HomeMatic

Homematic Script Documentation

Part 1: Script Language description

Copy from Version 2.2

English Version 0.2

This file is not an official version - it was created with the help of Google Translate as a Fan Project.

Table of Contents

1 Introduction

With the Homematic central it is possible to execute a script in response to an event. Here, a custom scripting language is used, hereinafter referred to as the Homematic script.

Homematic script is used at the Homematic CCU. With this scripting language, the logic layer of the Homematic Central can be accessed. This allows, for example, Use Homematic components.

The aim of the Homematic script documentation is to give the user an insight into the programming of scripts that can be used by programs from the Homematic Central Station.

The Homematic script documentation consists of the following 4 documents:

- Part 1: Language description
  Describes the scripting language Homematic script.

- Part 2: Object model
  Describes in part the object model, which is based on the Homematic Central.

- Part 3: Examples
  A collection of examples illustrating the handling of Homematic script in a practical way.

- Part 4: Data points
  Provides an overview of the available data points of the Homematic devices.

This document is part 1, the Homematic language description. Here, the syntax of the scripting language is described and pointed out to general peculiarities in the use of languages.

Finally, this document describes the environment in which the scripts run within the Homematic CCU.

2 General Information

Programs within the Homematic Center are used to respond to certain events, such as the triggering of a channel or a time control. Such a reaction can be, for example, the change of a channel state or even a system variable.

Typically, programs are created at the CCU in a graphical manner. However, there may be situations where a graphical representation is not appropriate . Especially when the complexity of the reactions increases, an alternative form of presentation is desirable.

For this reason, it is possible to execute a script in response to an event the Homematic script. Homematic script allows access to the entire logic layer of the Homematic CCU. This means that all fishing equipment can be controlled remotely via Homematic script. However, it is not possible to learn, delete or configure devices using Homematic script in a comprehensive way.

2.1 S*pelling* methods in the script

Homematic script is case-sensitive, i.e. the scripting language differs from upper-and lower-case. The names "test", "Test" and "test" are available for three different symbols.

2.2 Comments in script

Comments are initiated in Homematic script by a quotation mark ("!"). All comments are single line; You start with the execution character and go to the end of the current line.

Example:

```
! This is a comment.

string MyString = "Hello world!"; ! This is also a comment.
```

3 Script variables

Homematic script knows a lot of primitive data types, which are shown in the following table:

Type	Range	Description
boolean	{True, false}	Boolean value
integer	$[-2^{31};2^{31-1}]$	Signed integer
real	$\pm 1.7 * 10^{308}$, 15 significant decimal places	Float
string	ISO-8859-1, null-terminated	String
time	[01.01.1970; 01.01.2037], Seconds exactly	Date and time
var		Untyped variable
object		Reference

The data types mentioned here are described in detail in section 0 "Primitive data types".

The types "var" and "object" represent special types. While "var" denotes an untyped variable (see section 3.3, "Dynamic type Binding"), the Variable "object" is a reference within the Homematic object model.

The Homematic object model is described in the Homematic script documentation Part 2: Object model.

3.1 Variable names

You can use the letters of the English alphabet, the numerals, and the underscore character "_" to label variables. However, a variablename cannot start with a digit.

3.2 Declaring and initializing variables

In principle, variables can be declared at any point. They are visible from the declaration to the end of the script. A declaration can be made either with or without value assignment. If the declaration does not initialize, a default value is assigned to the variable in question.

Example:

```
integer i;        ! declaration without initialization
integer j = 1;    ! Declaration with initialization
```

3.3 Dynamic typ b*inding*

Homematic script consistently uses dynamic type binding, i.e. variables are not bound to a fixed data type by declaration, but can change their data types during execution. In practice, the type of a variable can change with each assignment.

Example:

```
integer myVar = 1;                          ! MyVar is an integer
myVar = true;!                              MyVar is a Boolean value
myVar = "Hello world!";                     ! MyVar is a string
myVar = 1.0;                                ! MyVar is a floating-point number
myVar = @ 2001-01-01 00:00:00 @;            ! MyVar is a time
```

The rule that each assignment can change the type of a variable also applies to the declaration of a variable with simultaneous initialization.

