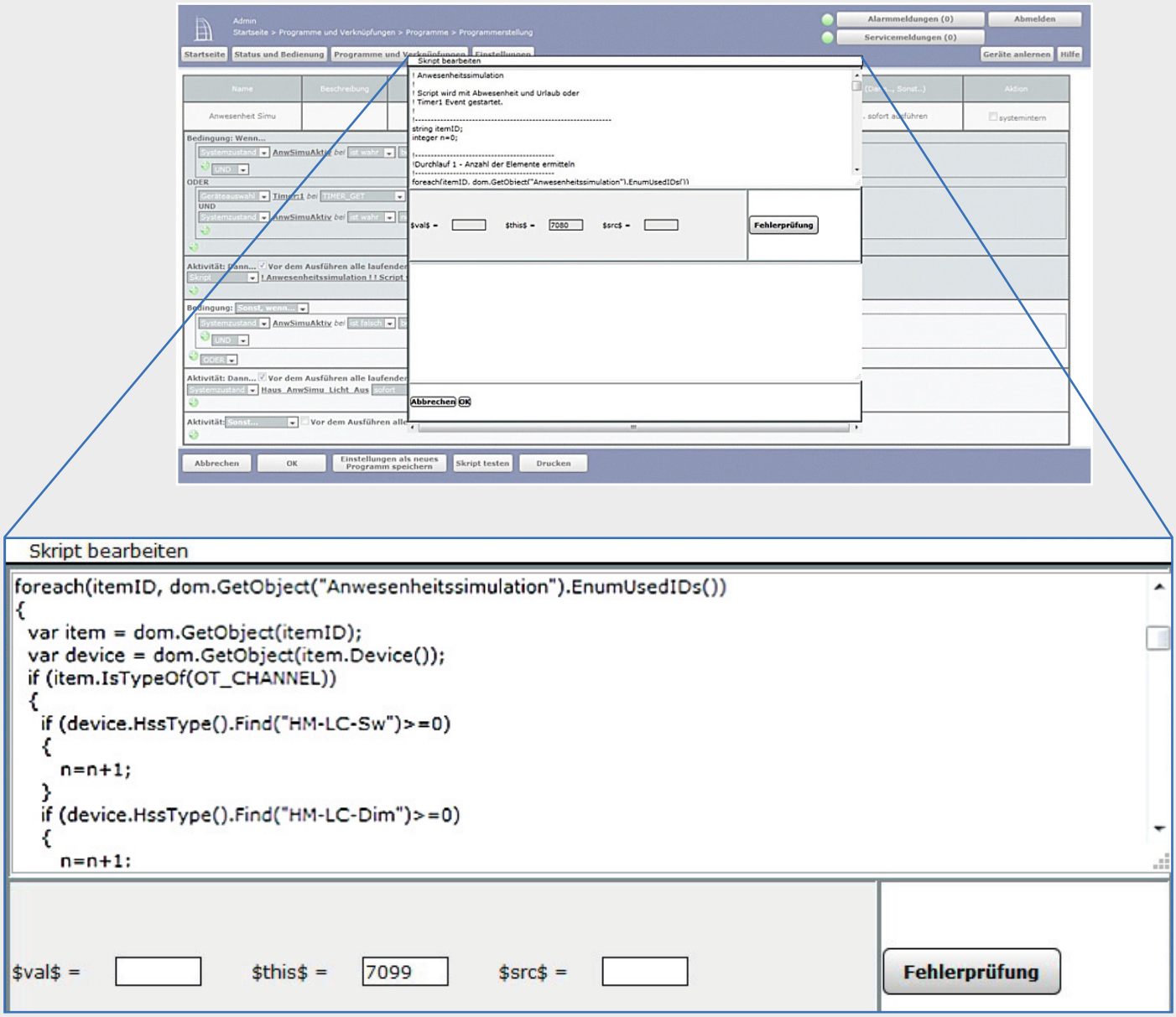
Homematic Scriptprogrammierung

Teil 4 – Das CCU-Add-on CUxD und Scripte



Im vierten Teil der Artikelserie über die Scriptprogrammierung beschäftigen wir uns mit dem CCU-Add-on CUxD und schreiben weitere Scripte.

