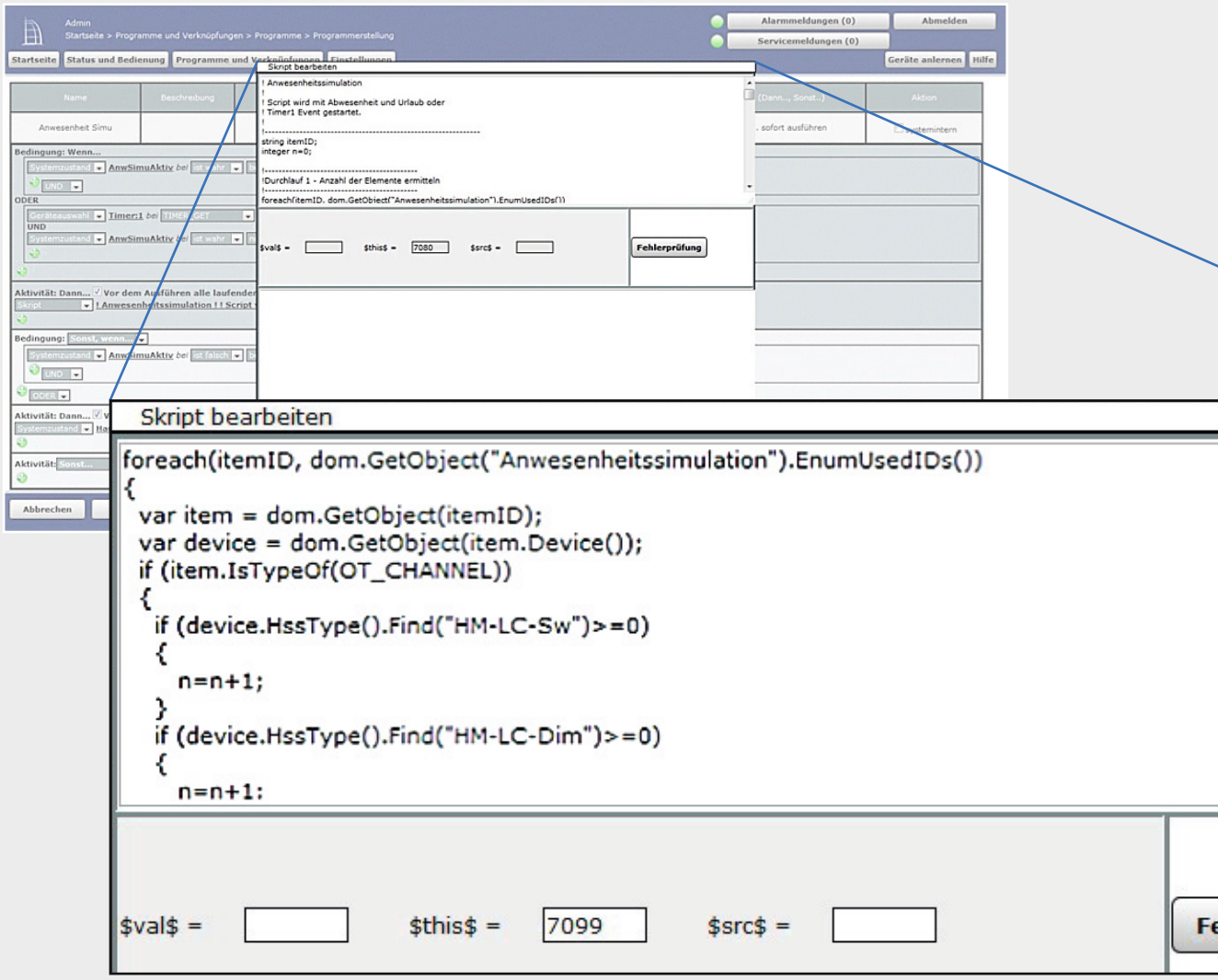
Homematic Scriptprogrammierung

Teil 1 – Einführung, Variablen, Befehle, Struktur, Programmierstil



Mit der CCU automatisch E-Mails verschicken, alle Lichter auf einfache Art und Weise ausschalten, Schaltvorgänge nach dem Inhalt der Wettervorhersage ausführen – dies und vieles mehr ist mit der Homematic Scriptprogrammierung möglich.

