Mittlerweile wurde bekannt und auch vom Hersteller eingeräumt, dass auf einer HomeMatic-Zentrale maximal 200 Variablen deklariert werden können. Hierbei sind die in den Skripten intern genutzten Variablen gemeint, nicht die Systemvariablen. Die Zahl 200 markiert dabei die Gesamtanzahl von Skriptvariablen in allen verwendeten Skripten. Bei Überschreiten dieser Zahl kommt es nach einiger Zeit dazu, dass Programme nicht mehr laufen oder die CCU abstürzt.

Die 200 Skriptvariablen erreicht man bei intensiver Nutzung des Systems recht schnell, so führten auch bei mir zuletzt 234 Skriptvariablen regelmäßig zu den genannten Phänomenen.

Falls sich die eigene Installation dieser Größe nähert oder sie gar bereits überschritten hat (ein Skript zum Ermitteln dieses Wertes findet man im HomeMatic-Forum), ist es an der Zeit zu handeln. Die herstellerseitige Lösung des Problems im Rahmen eines Firmwareupdates wurde nicht wirklich in Aussicht gestellt. So sollte man sich anderweitig behelfen.

Im HomeMatic Forum gibt es hierzu mehrere Ansätze. Ich habe mich für die Variante entschieden, die internen Skriptvariablen in allen genutzten Skripten durch „temporäre Variablen“ zu ersetzen und zwar nach dem Schema…

tmpA bis *tmpZ*

…und falls das noch nicht reicht, zusätzlich noch…

tmpA1 bis *tmpZ1*

…, sodass bei konsequenter Nutzung maximal 52 Skriptvariablen benötigt werden. Dort sind auch einige gängige Skripte nach der Methode angepasst hinterlegt.

Man kann die Skripte zwar nahezu nicht mehr verstehen aber wenn es der Sache dient . Ich empfehle, eventuelle Anpassungen und Versuche mit den Original-Skripten anzustellen und diese dann anzupassen, wenn alles klappt.

Übrigens hat es offensichtlich keine negativen Auswirkungen, wenn man ein Skript in der „Skript testen“-Funktion des WebUI oder im erweiterten Skript Parser ausführt. Erst bei Erstellen eines neuen Skriptes bzw. Benutzen der „Fehlerprüfung“ werden die Variablen in den Cache geschrieben und bis zum nächsten Reboot der CCU nicht mehr gelöscht.

Die entsprechend umgestellten wesentlichen Skripte aus meinen Tutorials habe ich nachfolgend zusammengefasst. Falls jemand diesen Lösungsansatz verwenden möchte, spart er sich somit die eigenhändige Umstellung. Die Skripte brauchen – sofern sie nicht verändert wurden – nur eins zu eins ersetzt zu werden.

Derzeit habe ich so insgesamt auf gut 100 Skriptvariablen reduzieren können und es läuft wieder ganz stabil.

HomeMatic – Stromzähler auswerten Version 2 mit HM-EM-TX-WM

Auswerteskript:

```
object tmpA = dom.GetObject("BidCos-RF.MEQ00XXXXX:1.POWER");
object tmpB = dom.GetObject("BidCos-RF.MEQ00XXXXX:1.ENERGY_COUNTER");
var tmpC = dom.GetObject("Strom ENERGY_COUNTER");
var tmpD = dom.GetObject("Strom Referenz Zaehlerstand");
var tmpE = dom.GetObject("Strom Zaehlerstand");
var tmpF = dom.GetObject("Strom Leistungsaufnahme aktuell");
var tmpG = dom.GetObject("Strom Referenz Verbrauch seit letzter Ablesung");
var tmpH = dom.GetObject("Strom Verbrauch seit letzter Ablesung");
var tmpI = dom.GetObject("Strom Referenz Verbrauch heute");
var tmpJ = dom.GetObject("Strom Verbrauch heute");
var tmpK = dom.GetObject("Strom Referenz Verbrauch laufende Woche");
var tmpL = dom.GetObject("Strom Verbrauch laufende Woche");
var tmpM = dom.GetObject("Strom Referenz Verbrauch laufender Monat");
var tmpN = dom.GetObject("Strom Verbrauch laufender Monat");
var tmpO = dom.GetObject("Strom Referenz Verbrauch laufendes Kalenderjahr");
var tmpP = dom.GetObject("Strom Verbrauch laufendes Kalenderjahr");
        if ((tmpB.State() + 0.001) < tmpC.State()) {
        tmpD.State(tmpD.State() + 838.8607);
        }
tmpC.State(tmpB.State());
tmpE.State(tmpD.State() + (tmpB.State()/1000));
tmpF.State(tmpA.State());
tmpH.State(tmpE.State() - tmpG.State());
tmpJ.State(tmpE.State() - tmpI.State());
tmpL.State(tmpE.State() - tmpK.State());
tmpN.State(tmpE.State() - tmpM.State());
tmpP.State(tmpE.State() - tmpO.State());
```