Das CCU-Add-on CUxD

Für die Homematic CCU gibt es mittlerweile eine ganze Reihe von sogenannten Add-ons, also von Softwarepaketen, die auf der Homematic Zentrale (CCU) installiert werden können und neue Funktionalitäten zur Verfügung stellen, die nicht Teil der offiziellen Firmware sind.

Der CUx-Daemon (CUxD) ist eine universelle Schnittstelle zwischen der Homematic Zentrale (CCU) und entsprechend kompatiblen Komponenten von EnOcean, FS20, ELV-Wetterstationen und vielen mehr. Durch Einbindung dieser (eigentlich inkompatiblen) Produkte lässt sich der Umfang eines Homematic Systems also über die Grenzen hinweg erweitern.

Weiterhin bietet CUxD Funktionen, die nicht im Umfang der Homematic Scriptprogrammierung vorhanden sind.

Die Einrichtung und Verwendung des Add-ons ist in der dazugehörigen Dokumentation sehr gut beschrieben. Im Folgenden ein paar Beispiele mit Funktionen aus dem CUxD-Paket :

SYSTEM_EXEC

Die wohl bekannteste Anwendung von CUxD-Möglichkeiten ist der Ersatz des undokumentierten und nicht unproblematischen SYSTEM_EXEC-Script-Befehles durch den korrespondierenden CUxD-Befehl.

Hier ein Beispiel für den Versand einer E-Mail mithilfe des E-Mail-Add-ons und des CUxD-System-Exec-Befehls:

Nicht verwenden:

```
1  string stderr;
2  string stdout;
3  system.Exec("/etc/config/addons/email/email 03 '"#MailText#"'",&stdout,&stderr);
```

Stattdessen:

```
1  dom.GetObject("CUxD.CUX2800001:10.CMD_EXEC").State("/etc/config/addons/email/
   email 03 '"#MailText#"'");
```

CUxD.CUX2800001:10 muss natürlich entsprechend den eigenen Gegebenheiten abgeändert werden!

Zufallszahlen und Timer

Im Zusammenhang mit einem weiteren Script, das wir vorstellen möchten, wird die Generierung von Zufallszahlen mithilfe des CUxD-Add-ons gezeigt.

Bei dem Script handelt es sich um eine Anwesenheitssimulation, die zufällig gewählte Lampen im Haus für zufällig gewählte Zeiten ein- und dann wieder ausschaltet.