Example:

```
integer i = "Hello world!"; ! I is a string
```

In this example, a variable i is first created as an integer. The value "Hello World!" is then assigned to her. This changes the data type and I is a string.

4 operators

The following table shows a list of operators that are available under Homematic script:

Operator	Allowed data types	Description	Example
=	All	Assignment	var i = 1;
+	Integer, Real, string, time	Addition Concatenating	i = i + 1;
-	Integer, Real, time	Subtraction	i = i-1;
*	Integer, Real	Multiplication	i = i * 10;
/	Integer, Real	Division	i = I/10;
==	Boolean, Integer, Real, string, equality t Ime		boolean b = (i = = 1);
<>, !=	Boolean, Integer, Real, string, inequality time		b = (i < > 1);
<	Integer, Real, string, time	Small	b = (i < 1);
<=	Integer, Real, string, time	Less than or equal to	b = (i < = 1);
>	Integer, Real, string, time	Greater	b = (i > 1);
>=	Integer, Real, string, time	Greater than or equal (	b = (i = 1);
&&	Boolean	Logical AND	b = (b & & True);
||	Boolean	Logical OR	b = (b | | true);
!	Boolean	Logical not	b =! b;
&	Integer	Bitwise AND	i = i & 1;
|	Integer	Bitwise OR	i = i | 1
#	String	Character string Concatenating	string s = "Hello" # "World";
.	Object	Method Access.	b = System. IsVar ("I");
%	Integer, Real	Modulo (integer i = i% 3; Remainder of a division)	

4.1 Ranking

In Homematic script there is no natural precedence of operators. Rules known from mathematics, such as "point calculation before line calculation", do not apply. Rather, expressions are simply calculated from right to left. Parentheses can be used to influence the order of processing.

Example:

```
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 Special features for complex expressions

For complex expressions, special effects in the context of Homematic script are summarized here.

Expressions are resolved from right to left. Because operators have no precedence, parentheses can also be used.

The data type of an operation is determined by the left operand. This also affects expressions that work with different types of data.

Example:

```
var x = 1/10.0;! x = 0; X is an integer
var y = 1.0/10;! y = 0.1; Y is a floating-point number
```

This behavior can cause problems when integers and floating-point numbers are mixed. In such a case, the replacements of the operands can be a solution.

Example:

```
var a = 3 * 2.5;! A = 6; A is an integer
var b = 2.5 * 3;! b = 7.5; B is a floating-point number
```

To force a result or subtotal of type "real", 0.0 can be added to the left side.

Example:

```
var c = 0.0 + 3;! c = 3.0; C is a floating-point number
```

5 Control structures

In order to influence the program flow, there are various control structures in Homematic script which are described below.

5.1 Conditions – *if*

```
if (< Boolean expression >) {<instruction block>}
[else {< statement block}] [elseIf (< Boolean expression >) {<instruction block>}]
```

You can use the "if" keyword to conditionally execute statement blocks. Behind the "If" is a Boolean expression in parentheses. If this expression returns true, the statement block is executed in the body of the IF clause. The curly braces around the statement block are obligatory and must not be omitted.

Optionally, the conditional statement block can also follow an "else" path. This is executed if the condition evaluates to False. Here too, the curly braces are obligatory.

Since firmware 2.29.22 exists the possibility to have an "elseif ()" Path followed by the conditional statement block, so that the following statements are not executed until the Boolean expression of the "ElseIf ()" Part has the value "true".

Example:

```
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 Loops – while

```
while (< Boolean expression >) {<instruction block>}
```

The while loop is an input-tested loop, which means that the Boolean expression is checked before each iteration. The loop body will continue to run until the value of the Boolean expression becomes false.

After the latest 500000 iterations, the script interpreter assumes that the relevant "while" loop does not terminate. In this case, the "while" loop will also exit.