Diese Artikelserie beschreibt die Grundlagen der Scriptprogrammierung für Homematic und bringt Scriptbeispiele.

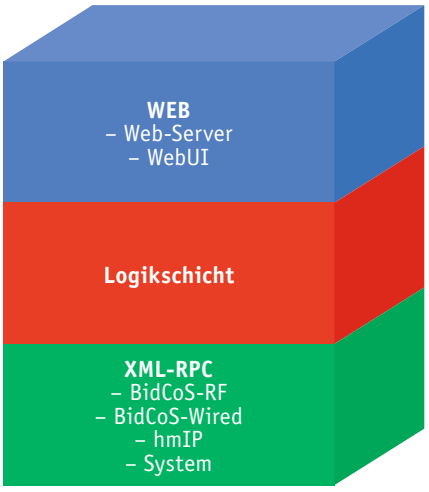
Allgemeines

Was ist Scriptprogrammierung?

Homematic Script ist eine Programmiersprache ähnlich C oder BASIC, die einen Zugriff auf die Homematic Logikschicht ermöglicht.

Warum Scriptprogrammierung?

Die Homematic Scriptprogrammierung greift direkt auf die Logikschicht der CCU zu. Damit sind die Möglichkeiten, die man als Anwender hat, um ein Vielfaches größer, als wenn man sich auf die reine Zentralenprogrammierung beschränkt.



Bestimmte Aufgaben wie z. B. arithmetische Operationen, erweiterte Zugriffe auf Variablen, das Versenden von E-Mails oder Push-Nachrichten, das Einholen von Informationen über Webseiten wie z. B. „Wunderground" oder auch Zugriffe auf eine API (eine Programmierschnittstelle, genauer: eine Schnittstelle zur Anwendungsprogrammierung – application programming interface) lassen sich erst mit der Scriptprogrammierung lösen.

Quellen

Wichtige Quellen für die Arbeit mit Scripten sind die Dokumentationen, die z. B. über den Downloadbereich von eQ-3 zu bekommen sind:
Teil 1: Sprachbeschreibung
Teil 2: Objektmodell
Teil 3: Beispiele
Teil 4: Datenpunkte
Teil 5: Beschreibung der XML-RPC-Schnittstelle
sowie Homematic XML-RPC-Schnittstelle – HmIP Addendum

Einsatz der Scriptprogrammierung

Wir sprechen im Wesentlichen von 3 Einsatzmöglichkeiten:

- Das wichtigste Einsatzgebiet der Scriptprogrammierung ist innerhalb eines Homematic Programms zu sehen. Hier kann z. B. ein Script als Aktion zu einem Ereignis definiert werden.

- Eine zweite Einsatzmöglichkeit findet sich im Rahmen der sogenannten Tcl-Scripte. Tcl-Scripte werden oft verwendet, die Scripte werden in diesem Fall aber über den Homematic Script-Interpreter verarbeitet. Einzelne Scripte werden über die Befehl *„rega_script"* ausgeführt. Die Rückgabe von Werten der verwendeten Variablen erfolgt über ein Tcl-Array. In diesem Array befindet sich auch die Standardausgabe, auf sie erfolgt der Zugriff über *„STDOUT"*.
- Als dritte Einsatzmöglichkeit wäre die Remote Homematic Schnittstelle zu nennen, wobei das Script per http-POST an die Homematic Zentrale gesendet wird.

Hinweise zur Firewall
Der Zugriff auf die Remote Homematic Script-Schnittstelle kann über die Firewall-Einstellungen der Homematic Zentrale innerhalb der Systemsteuerung begrenzt werden. Die Standardeinstellung lässt nur Zugriffe aus dem lokalen Netzwerk zu. Über die Firewall-Einstellungen ist es jedoch möglich, Remote Homematic Script global zuzulassen oder auch komplett zu blocken.