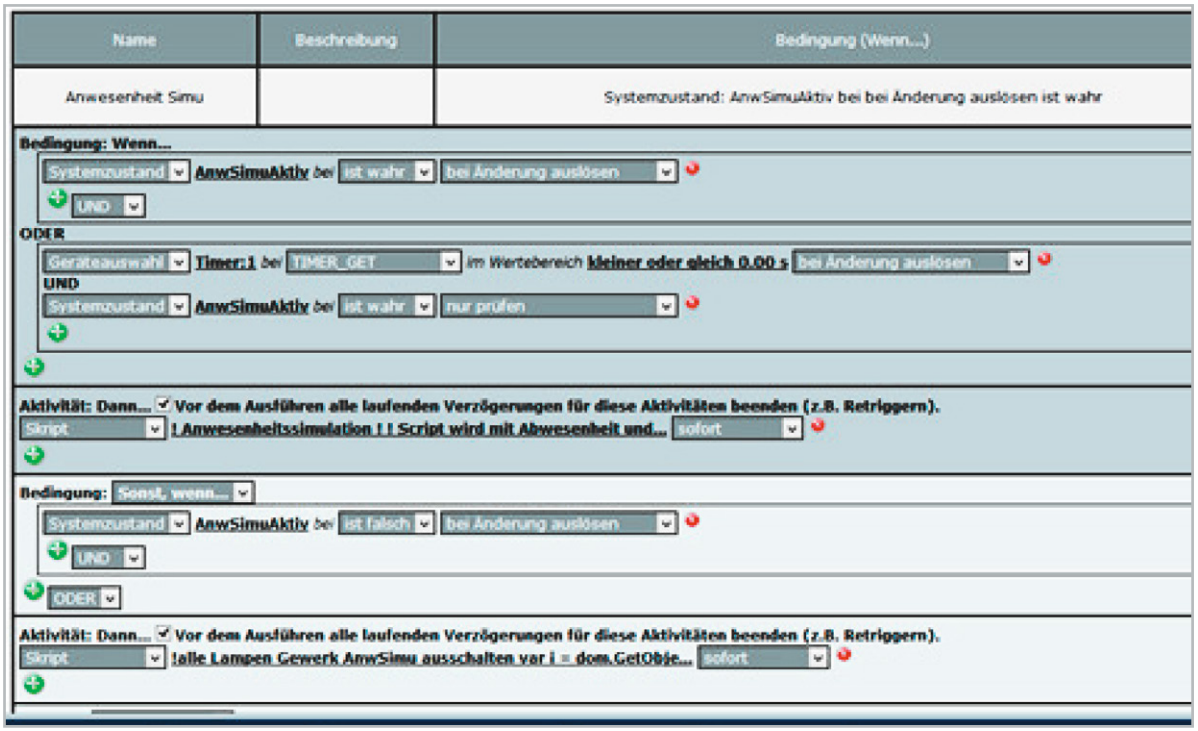
Wir verwenden dazu folgende CUxD-Befehlsstruktur:

```
dom.GetObject("CUxD.CUX2801001:2.RAND").State(xx);
integer RNDTime=dom.GetObject("CUxD.CUX2801001:2.RAND").State().ToInteger();
```

CUxD.CUX2801001:2 ist der Name und der Kanal des CUxD-Elementes, die Einrichtung ist in der CUxD-Dokumentation sehr umfangreich beschrieben.

Die erste Programmzeile wählt den Bereich an, aus dem die Zufallszahl stammt, der Bereich ist:
0 <= Zufallszahl <= xx
D. h., bei xx = 90 wird mit der nächsten Zeile in die im Script deklarierte Variable RNDTime eine zufällige Zahl aus dem Bereich 0 ... 90 gespeichert.

Voraussetzung für den Betrieb des Scriptes ist nun lediglich noch das Anlegen eines Gewerkes „Anwesenheitssimulation“, dem alle Lampen zugeordnet werden, die innerhalb der Simulation ein- und ausgeschaltet werden sollen. Das folgende Zentralenprogramm startet dann das Script:



Gestartet wird die Anwesenheitssimulation in Form eines ersten Scriptes durch Setzen der Systemvariablen (CCU-Variablen) *AnwSimuAktiv* auf *true*.

Zum Ausschalten der Anwesenheitssimulation wird ein zweites Script, das eventuell noch leuchtende Lampen ausschaltet, mit dem Zustand *false* der Systemvariablen *AnwSimuAktiv* gestartet.

Anmerkung: Vor allem das erste Script lässt sich natürlich vereinfachen, mit der gezeigten Struktur lassen sich aber hervorragend (weitere) Versuche machen.