Rücksetzskripte:

```
var tmpA = dom.GetObject("Strom Verbrauch heute");
var tmpB = dom.GetObject("Strom Referenz Verbrauch heute");
tmpB.State(tmpA.State() + tmpB.State());


var tmpA = dom.GetObject("Strom Verbrauch laufende Woche");
var tmpB = dom.GetObject("Strom Referenz Verbrauch laufende Woche");
tmpB.State(tmpA.State() + tmpB.State());


var tmpA = dom.GetObject("Strom Verbrauch laufender Monat");
var tmpB = dom.GetObject("Strom Referenz Verbrauch laufender Monat");
tmpB.State(tmpA.State() + tmpB.State());


var tmpA = dom.GetObject("Strom Verbrauch laufendes Kalenderjahr");
var tmpB = dom.GetObject("Strom Referenz Verbrauch laufendes Kalenderjahr");
tmpB.State(tmpA.State() + tmpB.State());


var tmpA = dom.GetObject("Strom Verbrauch seit letzter Ablesung");
var tmpB = dom.GetObject("Strom Referenz Verbrauch seit letzter Ablesung");
```

```
tmpB.State(tmpA.State() + tmpB.State());
```

HomeMatic – Gaszähler auswerten mit HM-EM-TX-WM

Auswerteskript:

```
object tmpA = dom.GetObject("BidCos-RF.MEQ00XXXXX:1.GAS_POWER");
object tmpB = dom.GetObject("BidCos-RF.MEQ00XXXXX:1.GAS_ENERGY_COUNTER");
var tmpC = dom.GetObject("Gas ENERGY_COUNTER");
var tmpD = dom.GetObject("Gas Referenz Zaehlerstand");
var tmpE = dom.GetObject("Gas Zaehlerstand");
var tmpF = dom.GetObject("Gas Leistungsaufnahme aktuell");
var tmpG = dom.GetObject("Gas Referenz Verbrauch seit letzter Ablesung");
var tmpH = dom.GetObject("Gas Verbrauch seit letzter Ablesung");
var tmpI = dom.GetObject("Gas Referenz Verbrauch heute");
var tmpJ = dom.GetObject("Gas Verbrauch heute");
var tmpK = dom.GetObject("Gas Referenz Verbrauch laufende Woche");
var tmpL = dom.GetObject("Gas Verbrauch laufende Woche");
var tmpM = dom.GetObject("Gas Referenz Verbrauch laufender Monat");
var tmpN = dom.GetObject("Gas Verbrauch laufender Monat");
var tmpO = dom.GetObject("Gas Referenz Verbrauch laufendes Kalenderjahr");
var tmpP = dom.GetObject("Gas Verbrauch laufendes Kalenderjahr");
        if ((tmpB.State() + 0.001) < tmpC.State()) {
        tmpD.State(tmpD.State() + 838.8607);
        }
tmpC.State(tmpB.State());
tmpE.State(tmpD.State() + (tmpB.State()/1000));
tmpF.State(tmpA.State());
tmpH.State(tmpE.State() - tmpG.State());
tmpJ.State(tmpE.State() - tmpI.State());
tmpL.State(tmpE.State() - tmpK.State());
tmpN.State(tmpE.State() - tmpM.State());
tmpP.State(tmpE.State() - tmpO.State());
```

