Teilzeichenkette „src“ zu suchen und diese innerhalb der Zeichenkette gegen die als „dest“ angegebene Ersatzteilzeichenkette zu ersetzen.

Beispiel:

```
string str = "John hates Jane";
string replaced = str.Replace("hates“, "loves"); ! replaced = "John loves Jane"
```

7 Scriptumgebung

In der Regel wird Homematic Script innerhalb von Programmen auf der Homematic Zentrale verwendet. Für diesen Einsatzzweck sind spezielle Symbole definiert, welche die Umgebung des Scripts darstellen. Die Symbole werden bei der Ausführung ersetzt.

7.1 Selbstbezug - $this$

Das Symbol „$this$“ ist in jedem Fall vorhanden. Es wird ersetzt durch die Id des Programms, in dem das Script abläuft.

Für das folgende Beispiel wurde eine Systemvariable mit dem Namen „i“ vom Typ „Zahl“ angelegt. Anschließend wurde ein Programm erstellt, welches keine Bedingung und als einzige Aktion das folgende Script besitzt:

Beispiel:

```
var i = dom.GetObject("i");
i.Variable($this$);
```

Anschließend wurde das Programm gespeichert und auf „aktiv“ gesetzt. Sobald man das Programm über die Bedienung ausführt, wird der Systemvariablen die Id des Programms zugewiesen.

7.2 Quelle - src

Das Symbol „src“ bezeichnet den Datenpunkt, welcher das Programm, in dem das Script abläuft, ausgelöst hat. Reagiert ein Programm auf mehrere Ereignisse, kann auf diese Weise herausgefunden werden, welches der Ereignisse die Ausführung ausgelöst hat.

Unter bestimmten Umständen, z.B. wenn ein Programm manuell ausgeführt wird, ist das Symbol „src“ nicht gesetzt. In diesen Fällen wird es nicht aufgelöst.

Im folgenden Beispiel wird die Systemvariable i (Typ: Zahl) auf die Id des Datenpunktes gesetzt, der das betreffende Programm angestoßen hat. Ist keine Quelle verfügbar, weil das Programm z.B. manuell ausgeführt wurde, wird i auf -1 gesetzt.

Beispiel:

```
var i = dom.GetObject("i");
var source = dom.GetObject("$src$");
if (source)      { i.Variable(source.ID()); }
else             { i.Variable(-1); }
```

Bevor das Script ausgeführt wird, werden alle Vorkommen von „src“ durch die Id des tatsächlichen Auslösers ersetzt. Wird das Programm von einem externen Ereignis angestoßen, so wird in Zeile 2 die Variable „source“ auf den Auslöser gesetzt. Wird das Programm dagegen manuell ausgeführt, existiert kein Auslöser und „src“ wird nicht ersetzt. Da es kein Objekt mit der Bezeichnung „src„ gibt, liefert „dom.GetObject“ den Wert „null“.

8 Mathematische Funktionen

Beginnend mit Firmware 2.29.18 wurden verschiedene mathematische Funktionen und Konstanten hinzugefügt die innerhalb eines Scripts verwendet werden können um komplexe mathematische Berechnungen durchzuführen.

8.1 Funktionen

Die folgende Tabelle umfasst alle mathematischen Funktionen die direkt auf eine Variable vom Type „real“ oder „integer“ gleichermaßen angewendet werden können.

Funktion	Kurzbeschreibung	Beispiel
Abs()	Absoluter Wert einer Zahl berechnen	r = -1.5; a = r.Abs(); ! a = 1.5
Mod(y)	Berechnet restlichen Anteil einer Division	r = 5.0; d = r.Mod(3); ! d = 2.0
Min(y)	Gibt den minimalen Wert zweier Zahlen zurück	r = 5.0; m = r.Min(8.0); ! m = 5.0
Max(y)	Gibt den maximalen Wert zweier Zahlen zurück	r = 5.0; m = r.Min(8.0); ! m = 8.0
Exp()	Exponentialfunktion der Basis e	r = 2.0; e = r.Exp(); ! e = 7.389056
Exp2()	Exponentialfunktion der Basis 2	r = 2.0; e = r.Exp2(); ! e = 4.0
Exp10()	Exponentialfunktion der Basis 10	r = 2.0; e = r.Exp10(); ! e = 100.0
Expm1()	Exponentialfunktion der Basis e minus 1	r = 2.0; e = r.Expm1(); ! e = 6.389056
Log()	Natürliche Logarithmusfunktion	r = 2.0; l = r.Log(); ! l = 0.693147
Log2()	Logarithmusfunktion der Basis 2	r = 2.0; l = r.Log2(); ! l = 1.0
Log10()	Logarithmusfunktion der Basis 10	r = 2.0; l = r.Log10(); ! l = 0.301030
Log1p()	Natürlicher Logarithmus plus 1	r = 2.0; l = r.Log1p(); ! l = 1.098612
Sqrt()	Quadratwurzelfunktion	r = 2.0; s = r.Sqrt(); ! s = 1.414214
Cbrt()	Kubische Wurzelfunktion	r = 2.0; s = r.Cbrt(); ! s = 1.259921
Pow(y)	Potenzfunktion	r = 2.0; p = r.Pow(2.0); ! p = 4.0
Sin()	Sinusfunktion	r = 2.0; s = r.Sin(); ! s = 0.909297
Cos()	Kosinusfunktion	r = 2.0; c = r.Cos(); ! c = -0.416147