Example:

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

5.3 iterators – *f*oreach

```
foreach (<Indexvariable>, <Liste>) {<instruction block>}
```

You can iterate through a list by using the Foreach loop. The list is a specially formatted character string. In each iteration, the index variable is assigned a value from the list. This will go through the complete list.

The individual elements of the index list are separated by tabs. The index variable itself must necessarily be a string.

Example:

```
string list        = "a\tb\tc";          ! List {"a", "b", "c"}
String output = "";                      !Output
string index;                            ! Index variable
foreach (index, list) {
  output = index # output;
}
! output = "cba";
```

As with while (), the use of foreach () makes it possible to cancel a loop after a maximum of 500000 iterations to ensure that an accidental infinite loop cannot cripple the complete execution of the script interpreter.

5.4 C*ancel* – Quit

```
quit
```

With the keyword "quit" scripts can be aborted prematurely.

Example:

```
Integer even = 3;
if (even & 1) {quit;}! "even" is not straight-> abort
```

6 Primitive data types

Homematic script knows a number of primitive data types, which are summarized in the following table:

Type	Description
var	Untyped data type
boolean	Boolean value
integer	Signed integer
real	Float
string	String
time	Time
object	Object of the Homematic object model
idarray	Identifier array

Primitive data types can be understood as objects, i.e. they have methods that can be used to access their content.

For the Var, object, and IDArray data types, the VarType () and type () methods are applicable as of firmware 2.29.18, as described in the following.

6.1 General Methods

Each primitive data type has a set of common methods. The methods can be used to determine the type of a variable at run time. It is also possible to convert values from one primitive data type to another.

6.1.1 Determine data type – VarType/Type

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

This method can be used to determine the data type of a variable. The result is an integer. The following table provides a mapping between the data types and the corresponding number constants:

VarType ()	Type ()– *Data* Type
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 Convert to character string – Tostring

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

Converts the variable to a string. When applied to a floating-point number, an implicit rounding takes place on the sixth decimal point.

For a floating-point number (type ' real '), as well as a string (type ' string ') it is possible to round the current value to the nearest number with precision p, specifying the optional parameter ' P ' before converting it to a string (equivalent Round ()).

Example:

```
var i = 1.23456;
var s = i.ToString(3); ! s = "1,235"; s is a string
var r = s.ToString(1) ;! r = "1.2"; r is a string
```

When using ToString () on a time variable of type ' time ', it is possible to specify an optional parameter's '. This corresponds to a direct call to the time. Format () function as in chapter 6.5.2.

Example:

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

6.1.3 Convert to Integer – ToInteger

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

Converts a variable to an integer. When applied to floating-point numbers, a round of precision occurs on the sixth decimal point before being converted to an integer. In addition, an optional parameter p can be specified so that this precision can be explicitly specified before a conversion takes place.

Example:

```
var s = "100";
var s = S. Tointeger ();! i = 100; I is an integer
```

6.1.4 Convert to T*ime* – ToTime

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

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

Converts a variable to a time.

Example:

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

6.2 Boolean values – *b*oolean

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

Boolean variables can take the values true and false.

6.3 Signed integers – *i*nteger

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

Signed integers are in the range of-2 ^{31} Up to $^{31-1}$.

6.4 Floating-point numbers – *r*eal

```
real r = 1.0;
real r = "-1.0 e-1". Tofloat ();! -0.1
```

Floating-point numbers have 15 significant decimal places. They are in the range of ± 1.7 * 10 $^{\pm 308}$.

6.5 Points of time – *t*ime

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

The time data type refers to times between January 01, 1970 and January 01, 2037. A time can be specified exactly in seconds.

6.5.1 Accessing individual items

Special methods can be used to access the individual fields of a point in time. The following table shows an overview of the available methods:

Method	Description
integer time.Year ();	Determines the year
integer time.Month ();	Determines the number of months
integer time.Day ();	Determines the month
integer time.Hour ();	Determines the hour

Integer time.Minute ();	Determines the minute
Integer time.Second ();	Determines the second
Integer time.Week ();	Determines the week number
Integer time.Weekday ();	Determines the number of the day of the week
Integer time.Yearday ();	Determines the number of the day in the year
Integer Time.IsLocalTime ();	Determines whether the time is specified in local time (1) or in the lifetime (0)
Integer time.IsDST ();	Determines whether the time is in local daylight saving time (1) or not (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 Formatted output – *f*ormat

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

Formats the time according to the data in "formatstring" and returns the result as a string.