Rücksetzskripte:

```
var tmpA = dom.GetObject("Gas Verbrauch heute");
var tmpB = dom.GetObject("Gas Referenz Verbrauch heute");
tmpB.State(tmpA.State() + tmpB.State());
```

```
var tmpA = dom.GetObject("Gas Verbrauch laufende Woche");
var tmpB = dom.GetObject("Gas Referenz Verbrauch laufende Woche");
tmpB.State(tmpA.State() + tmpB.State());
```

```
var tmpA = dom.GetObject("Gas Verbrauch laufender Monat");
var tmpB = dom.GetObject("Gas Referenz Verbrauch laufender Monat");
tmpB.State(tmpA.State() + tmpB.State());
```

```
var tmpA = dom.GetObject("Gas Verbrauch laufendes Kalenderjahr");
var tmpB = dom.GetObject("Gas Referenz Verbrauch laufendes Kalenderjahr");
tmpB.State(tmpA.State() + tmpB.State());


var tmpA = dom.GetObject("Gas Verbrauch seit letzter Ablesung");
var tmpB = dom.GetObject("Gas Referenz Verbrauch seit letzter Ablesung");
tmpB.State(tmpA.State() + tmpB.State());
```

HomeMatic - Verbrauchs- und Betriebsstundenzähler für Ölheizungen

Einschaltskript:

```
var tmpA= dom.GetObject("Brenner letzte Einschaltzeit");
tmpA.State(system.Date("%F %T")); !Speichern der Einschaltzeit
```

Ausschaltskript:

```
var tmpB= dom.GetObject("Brenner letzte Ausschaltzeit");
var tmpA= dom.GetObject("Brenner letzte Einschaltzeit");
var tmpC= dom.GetObject("Brenner Tankinhalt bei letzter Fuellung").Value();
var tmpD= dom.GetObject("Brenner Tankinhalt aktuell");
var tmpE= dom.GetObject("Brenner Betriebsstunden seit letzter Fuellung");
var tmpF= dom.GetObject("Brenner Verbrauch seit letzter Fuellung");
var tmpG= dom.GetObject("Brenner Betriebsstunden laufendes Kalenderjahr");
var tmpH= dom.GetObject("Brenner Verbrauch laufendes Kalenderjahr");
var tmpI= dom.GetObject("Brenner Betriebsstunden laufender Monat");
var tmpJ= dom.GetObject("Brenner Verbrauch laufender Monat");
var tmpK= dom.GetObject("Brenner Betriebsstunden laufende Woche");
var tmpL= dom.GetObject("Brenner Verbrauch laufende Woche");
var tmpM= dom.GetObject("Brenner Betriebsstunden heute");
var tmpN= dom.GetObject("Brenner Verbrauch heute");

! Speichern der Ausschaltzeit
tmpB.State(system.Date("%F %T"));

! Den Einschaltzeit String aus der Systemvariablen in ein Zeitobjekt
umwandeln
time tmpO = tmpA.Variable().ToTime();

! Die aktuelle (Ausschalt)Zeit String erzeugen und in ein Zeitobjekt
umwandeln
time tmpP = system.Date("%F %T").ToTime();

! Das Zeitobjekt Einschaltzeit in Sekunden seit 1.1.1970 umwandeln