Das erste Script:

```
! Anwesenheitssimulation
!
! Script wird z.B. mit Abwesenheit oder Urlaub gestartet.
! Das Script muss zur ordnungsgemaessen Funktion auch mit dem Timer1 Event gestartet
! werden.
!-----------------------------------------------------------------
string itemID;
integer n=0;
!-----------------------------------------------
!Durchlauf 1 - Anzahl der dem Gewerk Anwesenheitssimulation zugeh. Elemente ermitteln
!-----------------------------------------------
foreach(itemID, dom.GetObject("Anwesenheitssimulation").EnumUsedIDs())
{
  var item = dom.GetObject(itemID);
  var device = dom.GetObject(item.Device());
  if (item.IsTypeOf(OT_CHANNEL))
  {
    if (device.HssType().Find("HM-LC-Sw")>=0)
    {
       n=n+1;
    }
    if (device.HssType().Find("HM-LC-Dim")>=0)
    {
       n=n+1;
    }
  }
}
! kein Element fuer Anwesenheitssimulation gefunden
if (n==0) {
   quit;
}
! Eine Zufallszahl (RND_Nr) im Bereich von 0 bis zur gefundenen Anzahl von Lampen im
  Gewerk (n) ermitteln
dom.GetObject("CUxD.CUX2801001:1.RAND").State(n);
integer RND_Nr = dom.GetObject("CUxD.CUX2801001:1.RAND").State().ToInteger();
!fuer Durchlauf 2 n=-1 setzen
n=-1;
!-----------------------------------------------
!Durchlauf 2 - Zufalls-Element schalten
!-----------------------------------------------
foreach(itemID, dom.GetObject("Anwesenheitssimulation").EnumUsedIDs())
{
  var item = dom.GetObject(itemID);
  if (item.IsTypeOf(OT_CHANNEL))
  {
     var device = dom.GetObject(item.Device());
     !---------------------------------------
     !Einschalten bzw. auf 100 Prozent dimmen
     !---------------------------------------
     if (device.HssType().Find("HM-LC-Sw")>=0)
     {
       n=n+1;
       if (n == RND_Nr) {
         !-----------------------------------------------------------------
         !Einschaltzeit (RNDTime) innerhalb von 0...90 Minuten Zufallszahl
         !-----------------------------------------------------------------
         dom.GetObject("CUxD.CUX2801001:2.RAND").State(90);
         integer RNDTime=dom.GetObject("CUxD.CUX2801001:2.RAND").State().ToInteger();
         if (RNDTime==0) {RNDTime=1;}
         !---------------------
         !Einschaltzeit setzen
         !---------------------
         item.DPByHssDP("ON_TIME").State(60*RNDTime);


         !-------------
         !Einschalten
         !-------------
         item.State(1);
       }
     }

     if (device.HssType().Find("HM-LC-Dim")>=0)
     {
       n=n+1;
       if (n == RND_Nr) {
```

```
76          !-------------------------------------------------------
77          !Einschaltzeit innerhalb von 0...90 Minuten Zufallszahl
78          !-------------------------------------------------------
79          dom.GetObject("CUxD.CUX2801001:2.RAND").State(90);
80          integer RNDTime=dom.GetObject("CUxD.CUX2801001:2.RAND").State().ToInteger();
81          if (RNDTime==0) {RNDTime=1;}
82          !---------------------
83          !Einschaltzeit setzen
84          !---------------------
85          item.DPByHssDP("ON_TIME").State(60*RNDTime);
86          !-------------
87          !Einschalten
88          !-------------
89          item.State(1.00);
90        }
91      }
92    }
93  }
94  !-----------------------
95  ! CUxD Timer setzen
96  !-----------------------
97  dom.GetObject("CUxD.CUX2800001:1.TIMER_SET").State(60*RNDTime);
```

CUxD.CUX2800001:1 bzw. CUxD.CUX2801001:2 sind natürlich auf die eigenen Gegebenheiten anzupassen.

Funktion des Scriptes:
Zunächst werden in einer Schleife (Zeilen 12 bis 27) alle Aktoren gezählt, die dem Gewerk „Anwesenheitssimulation" angehören. Diese Funktionalität lässt sich natürlich auch mit der Funktion .Count() erreichen, dann dürfen aber wirklich nur Relais und Dimmer zum Gewerk „Anwesenheitssimulation" gehören.