The formatstring parameter is a string that can contain placeholders for various elements of the point in time. The following placeholders are available:

Placeholder	Description	Example (24th December 18:30:00)
%%	The percent sign	"%"
%a	Abbreviated weekday name	Wed
%A	Full weekday name	Wednesday
%b	Abbreviated month name	Dec
%B	Full month name	December
%c	Date and time	Wed Dec 24 18:30:00 2008
%B	Century - 1	20
%d	Month	24
%D	Date% m/% d% y	12/24/08
%F	Date% Y-% m-% d	2008-12-24
%h	Abbreviated month name	Dec
%H	Hour (24-hour clock)	18
%I	Hour (12-hour clock)	06
%j	Number of the day in the year	359
%m	Month number	12
%M	Minute	30
%n	New Line control characters	
%p	AM or PM	Pm
%r	Time (12-hour clock)	06:30:00 pm
%S	Second	00
%t	tab control characters	
%T	Time (24-hour clock)	18:30:00
%u	Weekday (Monday = 1)	3
%U	Week number (Week 1 from 1st Sunday in January)	51
%v	Week number (ISO 8601)	52
%w	Weekday (Sunday = 0)	3
%W	Week number (Week 1 from 1st Monday in January)	51
%x	Date	12/24/08

%X	Time	18:30:00
%y	Year (2 digits)	08
%Y	Year (4 digits)	2008
%z	Time Interval to GMT	+ 0100
%Z	Time Zone Name	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 Character strings – string

```
string s = "Hello world!";
```

A character string in Homematic script is encoded according to ISO-8859-1. It is a null-terminated string. Strings are optionally enclosed by single or double quotation marks. Special and control characters can be specified via escape sequences.

A character string can be formatted as a list. To do this, the individual list items are separated by tabs. This formatting is used by the Foreach Loop, among other things.

6.6.1 Escape sequences

The following table lists a series of escape sequences. These can be used within strings to encode special and control characters.

Escape sequence	Description
\\	backslash ()
\"	Double quotation mark (")
\'	Single quotation mark (')
\t	tab (ASCII 0x09)
\n	New Line (ASCII 0x0A)
\r	Carriage return (ASCII 0x0D)

6.6.2 Convert to floating-point number – Tofloat

```
Real string. Tofloat ();
```

Converts a string to a floating-point number.

Example:

```
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 Determine length – Length

```
Integer string. Length();
```

Determines the number of characters in a character string.

Example:

```
string s          = "HalloWelt!";
integer length = s.Length();  ! length = 11
```

6.6.4 Determine substring – Substr

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

Returns a substring. Where index is the first character and length is the length of the substring. The first character starts at index 0.

Example:

```
string s          = "Hello world!";
string world    = s.Substr (6, 4);! world = "World"
```

6.6.5 Determining a substring – Find/Contains/StartsWith/EndsWith

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

The Find () method determines the integer index of the first occurrence of the "key" substring in a string. If "key" is not found, "-1" is returned.

Since firmware 2.29.22 exist methods (contains, StartsWith, EndsWith) which, like find, perform a substring search, but as a result instead of an index return a Boolean value when (not) locating the substring. where contains () returns a value of TRUE/False if the specified substring is found or not found (equate to "find () > = 0" or "find () = =-1"). StartsWith () and EndsWith () return true/false respectively if the substring was either found at the beginning or end of the string or was not found.

Example:

```
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 Prepare for Iteration – Split

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

Creates a list. This replaces all occurrences of separator with tabs. The resulting string can be traversed by the Foreach loop.

Example:

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

6.6.7 Find list item – StrValueByIndex

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

The string is a list whose elements are separated from each other by separators. The StrValueByIndex method returns the list item specified by the index. The index count starts at 0.