var tmpQ = tmpO.ToInteger();
```

```
! Das Zeitobjekt aktuelle Zeit in Sekunden seit 1.1.1970 umwandeln
var tmpR = tmpP.ToInteger();

!Die Differenz ist die Einschaltdauer in Stunden umgerechnet
var tmpS = 0.01*(tmpR-tmpQ)/36;

! Die Einschaltdauer seit der letzten Füllung kumulieren = Betriebsstunden
seit der letzten Füllung
var tmpT = tmpS + tmpE.Variable();

! Die Betriebsstunden seit der letzten Füllung in die Systemvariable
einstellen
tmpE.State (tmpT);

! Berechnung von Verbrauch in Liter mit 1,87 kg/h seit der letzten Füllung
var tmpU = tmpT * 1.87 * 1.197;

! Verbrauch seit der letzten Füllung in die Systemvariable einstellen
tmpF.State (tmpU);

! Berechnung Tankinhalts in Litern
var tmpV = tmpC - tmpU;

! Tankinhalt in die Systemvariable einstellen
tmpD.State (tmpV);

! Die Einschaltdauer im laufenden Kalenderjahr kumulieren = Betriebsstunden
var tmpW = tmpS + tmpG.Variable();

! Die Betriebsstunden im laufenden Kalenderjahr in die Systemvariable
einstellen
tmpG.State (tmpW);

! Berechnung von Verbrauch in Liter mit 1,87 kg/h im laufenden Kalenderjahr
var tmpX = tmpW * 1.87 * 1.197;

! Verbrauch im laufenden Kalenderjahr in die Systemvariable einstellen
tmpH.State (tmpX);

! Die Einschaltdauer im laufenden Kalendermonat kumulieren = Betriebsstunden
var tmpY = tmpS + tmpI.Variable();

! Die Betriebsstunden im laufenden Kalendermonat in die Systemvariable
einstellen
tmpI.State (tmpY);

! Berechnung von Verbrauch in Liter mit 1,87 kg/h im laufenden Kalendermonat
var tmpZ = tmpY * 1.87 * 1.197;

! Verbrauch im laufenden Kalendermonat in die Systemvariable einstellen
tmpJ.State (tmpZ);
```

```
! Die Einschaltdauer in der laufenden Kalenderwoche kumulieren =
Betriebsstunden
var tmpA1 = tmpS + tmpK.Variable();

! Die Betriebsstunden in der laufenden Kalenderwoche in die Systemvariable
einstellen
tmpK.State (tmpA1);

! Berechnung von Verbrauch in Liter mit 1,87 kg/h in der laufenden
Kalenderwoche
var tmpB1 = tmpA1 * 1.87 * 1.197;

! Verbrauch in der laufenden Kalenderwoche in die Systemvariable einstellen
tmpL.State (tmpB1);

! Die Einschaltdauer heute kumulieren = Betriebsstunden
var tmpC1 = tmpS + tmpM.Variable();

! Die Betriebsstunden heute in die Systemvariable einstellen
tmpM.State (tmpC1);

! Berechnung von Verbrauch in Liter mit 1,87 kg/h heute
var tmpD1 = tmpC1 * 1.87 * 1.197;

! Verbrauch in der laufenden Kalenderwoche in die Systemvariable einstellen
tmpN.State (tmpD1);
```