Tan()	Tangensfunktion	r = 2.0; t = r.Tan(); ! t = -2.185040
Asin()	Arkussinus-Funktion	r = 1.0; s = r.Asin(); ! s = 1.570796
Acos()	Arkuskosinus-Funktion	r = 0.0; c = r.Acos(); ! c = 1.570796
Atan()	Arkustangens-Funktion	r = 2.0; t = r.Atan(); ! t = 1.107149
Sinh()	Sinus Hyperbolicus-Funktion	r = 2.0; s = r.Sinh(); ! s = 3.626860
Cosh()	Kosinus Hyperbolicus-Funktion	r = 2.0; c = r.Cosh(); ! c = 3.762196
Tanh()	Tangens Hyperbolicus-Funktion	r = 2.0; t = r.Tanh(); ! t = 0.964028
Asinh()	Arkussinus Hyperbolicus-Funktion	r = 2.0; s = r.Asinh(); ! s = 1.443635
Acosh()	Arkuskosinus Hyperbolicus-Funktion	r = 2.0; c = r.Acosh(); ! c = 1.316958
Atanh()	Arkustangens Hyperbolicus-Funktion	r = 0.5; t = r.Atanh(); ! t = 0.549306
Ceil()	Kleinster ganzzahligen Wert, der nicht kleiner als der momentane Wert ist	r = 0.4521; c = r.Ceil(); ! c = 1.0
Floor()	Größte ganze Zahl, die nicht größer als der momentane Wert ist	r = 0.4521; f = r.Floor(); ! f = 0.0
Trunc(p)	Rundet den momentanen Wert mit Genauigkeit p in Richtung 0	r = 0.4521; t = r.Trunc(1); ! t = 0.4
Round(p)	Rundet den momentanen Wert zur nächstliegenden Zahl mit Genauigkeit p	r = 0.4521; o = r.Round(1); ! o = 0.5

8.2 Konstanten

Die folgende Tabelle listet die innerhalb eines Scripts zur Verfügung stehenden mathematischen Konstanten die anstelle von Gleitkomma- und Ganzzahl variablen verwendet werden können.

Konstante	**= Mathematische Operation**	**Absoluter Wert**
M_E	e = „1“.Exp()	2.71828182845904523536
M_LOG2E	e.Log2()	1.44269504088896340736
M_LOG10E	e.Log10()	0.434294481903251827651
M_LN2	„2“.Log()	0.693147180559945309417
M_LN10	„10“.Log()	2.3025850929940456840 2
M_PI	pi	3.14159265358979323846
M_PI_2	pi / 2	1.57079632679489661923

M_PI_4	pi / 4	0.78539816339744830961 6
M_1_PI	1 / pi	0.31830988618379067153 8
M_2_PI	2 / pi	0.63661977236758134307 6
M_2_SQRTPI	2 / pi.Sqrt()	1.12837916709551257390
M_SQRT2	„2“.Sqrt()	1.41421356237309504880
M_SQRT1_2	1 / „2“.Sqrt()	0.70710678118654752440 1

Beispiel:

```
var r = 4.2; ! Kreisradius in cm
var A = (M_PI * r.Pow(2)).Round(2); ! Kreisfläche A =  pi * r^2 = 55.42 cm
```

9 Zusatzfunktionen

Beginnend mit Firmware 2.29.18 wurden verschiedene systemweit gültige Funktionen hinzugefügt die sich für die Programmierung komplexer Scripts als nützlich herausgestellt haben. Diese sollen im Folgenden aufgelistet werden.

9.1 Zufallszahlen

Für die Berechnung von Zufallszahlen existieren die folgenden Funktionen:

Funktion	Beschreibung	Beispiel
system.Random(min, max)	Berechnung einer ganzzahligen Zufallszahl mit der optionalen Angabe eines akzeptieren Bereiches	var dice = system.Random(1, 6); ! dice = 3
system.Srandom(int)	Erlaubt den ‚Seed' des Zufallsgenerators auf einen definierten Wert zu setzen (Nur für Debugzwecke)	var seed = system.Random(); system.Srandom(seed);

Bevollmächtigter des Herstellers:
Manufacturer's authorised representative:

eQ-3

eQ-3 AG
Maiburger Straße 29
26789 Leer / GERMANY
www.eQ-3.de