Example:

```
string recipe    = "butter, eggs, flour, milk, sugar";
string ErsteZutat = recipe.StrValueByIndex (",", 0);! ErsteZutat = butter;
```

6.6.8 URI compliant String c*oding* – UriEncode/UriDecode (from F*W* 2.29.22)

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

These methods allow the conversion of a string to/from an RFC 3986 compliant way (% xx for non-URI-compliant characters) in order to be able to use it in a URL or to convert it from there to a normal string back.

Example:

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

6.6.9 UTF8/Latin String c*oding* – To*UTF*8 / ToLatin (from F*W* 2.29.22)

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

The ToUTF8 () method allows a string (which is standard in ISO-8859-1 encoding) to be converted to a UTF8 encoded string. Furthermore, a string already in UTF8 format can be converted back to a standard compliant ISO-8859 encoding.

Example:

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

6.6.10 Conversion u*pper and lower* case – *to*Upper/to*L*ower (from F*W* 2.29.22)

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

These methods allow you to convert a string to uppercase (upper) or lowercase (tolower).

Example:

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

6.6.11 Trimming Strings – Trim/LTrim/RTrim (from F*W* 2.29.22)

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

For a string, these methods allow the characters specified in the optional argument "chars" to be removed from the beginning (LTrim), from the End (RTrim), or from the beginning and end (trim) until a character is found that does not exist in the specified "chars" argument (So-called "trimming" of character strings).

If the optional chars argument is not specified, these methods remove unindented spaces (), form feeds (□), newline (or LF), carriage return (
or CR), tab character (□), or Vertical tab character (v) from the respective location of the Character string until another character was found.