Grundsätzliches
Die Scriptprogrammierung erfolgt in einer Sprache, die C oder BASIC ähnlich ist. Der Sprachumfang ist allerdings nicht so groß wie z. B. in BASIC, reicht aber für die üblichen Erfordernisse aus.

Wie bei anderen Programmiersprachen auch gibt es Variablen, Befehle, Kontrollstrukturen usw. Der Sprachumfang ist auch in der „Dokumentation zur Scriptprogrammierung Teil 1 Sprachbeschreibung" ausführlich erklärt.

Schreibweise
Die Homematic Scriptsprache unterscheidet zwischen Groß- und Kleinschreibung – sie ist also case-sensitive. Die Bezeichnungen „heute" und „Heute" stehen für zwei verschiedene Symbole!

Kommentare
Geizen Sie in Ihren Scripten nicht mit Kommentaren. Sie machen die Scripte besser les- und verstehbar. Wenn Sie Ihre Scripte nach 2 Jahren durchlesen, haben selbst Sie als Autor Schwierigkeiten, das Script ohne entsprechende Kommentare schnell zu durchschauen!

Kommentare werden am Anfang durch ein Ausrufezeichen (!) eingeleitet, sie sind einzeilig und gehen bis zum Ende der aktuellen Zeile.

Struktur
Achten Sie beim Programmieren darauf, dass Ihre Programme strukturiert sind. Ein strukturiertes Programm ist auch nach längerer Zeit besser lesbar und vor allem besser verständlich.

Vergleichen Sie einmal die beiden folgenden Scripte:

Variante 1:

```
!Alles ausschalten, was zum Gewerk Garten gehoert
var i = dom.GetObject("Garten");
string itemID;
!Gewerk Garten durchsuchen und eingeschaltete Elemente ausschalten
foreach(itemID, i.EnumUsedIDs())
{
  var item = dom.GetObject(itemID);
  if (item.IsTypeOf(OT_CHANNEL))
  {
    var device = dom.GetObject(item.Device());
    if ((device.HssType().Find("HM-LC-Sw") >= 0) && (item.State()))
    {
      item.State(0);  !Ausschalten
    }
    if ((device.HssType().Find("HM-LC-Dim") >= 0) && (item.State()))
    {
      item.State(0.00);  !Wert auf 0 setzen
    }
  }
}
```

Variante 2:

```
string itemID;
foreach(itemID, dom.GetObject("Garten").EnumUsedIDs()){
var item = dom.GetObject(itemID);
if (item.IsTypeOf(OT_CHANNEL)) {
var device = dom.GetObject(item.Device();)
if ((device.HssType().Find("HM-LC-Sw") >= 0) && (item.State())){
item.State(0) };
if ((device.HssType().Find("HM-LC-Dim") >= 0) && (item.State())){item.State(0.00); }}}
```

Stellen Sie sich einfach einmal vor, ein Fremder soll sich in das Programm Variante 2 einlesen, oder Sie müssen im Programm nach längerer Zeit einen Fehler suchen …

Zu einem strukturierten Programm gehören:
- Sauberer logischer Aufbau
- Kommentare
- Verwendung von Variablen
- Einrückungen (IF, Schleifen …)
- Anweisungen bzw. Deklarationen müssen mit einen Semikolon ; abgeschlossen werden
- Werte in Zeichenketten sind in Hochkommata " oder Apostroph ' zu setzen

Besonderheiten der Homematic Scriptprogrammierung/Sprachelemente

- Die Anzahl der Variablen ist auf 200 begrenzt!
- Es gibt folgende Variablentypen:

Variablentyp (Datentyp)	Wertebereich	Beschreibung
boolean	true / false	logische Variable mit den Werten wahr und falsch
integer	-2^{31} … 2^{31-1}	Ganzzahl mit Vorzeichen
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 (Referenz innerhalb des Homematic Objektmodells)