Nun wird aus diesen Aktoren derjenige bestimmt, der eingeschaltet werden soll. Dies geschieht durch Wahl einer Zufallszahl zwischen Null und der Anzahl, die vorher ermittelt wurde (Zeilen 33 bis 35).

Angenommen, es werden 6 Geräte gefunden, die zum Gewerk „Anwesenheitssimulation" gehören, dann hat n in der Zeile 33 den Wert 6. Somit gilt:

Mit der Zeile

```
dom.GetObject("CUxD.CUX2801001:1.RAND").State(6);
```

wird bestimmt, dass die Zufallszahl im Bereich von 0 bis 6 liegen soll.

Mit der Zeile

```
integer RND_Nr = dom.GetObject("CUxD.CUX2801001:1.RAND").State().ToInteger();
```

steht die Zufallszahl dann in der Variablen *RND_Nr*.

In einem zweiten Durchlauf (Zeile 40 bis 93) wird dann der per Zufallszahl ermittelte Aktor (Relais oder Dimmer) eingeschaltet. Dies geschieht für eine Zeit im Bereich von 0 bis 90 Minuten, die ebenfalls per Zufallszahl ermittelt wird (Zeile 56 bis 58 bzw. 79 bis 81).

Sollte als Zufallszahl die 0 ermittelt werden, wird stattdessen der Wert 1 genommen.
Abschließend wird dann ein CUxD-Timer auf die zufällig ermittelte Einschaltzeit gesetzt, damit nach deren Ablauf das Script wieder gestartet wird, um die nächste (zufällig ermittelte) Lampe für eine (ebenfalls zufällig ermittelte) Zeit einzuschalten.

Das zweite Script mit dem Ausschalten der eventuell noch leuchtenden Lampen:

```
1   !alle Lampen Gewerk Anwesenheitssimulation ausschalten
2   var i = dom.GetObject("Anwesenheitssimulation");
3   string itemID;
4   foreach(itemID, i.EnumUsedIDs())
5   {
6     var item = dom.GetObject(itemID);
7     if (item.IsTypeOf(OT_CHANNEL))
8     {
9       var device = dom.GetObject(item.Device());
10      if ((device.HssType().Find("HM-LC-Sw") >= 0))
11      {
12        item.DPByHssDP("ON_TIME").State(0);
13        item.State(0);
14      }
15      if ((device.HssType().Find("HM-LC-Dim") >= 0))
16      {
17        item.DPByHssDP("ON_TIME").State(0);
18        item.State(0.00);
19      }
20    }
21  }
```

Das CUxD-Add-on bietet über die gezeigten Funktionen hinaus viele weitere mehr. Probieren Sie es aus!

Beispiel für die Ausgabe von http-Befehlen

Mit dem zunehmenden Wachstum des „Internet of Things" steigt die Anzahl der Geräte, die über http-Befehle gesteuert werden. Für das Absetzen eines http-Befehls wird ebenfalls die oben schon angesprochenen CUxD-Funktion SYSTEM_EXEC verwendet. Hier ein Beispiel für die Ansteuerung eines Mediola Interfaces (Schnittstelle zu Infrarot-Geräten) mit einem Infrarot-Steuerbefehl:

Laut Dokumentation des Interfaces setzt sich der Sendecode wie folgt zusammen:

```
http://aaa.bbb.ccc.ddd/command?XC_PASS=passwort&code=xyz&XC_FNC=Send2
```

Dabei sind:

aaa.bbb.ccc.ddd = IP-Adresse des Interfaces
passwort = Passwort des Interfaces
xyz = Infrarot Code

Der Infrarot-Code wird zunächst in eine Variable geschrieben (in diesem Fall z. B. der Code für die Taste ON/OFF einer Fernseh-Fernbedienung):

```
!----------------------------
!Code ON OFF
!----------------------------
string CODE = "190826000003002204046b046b008a01a5008a008a008a2afe000101010202020202010101020202020
201020201010202010201010202010101020 3";
```

ebenso die IP-Adresse des Adapters (Beispiel):

```
string IP_ADR = "192.168.2.155"
```

sowie das Passwort (Beispiel):

```
string PW = "Adapterpasswort"
```

Damit sieht der Sendebefehl beispielsweise folgendermaßen aus:

```
!----------------------------
!Sendecode
!----------------------------
string url="http://" # IP_ADR # "/command?XC_PASS=" # PW # "&code=" # CODE # "&XC_FNC=Send2";
```

Die Einzelstrings werden einfach zum Sendecode gekettet.