HomeMatic – Raumklimaüberwachung und Entfeuchtung

Beispiel 1: Schimmelgefahranalyse in einem unisolierten Kellerraum mit manueller Lüftung (Fenster) und einem elektrischen Luftentfeuchter

Auswertung:

```
    object tmpA = dom.GetObject("RaumXY_Raumregler:1");
    object tmpB = tmpA.DPByHssDP("TEMPERATURE");
    object tmpC = tmpA.DPByHssDP("HUMIDITY");
    object tmpD = dom.GetObject("RaumXY_Schimmel");
        object tmpE = dom.GetObject("RaumXY_Lueften");
    object tmpF = dom.GetObject("Aussen_TempFeuSens:1");
    object tmpG = tmpF.DPByHssDP("TEMPERATURE");
    object tmpH = tmpF.DPByHssDP("HUMIDITY");
        ! Programmteil Lüftungsempfehlung
        ! Lokale Variablen
    real tmpI = tmpB.Value(); ! Temperatur in °C innen
    integer tmpJ = tmpC.Value(); ! relative Feuchte in % innen
        real tmpK;  ! absolute feuchte in g/kg innen
        real tmpL = tmpG.Value(); ! Temperatur in °C außen
    integer tmpM = tmpH.Value(); ! relative Feuchte in % außen
```

```
        real tmpN; ! absolute feuchte in g/kg außen
        ! Berechnung der absoluten Feuchte innen
    if (tmpI < 0.0) {tmpI = 0.0;}
    if (tmpI < 10.0)
    { tmpK = (3.78 + (0.285 * tmpI) + (0.0052 * tmpI * tmpI) + (0.0005 * tmpI
* tmpI * tmpI));
    }
    else
    { tmpK = (7.62 + (0.524 * (tmpI-10.0)) + (0.0131 * (tmpI-10.0) *
(tmpI-10.0)) + (0.00048 * (tmpI-10.0) * (tmpI-10.0) * (tmpI-10.0)));
    }
        tmpK = (tmpK * tmpJ) / (100.0 + tmpK * (100.0 - tmpJ) / 622);
        ! Berechnung der absoluten Feuchte außen
        if (tmpL < 0.0) {tmpL = 0.0;}
    if (tmpL < 10.0)
    { tmpN = (3.78 + (0.285 * tmpL) + (0.0052 * tmpL * tmpL) + (0.0005 * tmpL
* tmpL * tmpL));
    }
    else
    { tmpN = (7.62 + (0.524 * (tmpL-10.0)) + (0.0131 * (tmpL-10.0) *
(tmpL-10.0)) + (0.00048 * (tmpL-10.0) * (tmpL-10.0) * (tmpL-10.0)));
    }
        tmpN = (tmpN * tmpM) / (100.0 + tmpN * (100.0 - tmpM) / 622);
        ! Berechnung der Lüftungsempfehlung mit 0,5 g/kg und 0,7 K Hysterese
        if ((tmpN <= (tmpK - 0.8)) && (tmpL <= (tmpI - 1.0)) && (tmpI >
12.7))
        {tmpE.State(true);}
        else
        { if ((tmpN >= (tmpK - 0.3)) || (tmpL >= (tmpI - 0.3)) || (tmpI <=
12.0))
        {tmpE.State(false);}
        }
        ! Programmteil Schimmelwarnung
        ! Berechnung der Oberflächentemperatur der Außenwandecke
    real tmpO; ! Oberfächentemperatur der Außenwandecke in °C
    real tmpP = tmpG.Value(); ! Außentemperatur in °C
    real tmpQ = tmpB.Value(); ! Raumtemperatur in °C
    tmpO = tmpQ + ((0.13 / 0.268) * (tmpP - tmpQ)); ! Rges = 0.268 empirisch
ermittelt
        ! Lokale Variablen
    real    tmpR;    ! Temperatur in °C
    integer tmpS; ! relative Feuchte in %
    real    tmpT;  ! Schimmelwarn-Grenzfeuchte in g/kg
    real    tmpU;  ! Schimmelalarm-Grenzfeuchte in g/kg
    tmpR = tmpO;
    ! Berechnung Warn-Grenzfeuchte
    tmpS = 70;
    if (tmpR < 0.0) {tmpR = 0.0;}
    if (tmpR < 10.0)
    { tmpT = (3.78 + (0.285 * tmpR) + (0.0052 * tmpR * tmpR) + (0.0005 * tmpR
* tmpR * tmpR));
```

```
    }
    else
    { tmpT = (7.62 + (0.524 * (tmpR-10.0)) + (0.0131 * (tmpR-10.0) *
(tmpR-10.0)) + (0.00048 * (tmpR-10.0) * (tmpR-10.0) * (tmpR-10.0)));
    }
        tmpT = (tmpT * tmpS) / (100.0 + tmpT * (100.0 - tmpS) / 622);
    ! Berechnung Alarm-Grenzfeuchte
    tmpS = 80;
    if (tmpR < 0.0) {tmpR = 0.0;}
    if (tmpR < 10.0)
    { tmpU = (3.78 + (0.285 * tmpR) + (0.0052 * tmpR * tmpR) + (0.0005 * tmpR
* tmpR * tmpR));
    }
    else
    { tmpU = (7.62 + (0.524 * (tmpR-10.0)) + (0.0131 * (tmpR-10.0) *
(tmpR-10.0)) + (0.00048 * (tmpR-10.0) * (tmpR-10.0) * (tmpR-10.0)));
    }
        tmpU = (tmpU * tmpS) / (100.0 + tmpU * (100.0 - tmpS) / 622);
        ! Schimmelwarnung
        ! 0 - keine Gefahr
        ! 1 - Warnung
        ! 2 - Alarm
        if ((tmpK > tmpU) && (tmpJ > 64 )) {tmpD.State(2);}
        else {
            if ((tmpK > tmpU) || (tmpK > tmpT)) {tmpD.State(1);}
            else {tmpD.State(0);}
        }
```