Example:

```
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 Replace partial rows – Replace (from 2.29.22)

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

The method replace () allows you to search for a substring "src" in a string and replace it within the string with the spare part character string specified as "Dest".

Example:

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

7 Script environment

Typically , Homematic script is used within programs on the Homematic CCU . For this purpose special symbols are defined which represent the environment of the script. The symbols are replaced when they are executed.

7.1 Self-reference $this$

The $this $ symbol is available in any case. It is replaced by the ID of the program in which the script expires.

For the following example, a system variable named "i" of type "number" was created. Then a program was created which has no condition and the only action is the following script:

Example:

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

Then the program was saved and set to "active". As soon as you run the program through the operation, the program's ID is assigned to the system variable.

7.2 Source $s*rc*$

The src symbol indicates the data point that triggered the program in which the script is running. When a program responds to multiple events, it can be used to find out which of the events triggered execution.

Under certain circumstances, such as when a program is run manually, the src icon is not set. In these cases, it is not resolved.

In the following example, the system variable I (type: number) is set to the ID of the data point that triggered the program in question. If no source is available because the program has been executed manually, I will be set to-1.

Example:

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

Before the script is executed, all occurrences of "src" are replaced by the ID of the actual trigger . If the program is triggered by an external event, the variable "source" is set to the trigger in line 2 . If the program is run manually, there is no trigger and "src" is not replaced. Because there is no object named "src", "DOM. GetObject "is a value of" null ".

8 Mathematical functions

Starting with firmware 2.29.18, various mathematical functions and constants have been added which can be used within a script to perform complex mathematical calculations.

8.1 Functions

The following table contains all the mathematical functions that can be applied directly to a variable of the type "real" or "integer".

Function	Description	Example
Abs()	Calculate the absolute value of a number	r =-1.5; a = R. Abs(); ! a = 1.5
Mod(y)	Calculates remaining share of a division	r = 5.0; d = r.Mod (3);! d = 2.0
Min(y)	Returns the minimum value of two numbers	r = 5.0; m = r.Min (8.0); ! m = 5.0
Max(y)	Returns the maximum value of two numbers	r = 5.0; m = r.Max(8.0);! m = 8.0
Exp()	exponential function of the base E	r = 2.0; e = r.Exp ();! e = 7.389056
Exp2()	Base 2 exponential function	r = 2.0; e = r. Exp2();! e = 4.0
Exp10()	Base 10 exponential function	r = 2.0; r = r.Exp10 ();! e = 100.0
Expm1()	exponential function of the base e minus 1	r = 2.0; e = r.Expm1();! e = 6.389056
Log()	Natural logarithmic function	r = 2.0; l = r.Log();! l = 0.693147
Log2()	Base 2 logarithm function	r = 2.0; l = r.Log2();! l = 1.0
Log10()	Base 10 logarithm function	r = 2.0; l = r. Log10();! l = 0.301030
Log1p()	Natural logarithm plus 1	r = 2.0; Ll = r.log1p ();! l = 1.098612
Sqrt()	Square root function	r = 2.0; s = r.Sqrt();! s = 1.414214
Cbrt()	Cubic root function	R = 2.0; s = r.Cbrt();! s = 1.259921
Pow(y)	Power function	Rr = 2.0; p = r. Pow(2.0);! p = 4.0
Sin()	Sine function	r = 2.0; s = r. Sin();! s = 0.909297
Cos()	cosine function	r = 2.0; c = r.cos();! c =-0.416147

Tan()	Tangent function	r = 2.0; r = r.Tan();! t =-2.185040
Asin()	Arc sine function	r = 1.0; s = r.Asin();! s = 1.570796
Acos()	Arc cosine function	r = 0.0; c = R.Acos();! c = 1.570796
Atan()	Inverse tangent function	r = 2.0; t = r.Atan ();! t = 1.107149
Sinh()	Sinus hyperbolic function	r = 2.0; s = r.Sinh();! s = 3.626860
Cosh()	Cosine hyperbolic function	r = 2.0; c = r.Cosh();! c = 3.762196
Tanh()	Tangent hyperbolic function	r = 2.0; r = R.Tanh();! t = 0.964028
Asinh()	Arc sine hyperbolic function	r = 2.0; s = R.Asinh();! s = 1.443635
Acosh()	Arc cosine hyperbolic function	r = 2.0; c = r.Acosh();! c = 1.316958
Atanh()	Inverse tangent hyperbolic function	r = 0.5; t = r.Atanh();! t = 0.549306
Ceil()	Smallest integer value that is not less than As the current value is	r = 0.4521; c = r.Ceil();! c = 1.0
Floor()	Largest integer that is not greater than the current value	r = 0.4521; f = r.Floor();! f = 0.0
Trunc(p)	Rounds the current value with Accuracy p in direction 0	r = 0.4521; t = R. Trunc(1);! t = 0.4
Round(p)	Rounds the current value to the Nearest number with accuracy P	r = 0.4521; o = r.Round (1);! o = 0.5

8.2 Constants

The following table lists the mathematical constants available within a script that can be used in place of floating-point and integer variables.

Constant	**= Mathematical operation**	**Absolute value**
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 ()	2.30258509299404568402
M_PI	Pi	3.14159265358979323846
M_PI_2	PI/2	1.57079632679489661923

M_PI_4	PI/4	0.78539816339744830961
M_1_PI	1/Pi	0.31830988618379067153
M_2_PI	2/Pi	0.63661977236758134307
M_2_SQRTPI	2/PI. sqrt ()	1.12837916709551257390
M_SQRT2	"2". sqrt ()	1.41421356237309504880
M_SQRT1_2	1/"2". sqrt ()	0.70710678118654752401

Example:

```
var r = 4.2;! Circle radius in cm
var A = (M_PI * r. POW (2)). Round (2);! Circular Area A = Pi * r ^ 2 = 55.42 cm
```

9 Additional functions

Starting with firmware 2.29.18, various system-wide functions have been added which have proved useful for the programming of complex scripts. These should be listed below.

9.1 Random Numbers

The following functions exist for the calculation of random numbers:

Function	Description	Example
system.Random (min, max)	Calculation of a Integer random number with optional indication of an acceptable range	var dice = System. Random (1, 6); ! dice = 3
system.Srandom (int)	Allows to set the seed of the random generator to a defined value (only for debugging purposes)	var seed = system.random (); system. Srandom (seed);