Bei der Variablendeklaration gilt:
- Der Variablentyp „var" ist untypisiert – die Typisierung erfolgt bei der Wertezuweisung
- Die Wertezuweisung kann mit der Variablendeklaration oder auch später erfolgen:
  Beispiel:

```
boolean bErgebnis = true;          ! Wertezuweisung mit der Deklaration
boolean bErgebnis;                 ! Deklaration ohne Wertezuweisung
bErgebnis = true;                  ! reine Wertezuweisung
```

- Variablen können prinzipiell an jeder Stelle im Script deklariert werden. Sie sind nicht durch die Deklaration an einen festen Datentyp gebunden und können während der Ausführung ihren Datentyp wechseln
  Beispiel:

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

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. Bei der Programmierung wird oft der Variablentyp als Kennbuchstabe vor den Variablennamen gesetzt. Dies hat den Vorteil, dass man den Variablentyp in einem umfangreicheren Programm sofort erkennen kann, auch wenn man den Deklarationsteil nicht vor Augen hat:

Beispiele:

```
boolean b_Ergebnis;
integer i_n;
real r_Messwert;
string s_Text1;
```

Operatoren

Operator	Verwendbar mit den Datentypen	Kurzbeschreibung	Beispiel
= ==	alle boolean, integer, real, string, time	Zuweisung Abfrage auf Gleichheit	integer i=1; if (i==1) {sErgebnis="gleich"};

+
-
*
/
	integer, real, string, time
	integer, real, time
	integer, real
	integer, real
		Addition (Aneinanderreihung),
		Subtraktion,
		Multiplikation,
		Division
			i = i + 1;
			i = i – 1;
			i = i * 10;
			i = i / 10;

<>, !=
<
<=
>
>=
	boolean, integer, real, string, time
	integer, real, string, time
	integer, real, string, time
	integer, real, string, time
	integer, real, string, time
		ungleich
		kleiner als
		kleiner als oder gleich
		größer als
		größer als oder gleich
			if (i <> 3) {sErgebnis="ungleich"};
			if (i < 3) {sErgebnis="kleiner"};
			if (i <> 3) {sErgebnis="kleinergleich"};
			if (i <> 3) {sErgebnis="groesser"};
			if (i <> 3) {sErgebnis="groessergleich"};

&& || ! & | # .	boolean boolean boolean integer integer string object	logisches UND logisches ODER logisches NICHT bitweises UND bitweises ODER Zeichenketten aneinanderreihen Methodenzugriff	

Wichtig:
In der Homematic Scriptprogrammierung 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, müssen Klammern verwendet werden.

Beispiel:

```
I = 3 + 4 * 2  –  4 / 2;
```

ergibt 3 und nicht wie man vermuten würde 9! Da von rechts nach links gerechnet wird, erfolgt folgende Rechnung:

```
4 / 2 = 2
2 – 2 = 0
4 * 0 = 0
3 + 0 = 3
```

Deshalb:

```
i = 3 + (4 * 2) – (4 /2);
```

ergibt den Wert i=9

Das Verfahren, komplexe Ausdrücke von rechts nach links aufzulösen, führt bei Rechnungen mit gemischten Variablentypen (z. B. integer, real) natürlich auch dazu, dass das Ergebnis gleich dem Typ des letzten bei der Rechnung verwendeten Elements ist, also:

Die Zeile

```
var ergebnis = (3 * 5) – 2.5;
```

ergibt 13 und nicht 12.5, da die 3 und die 5 als Ganzzahlen (3 statt 3.0 ...) geschrieben wurden.