Das Absetzen des Sendebefehls erfolgt nun durch den CUxD-Befehl:

```
!---------------
!Ausfuehren
!---------------
dom.GetObject("CUxD.CUX2801001:1.CMD_EXEC").State("wget -q -O - '" # url # "'");
```

Script: Verknüpfte Informationen

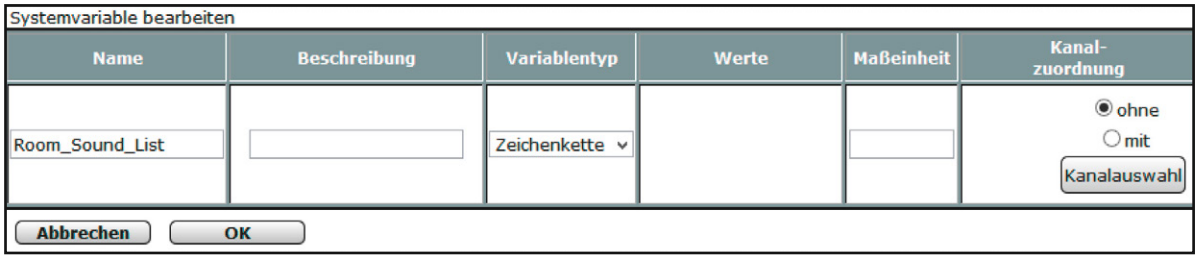
In CCU-Systemvariablen (z. B. Wertelisten oder Textvariablen) lassen sich auch sehr gut Listen mit Informationen unterbringen und verarbeiten. Dies ist vor allem dann sinnvoll, wenn Sie dadurch vermeiden, innerhalb von Scripten Informationen (Daten) zu programmieren, die sich irgendwann einmal ändern können. Auch können Programmteile, die Sie in anderen Scripten ebenfalls verwenden möchten, ohne Änderungen übernommen werden.

Wenn Sie z. B. den Funk-MP3-Gong zur Ausgabe von Sprachinformationen verwenden wollen, ergibt beispielsweise die Zuordnung von Raumnamen zu Textnummern der Soundfiles solch eine Liste:

Raum	Nummer des Soundfiles
Wohnzimmer	100
Esszimmer	101
Küche	102
Badezimmer	103
Schlafzimmer	104
Kinderzimmer	105
Flur	106
WC	107
Garten	108
Terrasse	109
Garage	110

Diese Tabelle lässt sich in einer Systemvariablen unterbringen und in einem Script verwenden.

Erstellung der Systemvariablen (Typ Zeichenkette):



In einem (Zentralen-)Programm wird dann die Variable *Room_Sound_List* einmalig mit dem folgenden String beschrieben:

```
"Wohnzimmer*100,Esszimmer*101,Küche*102,Badezimmer*103,Schlafzimmer*104,
Kinderzimmer*105,Flur*106,WC*107,Garten*108,Terrasse*109,Garage*110"
```

Es handelt sich also um eine *Liste* (Kommatrennung), deren Listenelemente jeweils aus dem Namen des Raumes sowie der zugehörigen Textnummer (immer dreistellig, also auch 001, 002 ...), getrennt durch einen Separator (in diesem Fall *) bestehen.