Entfeuchter:

```
object tmpA = dom.GetObject("BidCos-RF.FEQ0XXXXXX:1.HUMIDITY");
object tmpB = dom.GetObject("BidCos-RF.FEQ0XXXXXX:1.STATE");
object tmpC = dom.GetObject("UG-Werkstatt_Schimmel");
time tmpD = tmpB.Timestamp();
time tmpE = system.Date("%Y-%m-%d %H:%M:%S").ToTime();
integer tmpF = tmpE.ToInteger() - tmpD.ToInteger();
if (tmpF > 2700) {
  if ((tmpA.Value() > 64) && (tmpB.Value() == 0) && (tmpC.Value() > 1)) {
     tmpB.State(1);
  }
  if ((tmpA.Value() < 64) && (tmpB.Value() == 1)) {
     tmpB.State(0);
  }
}
```

Beispiel 2: Schimmelgefahranalyse in einem isolierten Badezimmer mit Sauna und einem einfachen Wandlüfter

```
    object tmpA = dom.GetObject("RaumXY_Raumregler:1");
    object tmpB = tmpA.DPByHssDP("TEMPERATURE");
    object tmpC = tmpA.DPByHssDP("HUMIDITY");
    object tmpD = dom.GetObject("RaumXY_Schimmel");
        object tmpE = dom.GetObject("RaumXY_Lueften");
    object tmpF = dom.GetObject("Aussen_TempFeuSens:1");
    object tmpG = tmpF.DPByHssDP("TEMPERATURE");
    object tmpH = tmpF.DPByHssDP("HUMIDITY");
        object tmpI = dom.GetObject("BidCos-RF.HEQ0XXXXXX:2.STATE");
        object tmpJ = dom.GetObject("BidCos-RF.HEQ0XXXXXX:1.STATE");
        ! Programmteil Lüftungsempfehlung
        ! Lokale Variablen
    real tmpK = tmpB.Value(); ! Temperatur in °C innen
    integer tmpL = tmpC.Value(); ! relative Feuchte in % innen
        real tmpM;  ! absolute feuchte in g/kg innen
        real tmpN = tmpG.Value(); ! Temperatur in °C außen
    integer tmpO = tmpH.Value(); ! relative Feuchte in % außen
        real    tmpP; ! absolute feuchte in g/kg außen
        ! Berechnung der absoluten Feuchte innen
    if (tmpK < 0.0) {tmpK = 0.0;}
    if (tmpK < 10.0)
    { tmpM = (3.78 + (0.285 * tmpK) + (0.0052 * tmpK * tmpK) + (0.0005 * tmpK
* tmpK * tmpK));
    }
    else
    { tmpM = (7.62 + (0.524 * (tmpK-10.0)) + (0.0131 * (tmpK-10.0) *
(tmpK-10.0)) + (0.00048 * (tmpK-10.0) * (tmpK-10.0) * (tmpK-10.0)));
    }
        tmpM = (tmpM * tmpL) / (100.0 + tmpM * (100.0 - tmpL) / 622);
        ! Berechnung der absoluten Feuchte außen
        if (tmpN < 0.0) {tmpN = 0.0;}
    if (tmpN < 10.0)
    { tmpP = (3.78 + (0.285 * tmpN) + (0.0052 * tmpN * tmpN) + (0.0005 * tmpN
* tmpN * tmpN));
    }
    else
    { tmpP = (7.62 + (0.524 * (tmpN-10.0)) + (0.0131 * (tmpN-10.0) *
(tmpN-10.0)) + (0.00048 * (tmpN-10.0) * (tmpN-10.0) * (tmpN-10.0)));
    }
        tmpP = (tmpP * tmpO) / (100.0 + tmpP * (100.0 - tmpO) / 622);
        ! Berechnung der Lüftungsempfehlung mit 0,5 g/kg und 0,7 K Hysterese
        if ((tmpP <= (tmpM - 0.8)) && (tmpN <= (tmpK - 1.0)) && (tmpK >