Um das Ergebnis durchgängig mit Gleitkommazahlen zu berechnen, muss geschrieben werden:

```
var ergebnis = (3.0 * 5.0) – 2.5;
```

Der Aufbau von Kontrollstrukturen

Kontrollstrukturen sind Werkzeuge, um den Programmablauf zu beeinflussen. Zu den bekanntesten Kontrollstrukturen gehören die IF-Abfrage sowie Schleifenstrukturen. Der schematische Aufbau ist hier immer gleich:

IF-Abfrage	WHILE-Schleife	ITERATION

if (<boolescher Ausdruck>)
{
Anweisung 1;
Anweisung 2;
Anweisung n;
}
else
{
Anweisung 1;
Anweisung 2;
Anweisung n;
}
	while (<boolescher Ausdruck>)
	{
	Anweisung 1;
	Anweisung 2;
	Anweisung n;
	}
		foreach (<Indexvariable>, <Liste>)
		{
		Anweisung 1;
		Anweisung 2;
		Anweisung n;
		}

Bei der IF-Abfrage kann der ELSE Zweig auch entfallen. Er behandelt den Fall, dass die Bedingung „if (<boolescher Ausdruck>)" falsch ist.	Die While-Schleife wird nicht verlassen, solange der boolesche Ausdruck wahr ist.	Bei der Iteration wird eine Liste Element für Element „durchgegangen". Bei der Liste handelt es sich um eine speziell formatierte Zeichenkette.

Boolesche Ausdrücke können z. B. sein:

```
if (iAnzahl == 5)
if (sName == "Helga")
if (rWert >= 3.75)
if (bErgebnis == true)
…
```

Ein Script-Abbruch kann über das Schlüsselwort „quit" erfolgen:

```
if (iAnzahl == 5)
{
   quit;
}
! weitere Programmzeilen
…
```

In diesem Fall wird das Script verlassen, wenn die Variable iAnzahl den Wert 5 hat, im anderen Fall wird das Script mit den Programmzeilen nach der geschweiften Klammer, die das IF-Konstrukt abschließt, fortgesetzt.

Methoden

In der objektorientierten Programmierung sind Methoden als Funktionen oder Prozeduren zu verstehen, die das Verhalten von Objekten beschreiben und implementieren. Über die Methoden des Objekts können Objekte interagieren. Methoden werden immer mit einem Punkt (.) an die Variable angeschlossen.
Die Methoden der einfachen Datenypen (integer, real, boolean, string) sind:

Methode	Funktion	Variablentyp der Methode	Beispiel
.VarType()	Gibt den Variablentyp zurück.	integer	iType = bErgebnis.VarType(); in der Variablen iType findet man nach dieser Programmzeile den Variablentyp der Variablen bErgebnis. Wenn die Variable bErgebnis vorher als boolean deklariert wurde, ist iType dann 1, denn es gilt: VarType() Datentyp 1 boolean 2 integer 3 real 4 time 5 string

.ToString()	Wandelt den Variablenwert (Integer oder Gleichkomma) in eine Zeichenkette um.	string	real rWert = 3.69;
			string sErgebnis = rWert.ToString();
			Nach dieser Programmzeile hat die Stringvariable sErgebnis den Wert „3.69", besteht also aus der Zeichenkette „3.69".

.ToInteger()	Wandelt den Variablenwert (String) in eine Ganzzahl um.	integer	string sWert = "243"; integer iWert = sWert.ToInteger(); Nach dieser Programmzeile hat die Integervariable iWert den Wert 243.

.ToTime()	Wandelt den Variablenwert (Integer) in eine Zeit bzw. ein Datum um.	time	var i = 1;
			time t = i.ToTime();
			Nach dieser Programmzeile hat die Variable t (Typ "time") den Wert @1970-01-01 01:00:01@
			Der Datentyp „time" bezeichnet Zeitpunkte zwischen dem 01. Januar 1970 und dem 01. Januar 2037 und kann sekundengenau angegeben werden.