Im folgenden Script ermitteln die Zeilen 9 bis 15 nun aus dem Raumnamen, der in der Variablen *sRaum* steht, und der Zentralenvariablen *Room_Sound_List* die Nummer des zur Küche gehörenden Textes (MP3-File):

```
1  string daten;
2  string sRaum = "Küche";
3  string sTextnummer;
4  string sCommand;
5  integer word_position;
6  !-------------------------------
7  ! Textnummer fuer Raum
8  !-------------------------------
9  daten = dom.GetObject("Room_Sound_List").State();
10 word_position = daten.Find(sRaum);
11 if (word_position>=-1)
12 {
13 daten= daten.Substr(word_position+1,daten.Length()- word_position);
14 word_position = daten.Find("*");
15 sTextnummer = daten.Substr(word_position+1,3);
16 sCommand = sCommand # "," # sTextnummer;
```

Die Textnummer wird als String ermittelt, da sie ja bei der Ansteuerung des MP3-Funk-Gongs auch im Stringformat an diesen übermittelt werden muss.

In der Zeile 16 wird die Nummer in den Kommandostring für den Funk-Gong eingesetzt.

Hier ein komplettes Beispiel für eine akustische Meldung über den Batteriezustand der batteriebetriebenen Geräte in einer Hausinstallation. Neben der Zentralenvariablen *Room_Sound_List* gibt es eine Variable *Ger_Sound_List*, in der analog zu obigem Beispiel die Zuordnungen zwischen Gerätenamen und Soundfilenummern gespeichert sind.

Im folgenden Beispiel wurden die Gerätenamen wie folgt gewählt:
*Raum*Gerätebezeichnung*laufende Nummer dieses Gerätetyps in diesem Raum*
Also zu Beispiel:
*Wohnzimer*Türschalter*01*
Die laufende Nummer einfach deshalb, weil es mehrere gleiche Geräte in einem Raum geben kann.