19.7))
        {tmpE.State(true);}
        else
        { if ((tmpP >= (tmpM - 0.3)) || (tmpN >= (tmpK - 0.3)) || (tmpK <=
19.0))
        {tmpE.State(false);}
        }
        ! Programmteil Schimmelwarnung
        ! Berechnung der Oberflächentemperatur der Außenwandecke
```

```
    real tmpQ; ! Oberfächentemperatur der Außenwandecke in °C
    real tmpR = tmpG.Value(); ! Außentemperatur in °C
    real tmpS = tmpB.Value(); ! Raumtemperatur in °C
    tmpQ = tmpS + ((0.13 / 0.42) * (tmpR - tmpS)); ! Rges = 0.42 empirisch
ermittelt
        ! Lokale Variablen
    real    tmpT;    ! Temperatur in °C
    integer tmpU; ! relative Feuchte in %
    real    tmpV;  ! Schimmelwarn-Grenzfeuchte in g/kg
    real    tmpW;  ! Schimmelalarm-Grenzfeuchte in g/kg
    tmpT = tmpQ;
    ! Berechnung Warn-Grenzfeuchte
    tmpU = 70;
    if (tmpT < 0.0) {tmpT = 0.0;}
    if (tmpT < 10.0)
    { tmpV = (3.78 + (0.285 * tmpT) + (0.0052 * tmpT * tmpT) + (0.0005 * tmpT
* tmpT * tmpT));
    }
    else
    { tmpV = (7.62 + (0.524 * (tmpT-10.0)) + (0.0131 * (tmpT-10.0) *
(tmpT-10.0)) + (0.00048 * (tmpT-10.0) * (tmpT-10.0) * (tmpT-10.0)));
    }
        tmpV = (tmpV * tmpU) / (100.0 + tmpV * (100.0 - tmpU) / 622);
    ! Berechnung Alarm-Grenzfeuchte
    tmpU = 80;
    if (tmpT < 0.0) {tmpT = 0.0;}
    if (tmpT < 10.0)
    { tmpW = (3.78 + (0.285 * tmpT) + (0.0052 * tmpT * tmpT) + (0.0005 * tmpT
* tmpT * tmpT));
    }
    else
    { tmpW = (7.62 + (0.524 * (tmpT-10.0)) + (0.0131 * (tmpT-10.0) *
(tmpT-10.0)) + (0.00048 * (tmpT-10.0) * (tmpT-10.0) * (tmpT-10.0)));
        }
        tmpW = (tmpW * tmpU) / (100.0 + tmpW * (100.0 - tmpU) / 622);
        ! Schimmelwarnung
        ! 0 - keine Gefahr
        ! 1 - Warnung
        ! 2 - Alarm
        if ((tmpM > tmpW) && (tmpL > 64 )) {tmpD.State(2);}
        else {
            if ((tmpM > tmpW) || (tmpM > tmpV)) {tmpD.State(1);}
            else {tmpD.State(0);}
        }
        ! Programmteil Ventilatorsteuerung
    ! Prüfen, ob Raum schon zu kalt
        if (tmpS > 20.0)
                { if ((tmpP <= (tmpM - 0.8)) && (tmpR <= (tmpS - 1.0)) &&
(tmpJ.Value() == 0) && (tmpD.Value() == 2))
                        {tmpI.State(1);}
                        else
```

```
                        { if ((tmpP >= (tmpM - 0.3)) || (tmpR >= (tmpS -
0.3)) || (tmpJ.Value() == 1) || (tmpD.Value() == 1) || (tmpD.Value() == 0))
                        {tmpI.State(0);}
                        }
                }
        else
                { if (tmpS <= 19.5)
                {tmpI.State(0);}
                }
```