.ToFloat()	Wandelt eine Zeichenkette in einen Gleitkommawert um.		var s = "3.65"; real rWert = s.ToFloat(); Nach dieser Programmzeile hat die Realvariable rWert den folgenden Wert: 3.65

Bei dem Datenformat time gibt es wiederum Methoden, mit deren Hilfe auf einzelne Elemente des Zeitformats zugegriffen werden kann:

Methode	Beschreibung	Beispiel
.Year()	Gibt die Jahreszahl zurück	time t = @2008-12-24 18:30:00@; integer iJahr = t.Year(); →iJahr = 2008
.Month()	Gibt die Zahl des Monats zurück (1...12)	time t = @2008-12-24 18:30:00@; integer iMonat = t.Month(); →iMonat = 12
.Day()	Gibt den Tag zurück	time t = @2008-12-24 18:30:00@; integer iTag = t.Day(); →iTag = 24
.Hour()	Gibt die Stundenzahl zurück	time t = @2008-12-24 18:30:00@; integer iStunde = t.Hour(); →iStunde = 18
.Minute()	Gibt die Minutenzahl zurück	time t = @2008-12-24 18:30:00@; integer iMinute = t.Minute(); →iMinute = 30
.Second()	Gibt die Sekunde zurück	time t = @2008-12-24 18:30:00@; integer iSekunde = t.Second(); →iSekunde = 0
.Week()	Gibt die Woche zurück	time t = @2008-12-24 18:30:00@; integer iWoche = t.Week(); →iWoche = 51
.Weekday()	Gibt den Wochentag zurück	time t = @2008-12-24 18:30:00@; integer iWochentag = t.Weekday(); →iWochentag = 4 Dies ist der Mittwoch, da mit Sonntag begonnen wird, zu zählen. Sonntag = 1
.Yearday()	Gibt den Tag im Jahr zurück	time t = @2008-12-24 18:30:00@; integer iJahrestag = t.Yearday(); →iJahrestag = 359
.isLocalTime()	Ermittelt, ob der Zeitpunkt in Lokalzeit (1) oder Weltzeit (0) angegeben ist	time t = @2008-12-24 18:30:00@; integer iLokalWelt = t.isLocalTime(); →i LokalWelt = 1
.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 iSommerzeit = t.isDST(); →iSommerzeit = 0

Formatierte Stringausgabe

Zeichenketten möchte man gerne formatiert ausgegeben haben. Die Formatierung geschieht über einen sogenannten Formatstring:

Format-string	Beschreibung	Beispiel
%%	Prozentzeichen	
%a	Abgekürzter Wochtagsname	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%a"); sFormatiert wird zu: „Wed“
%A	Vollständiger Wochentagsname	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%A"); sFormatiert wird zu: „Wednesday“
%b	Abgekürzter Monatsname	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%b"); sFormatiert wird zu: „Dec“
%B	Vollständiger Monatsname	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%B"); sFormatiert wird zu: „December“
%c	Tag, Monat, Datum und Uhrzeit	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%c"); sFormatiert wird zu: „Wed Dec 18:30:00 2008“
%C	Jahrhundert -1	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%C"); sFormatiert wird zu: „20“
%d	Monatstag	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%d"); sFormatiert wird zu: „24“
%D	Datum %m/%d/%y	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%D"); sFormatiert wird zu: „12/24/08“
%F	Datum %Y-%m-%d	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%F"); sFormatiert wird zu: „2008-12-24“
%h	Abgekürzter Monatsname	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%h"); sFormatiert wird zu: „Dec“
%H	Stunde (24-Stunden-Uhr)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%H"); sFormatiert wird zu: „18“
%I	Stunde (12-Stunden-Uhr)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%I"); sFormatiert wird zu: „06“
%j	Nummer des Tages im Jahr	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%j"); sFormatiert wird zu: „359“
%m	Monatsnummer	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%m"); sFormatiert wird zu: „12“
%M	Minute	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%M"); sFormatiert wird zu: „30“