Das Script überprüft den Batteriezustand jedes batteriebetriebenen Gerätes und gibt entweder eine O.K.-Meldung oder die Namen der Geräte aus, deren Batterien zur Neige gehen (max. 4 Stück gleichzeitig).

```
1  ! Batteriemeldungen Sprachausgabe
2  string itemID;
3  string text = "";
4  integer counter = 0;
5  string command;
6  string text_nr;
7  string daten;
8  string d_Raum;
9  string d_Geraet;
```

```
string d_Nummer;
integer word_position;
!****************************************
! Variablen (Textnummern) hier bitte anpassen
!****************************************
!Textnummer von "Die Batterien der folgenden Geräte sind bald leer"
string text_batt_leer = "126";
!Textnummer von "Alle Batterien sind voll"
string text_batt_ok = "213";
!maximale Anzahl von Geräten, die angesagt werden
integer max_anz_mldg = 4;
!Name des Gewerkes BATTERIE
string gew_batt = "Batteriebetrieb";
var myAssembly = dom.GetObject(gew_batt);
command= "1,1,108000" # "," # text_batt_leer;
foreach(itemID,myAssembly.EnumUsedIDs())
{
        var item = dom.GetObject(itemID);
        var device = dom.GetObject(item.Device());
        var interface = dom.GetObject(item.Interface());
        string interface_name = interface.Name();
        string device_address = device.Address();
        device_address = device_address.StrValueByIndex(":",0);
        boolean condition=false;
        if (device.HssType().Find("HM-CC-RT-DN")>=0) {
           string channel_name=interface_name # "." # device_address #
":4.FAULT_REPORTING";
           var channel = dom.GetObject(channel_name);
           if (channel.Value() ==6)
           {
                condition = true;
           }
           else
           {
                condition = false;
           }
        }
        else
        {
            string channel_name = interface_name # "." # device_address # ":0.LOWBAT";
            var channel = dom.GetObject(channel_name);
            if ( (channel.State() == true) && (channel.Value() == true) )
            {
                condition = true;
            }
            else
            {
                condition = false;
            }
        }
        if (counter < (max_anz_mldg-1))
        {
            if (condition == true)
            {
              !-------------------------------------------------------
              ! Raum, Geraet und Nummer ausdecodieren
              !-------------------------------------------------------
              word_position = daten.Find("*");
              if (word_position>0)
              {
                d_Raum = daten.Substr(0, word_position);
                daten = daten.Substr(word_position+1,daten.Length()- word_position-1);
                word_position = daten.Find("*");
                if (word_position>-1)
                {
                   d_Geraet = daten.Substr(0,word_position);
                   daten=daten.Substr(word_position+1,daten.Length()-word_position-1);
                   d_Nummer = daten;
                }
              }
              !------------------------
              !  UND
              !------------------------
              if (counter>0)
              {
                 command = command # "," # "020";
              }
              !-------------------------------
              ! Textnummer fuer Raum
```

```
88                  !--------------------------------
89                  daten = dom.GetObject("Room_Sound_List").State();
90                  word_position = daten.Find(d_Raum);
91                  if (word_position>=-1)
92                  {
93                     daten=daten.Substr(word_position+1,daten.Length()- word_position);
94                     word_position = daten.Find("*");
95                     text_nr = daten.Substr(word_position+1,3);
96                     command = command # "," # text_nr;
97                  }
98                   !--------------------------------
99                  ! Textnummer fuer Geraet
100                 !--------------------------------
101                 daten=dom.GetObject("Ger_Sound_List").State();
102                 word_position = daten.Find(d_Geraet);
103                 if (word_position>-1)
104                 {
105                    daten=daten.Substr(word_position+1,daten.Length()- word_position);
106                    word_position = daten.Find("*");
107                    text_nr = daten.Substr(word_position+1,3);
108
109                    command = command # "," # text_nr;
110                 }
111                 !-----------------------------------
112                 ! Geraetenummer (laufende Nummer)
113                 !-----------------------------------
114                 text_nr = d_Nummer;
115                 command = command # "," # text_nr;
116                 counter = counter + 1;
117              }
118        }
119  }
120  if ( counter > 0 )
121  {
122    dom.GetObject("FunkGong:2").DPByHssDP("SUBMIT").State(command);
123  }
124  else
125  {
126    command= "1,1,108000" # "," # text_batt_ok;
127    dom.GetObject("FunkGong:2").DPByHssDP("SUBMIT").State(command);
128  }
```

Im Script wurden noch folgende Texte (Soundfiles) verwendet:

Die Batterien der folgenden Geräte sind bald leer	– Filenummer 126 – Zeile 16
Alle Batterien sind voll	– Filenummer 213 – Zeile 18
und	– Filenummer 020 – Zeile 84

Die Zahlen von 1 bis 9 liegen im Funk-Gong auf den Speicherplätzen 1 bis 9, die MP3-Files haben also die Namen 001.mp3, 002.mp3 bis 009.mp3.

Gestartet werden kann das Script durch einen Taster, über die Zeitsteuerung oder man stellt in einem Programm zyklisch fest, ob ein Batteriefehler vorliegt, setzt dann eine Zentralenvariable und startet damit die Ausgabe. Das Script bietet viel Spielraum für eigene Versuche.

Eine weitere, sehr interessante Anwendung für die Scriptprogrammierung ist das Arbeiten mit Komponenten, die über eine sogenannte API anzusteuern sind.

Eine API ist eine Programmierschnittstelle, genauer gesagt eine Schnittstelle zur Anwendungsprogrammierung (englisch *Application Programming Interface*, wörtlich „Anwendungs-Programmier-Schnittstelle“).

Anwendungsfälle finden wir z. B. in der Ansteuerung von Audio-Anlagen, der Abfrage von Daten wie z. B. Wetterdaten und viele weitere mehr.

Ausblick

Im fünften Teil dieser Reihe werden wir uns also mit der Kommunikation mit Geräten mit API-Schnittstellen beschäftigen und weitere Scripte schreiben. ELV