HomeMatic - Sonnenstandberechnung

Auswerteskript:

```
! Sonnenstand Analemma

real tmpA = system.Date("%j").ToInteger();
real tmpB = (0.01 * system.Date("%M").ToInteger()) +
system.Date("%H").ToInteger();
real tmpC = (0.01 * system.SunriseTime("%M").ToInteger()) +
system.SunriseTime("%H").ToInteger();
real tmpD = (0.01 * system.SunsetTime("%M").ToInteger()) +
system.SunsetTime("%H").ToInteger();
integer tmpE = 0; ! Sonne unter Horizont

! Analemma 23.12. bis 10.02.
if ((tmpA > 356) || ((tmpA > 0) && (tmpA < 41)))
{
        if (tmpB < tmpD) {
                tmpE = 4; ! Süd/West
        }
        if (tmpB < tmpD - 2) {
                tmpE = 3; ! Süd
        }
        if (tmpB < tmpC + 2) {
                tmpE = 2; ! Süd/Ost
        }
        if (tmpB < tmpC -1) {
                tmpE = 0; ! Sonne unter Horizont
        }
}
! Analemma 11.02. bis 07.03. und 01.11. bis 22.12.
if (((tmpA > 304) && (tmpA < 357)) || ((tmpA > 40) && (tmpA < 67)))
{
        if (tmpB < tmpD) {
                tmpE = 4; ! Süd/West
        }
        if (tmpB < tmpD - 3) {
                tmpE = 3; ! Süd
```

```
        }
        if (tmpB < tmpC + 3) {
                tmpE = 2; ! Süd/Ost
        }
        if (tmpB < tmpC -1) {
                tmpE = 0; ! Sonne unter Horizont
        }
}
! Analemma 08.03. bis 12.04. und 05.10. bis 31.10.
if (((tmpA > 277) && (tmpA < 305)) || ((tmpA > 66) && (tmpA < 103)))
{
        if (tmpB < tmpD) {
                tmpE = 5; ! West
        }
        if (tmpB < tmpD - 1) {
                tmpE = 4; ! Süd/West
        }
        if (tmpB < tmpD - 4) {
                tmpE = 3; ! Süd
        }
        if (tmpB < tmpC + 4) {
                tmpE = 2; ! Süd/Ost
        }
        if (tmpB < tmpC + 1) {
                tmpE = 1; ! Ost
        }
        if (tmpB < tmpC -1) {
                tmpE = 0; ! Sonne unter Horizont
        }
}
! Analemma 13.04. bis 30.04. und 30.08. bis 04.10.
if (((tmpA > 241) && (tmpA < 278)) || ((tmpA > 102) && (tmpA < 121)))
{
        if (tmpB < tmpD) {
                tmpE = 5; ! West
        }
        if (tmpB < tmpD - 3) {
                tmpE = 4; ! Süd/West
        }
        if (tmpB < tmpD - 5) {
                tmpE = 3; ! Süd
        }
        if (tmpB < tmpC + 5) {
                tmpE = 2; ! Süd/Ost
        }
        if (tmpB < tmpC + 3) {
                tmpE = 1; ! Ost
        }
        if (tmpB < tmpC -1) {
                tmpE = 0; ! Sonne unter Horizont
        }
```

```
}
! Analemma 01.05. bis 09.06. und 12.08. bis 29.08.
if (((tmpA > 192) && (tmpA < 242)) || ((tmpA > 120) && (tmpA < 161)))
{
        if (tmpB < tmpD) {
                tmpE = 5; ! West
        }
        if (tmpB < tmpD - 4) {
                tmpE = 4; ! Süd/West
        }
        if (tmpB < tmpD - 6) {
                tmpE = 3; ! Süd
        }
        if (tmpB < tmpC + 6) {
                tmpE = 2; ! Süd/Ost
        }
        if (tmpB < tmpC + 4) {
                tmpE = 1; ! Ost
        }
        if (tmpB < tmpC -1) {
                tmpE = 0; ! Sonne unter Horizont
        }
}
! Analemma 10.06. bis 11.08. und 01.11. bis 22.12.
if ((tmpA > 160) && (tmpA < 193))
{
        if (tmpB < tmpD) {
                tmpE = 5; ! West
        }
        if (tmpB < tmpD - 5) {
                tmpE = 4; ! Süd/West
        }
        if (tmpB < tmpD - 7) {
                tmpE = 3; ! Süd
        }
        if (tmpB < tmpC + 7) {
                tmpE = 2; ! Süd/Ost
        }
        if (tmpB < tmpC + 5) {
                tmpE = 1; ! Ost
        }
        if (tmpB < tmpC -1) {
                tmpE = 0; ! Sonne unter Horizont
        }
}
dom.GetObject("Sonnenstand").State(tmpE);
```

Bitte beachten...

Die Verwendung meiner Hinweise, Anleitungen, Schaltungen und Software erfolgt auf eigenes Risiko. Ich übernehme hierfür keinerlei Gewährleistung bzw. Haftung.