\n	Steuerzeichen für neue Zeile/Umbruch	string s = "jetzt kommt eine \n neue Zeile"; Ausgabe: jetzt kommt eine neue Zeile
%p	AM oder PM	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%p"); sFormatiert wird zu: „PM“
%r	Uhrzeit (12-Stunden-Uhr)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%r"); sFormatiert wird zu: „06:30:00 PM“
%S	Sekunde	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%S"); sFormatiert wird zu: „00“
%t	Tabulator Steuerzeichen	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format('%t%p'); sFormatiert wird zu " PM"
%T	Uhrzeit (24-Stunden-Uhr)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%T"); sFormatiert wird zu: „18:30:00“
%u	Wochentag (Montag = 1)!!!	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%u"); sFormatiert wird zu: „3“
%U	Wochennummer (Woche 1 ab dem 1. Sonntag im Januar)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%U"); sFormatiert wird zu: „51“
%V	Wochennummer (ISO 8601)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%V"); sFormatiert wird zu: „52“
%w	Wochentag (Sonntag = 0)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%w"); sFormatiert wird zu: „3“
%W	Wochennummer (Woche 1 ab dem 1. Montag im Januar)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%W"); sFormatiert wird zu: „51“
%x	Datum	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%x"); sFormatiert wird zu: „12/24/08“
%X	Uhrzeit	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%X"); sFormatiert wird zu: „18:30:00 PM“
%y	Jahreszahl (2 Ziffern)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%y"); sFormatiert wird zu: „08“
%Y	Jahreszahl (4 Ziffern)	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%Y"); sFormatiert wird zu: „2008“
%z	Zeitabstand zu GMT	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%z"); sFormatiert wird zu: „+0100“
%Z	Name der Zeitzone	time t = @2008-12-24 18:30:00@; string sFormatiert = t.Format("%Z"); sFormatiert wird zu: „CET“

Innerhalb von Zeichenketten können Steuerzeichen oder Sonderzeichen benötigt werden. Diese Zeichen können mithilfe sogenannter ESCAPE-Sequenzen generiert werden:

ESCAPE-Sequenz	Beschreibung
\\	Backslash (\)
\"	Doppeltes Anführungszeichen (")
\'	Einfaches Anführungszeichen (')
\t	Tabulator (ASCII 0x09)
\n	Neue Zeile (ASCII 0x0A) (Line Feed)
\r	Wagenrücklauf (ASCI 0x0D) (Carriage Return)

Stringfunktionen

Speziell für die Stringverarbeitung gibt es zahlreiche sogenannte „Stringfunktionen“, Methoden, mit deren Hilfe Zeichenketten in einfacher Weise modifiziert, formatiert und verbunden werden können:

Methode	**Variablentyp der Methode**	**Beschreibung**	**Beispiel**
.Length()	integer	Gibt die Länge einer Zeichenkette zurück	string s = "Wie lang bin ich?"; integer nAnzahl = s.Length(); nAnzahl hat den Wert 17
.Substr(integer index, integer length)	string	Gibt einen Teil einer Zeichenkette zurück	string s = "Dies ist ein langer Text"; string sTeil= s.Substr(5,3); sTeil hat den Wert „ist“ Achtung: Das erste Zeichen beginnt bei 0! (index)
.Find(string key)	integer	Index einer Teilzeichenfolge ermitteln	string s = "Dies ist ein langer Text"; integer iBeginn = s.Find("ist"); iBeginn hat den Wert 5 Achtung: Das erste Zeichen beginnt bei 0! (index) Wird der Teilstring nicht gefunden, bekommt iBeginn den Wert -1 (entspricht false!). Die Suchfunktion ist Case-Sensitive
.Split(string separator)	string	Erstellt eine Liste. Dabei werden alle Vorkommen von „separator“ durch Tabulatoren ersetzt. Die so entstandene Zeichenfolge kann von der „foreach“-Schleife durchlaufen werden.	string s_zahlen = "1,2,3"; string s_split = s_zahlen.Split(","); Der String s_split sieht nun folgendermaßen aus: s_Split = "1\t2\t3"
.StrValueByIndex(string separator, integer index)	string	Listenelement ermitteln	string sRez="Butter,Eier,Mehl,Milch,Zucker"; string sErsteZutat = Rezept.StrValueByIndex(",", 0); sErsteZutat wird damit zu "Butter". Das erste Listenelement hat den Index 0!

Ausblick: Im zweiten Teil dieser Reihe werden wir uns mit Objekten und Datenpunkten beschäftigen und die ersten Scripte schreiben. ELV