



HomeMatic-Script Dokumentation

Teil 1: Sprachbeschreibung

Inhaltsverzeichnis

1	Einleitung	3
2	Allgemeines.....	3
2.1	Schreibweisen im Skript	4
2.2	Kommentare im Skript.....	4
3	Skript-Variablen	5
3.1	Variablen-Bezeichnungen	5
3.2	Deklaration und Initialisierung von Variablen	5
3.3	Dynamische Typbindung	6
4	Operatoren.....	7
4.1	Rangfolge	8
4.2	Besonderheiten bei komplexen Ausdrücken	8
5	Kontrollstrukturen	9
5.1	Bedingungen – if.....	9
5.2	Schleifen – while	9
5.3	Iteratoren – foreach.....	10
5.4	Abbruch – quit	10
6	Primitive Datentypen.....	11
6.1	Allgemeine Methoden.....	11
6.2	Boolesche Werte – boolean.....	13
6.3	Vorzeichenbehaftete Ganzzahlen – integer	13
6.4	Gleitkommazahlen – real	13
6.5	Zeitpunkte – time	13
6.6	Zeichenketten – string	17
7	Skriptumgebung	20
7.1	Selbstbezug - \$this\$	20
7.2	Quelle - \$src\$	20

1 Einleitung

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

HomeMatic-Script wird bei der HomeMatic Zentrale verwendet. Mit dieser Skriptsprache kann auf die Logikschicht der HomeMatic Zentrale zugegriffen werden. Dadurch lassen sich z.B. angelegte HomeMatic Komponenten bedienen.

Ziel der HomeMatic-Script Dokumentation ist es, dem Anwender einen Einblick in die Programmierung von Skripten 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 Skriptsprache 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 Skriptsprache beschrieben und auf allgemeine Besonderheiten im Umgang mit der Sprache hingewiesen.

Abschließend wird in diesem Dokument auf die Umgebung eingegangen, in welcher die Skripte 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.

HomeMatic-Script Dokumentation

Aus diesem Grund ist es möglich, als Reaktion auf ein Ereignis ein Skript 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 Skript

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

2.2 Kommentare im Skript

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 Skript-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	$\pm 1,7 \cdot 10^{\pm 308}$, 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 Variablenname 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 Skripts sichtbar. Eine Deklaration kann wahlweise mit oder ohne Wertzuweisung erfolgen. Falls bei der Deklaration keine Initialisierung stattfindet, wird der betreffenden Variable ein Standardwert zugewiesen.

HomeMatic-Script Dokumentation

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 = (>= 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.lsVar("i");

HomeMatic-Script Dokumentation

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> } ]
```

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.

Beispiel:

```
integer i = 1;  
string s;  
if (i == 1) { s = "i == 1"; }  
else      { s = "i != 1"; }
```

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.

Nach spätestens 5.000 Iterationen nimmt der Skript-Interpreter an, dass die betreffende „while“-Schleife nicht terminiert. In diesem Fall wird die „while“-Schleife ebenfalls verlassen.

Beispiel:

```
integer i = 0;  
while (true) { i = i + 1; }  
! i = 5001
```

HomeMatic-Script Dokumentation

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.

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 Skripte 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
boolean	Boolescher Wert
integer	Vorzeichenbehaftete Ganzzahl
real	Gleitkommazahl
time	Zeitpunkt
string	Zeichenkette

Neben diesen primitiven Datentypen existieren die speziellen Datentypen „var“ und „object“. Bei „var“ handelt es sich um einen untypisierten Wert; „object“ bezeichnet ein Objekt im HomeMatic Objektmodell. Für diese beiden Typen gelten die hier beschriebenen Eigenschaften nicht.

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

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

```
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()	Datentyp
-----------	----------

HomeMatic-Script Dokumentation

1	boolean
2	integer
3	real
5	time
4	string

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

6.1.2 In Zeichenkette umwandeln – ToString

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

Wandelt die Variable in eine Zeichenkette um.

Beispiel:

```
var i = 1;  
var s = i.ToString(); ! s = "1"; s ist eine Zeichenkette
```

6.1.3 In Ganzzahl umwandeln – ToInteger

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

Wandelt eine Variable in eine Ganzzahl um.

Beispiel:

```
var s = "100";  
var i = 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; ! -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:

HomeMatic-Script Dokumentation

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

HomeMatic-Script Dokumentation

%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 Listenelemente 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;
```

HomeMatic-Script Dokumentation

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 Index einer Teilzeichenfolge ermitteln – Find

```
integer string.Find(string key);
```

Ermittelt den Index des ersten Vorkommens der Zeichenkette „key“ in einem String. Wird „key“ nicht gefunden, wird „-1“ zurückgegeben.

Beispiel:

```
string s = "Hallo Welt";  
integer World = s.Find("Welt"); ! World = 6  
integer world = s.Find("welt"); ! world = -1
```

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;
```

7 Skriptumgebung

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 Skripts 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 Skript 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 Skript 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 Skript 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 Skript 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

Teil 1: Sprachbeschreibung



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“.

