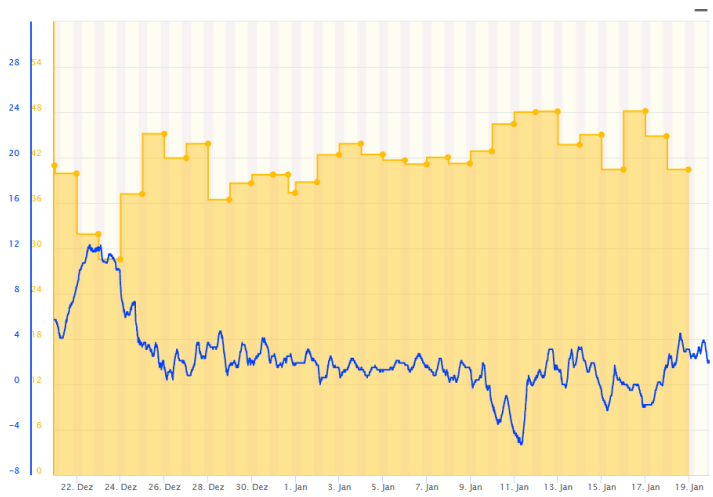
Projektvorstellung

Teil 1: Grundlagen
Teil 2: elektrischer Aufbau
Teil 3: Programme

Paradigma PELEO
Pelletsheizung mit HomeMatic überwachen

Nachdem nun die Hardware brummt, geht's im dritten und letzten Teil um die Programme. Insgesamt sind es für die verschiedenen Berechnungen und Funktionen 11 Programme, die vor allem Skripte beinhalten.
Zur Visualisierung auf dem Smartphone habe ich eine Fernbedienungsseite in AIO Neo erstellt. Die Funktionen in der App habe ich im ersten Teil schon vorgestellt.



Tagesverbräuche und Aussentemperatur

Elf Programme aus drei Signalen schaut erst einmal brachial überdimensioniert aus. Tatsächlich benutze ich 8 Programme direkt oder indirekt zur Berechnung der Werte, wobei zwangsläufig die Ergebnisse aufeinander aufbauen. Die übrigen 3 Programme dienen zur Bedienung über die AIO Neo Remote und haben die Funktionen zu- und abbuchen des Pelletslagerstands und Rücksetzen des Aschestands.

Was kommt hinten raus?

Zuerst einmal der Überblick, welche Meßwerte und Infos vom System erzeugt werden:

Funktion	Variable	Maßeinheit
Verbrauchsstatistik		
Pelletverbrauch heute in kg (fortlaufend)	Pellets_kg_heute	kg
Tagesverbrauch gestern in kg (Tagesstatistik)	Pellets_kg_gestern	kg
Tagesverbrauch gestern umgerechnet in kWh	Pellets_kWh_gestern	kWh
Verbrauch laufender Monat	Pellets_kg_lfd_Monat	kg
Verbrauch letzter Monat (Monatsstatistik)	Pellets_kg_letzter_Monat	kg
Verbrauch laufendes Jahr	Pellets_kg_lfd_Jahr	kg
Verbrauch letztes Jahr (Statistik)	Pellets_kg_letztes_Jahr	kg
Pelletsverbrauch insgesamt (fortlaufend)	Pellets_kg_total	kg
Durchschnittsverbrauch der letzten 10 Tage in kg	Pellets_kg_Avg10	kg
Betriebsinformationen		
Lagerinhalt Pelletslager in Tonnen	Pellets_Lager_t	t
Lagerinhalt Pelletslager in Prozent	Pellets_Lagerfuellgrad	%
geschätzte Reichweite des Lagers in Tagen	Pellets_Lagerreichweite	d
ungefähre aktuelle Brennerleistung in Prozent	Pellets_Brennerstatus	%
geschätzter Füllstand der Aschebox	Pellets_Aschebox_kg	kg
geschätzter Füllgrad der Aschebox	Pellets_Aschebox_Fuellgrad	%
Meldung Aschestand über Schwellwert	Pellets_Aschebox_voll	-

Meldung Lagerstand unter Schwellwert	Pellets_Lager_niedrig	-
geschätzte Dauer der kommenden Lagerentnahme	Pellets_RA-Dauer_erwartet	min
tatsächliche Dauer der letzten Lagerentnahme	Pellets_RA-Dauer_letzte	min
Abweichung Lagerentnahmedauer in Prozent	Pellets_RA-Dauer_Abweichung	%

Die Werte im oberen Teil der Tabelle geben verschiedene Ansichten der Verbrauchswerte wieder, die teilweise auch auf die Erfassung mit Statistik-Tools wie dem CCU-Historian optimiert sind.
Mit dem Tagesverbrauch gestern läßt sich z.B. gut eine Grafik wie auf der ersten Seite realisieren, die mit nicht zu hohem Detaillierungsgrad den Verbrauch darstellt.

Im unteren Teil sind Variablen zusammengestellt, die Betriebsinformationen über die Anlage anzeigen, zum Beispiel den Lagerstand und Infos, ob es Zeit wird, Pellets nachzubestellen oder die Aschebox zu leeren.
Die letzten drei Variablen sind evtl. erklärungsbedürftig. Hier geht es um die Überwachung der Saugförderung aus dem Lager in den Zwischenbehälter des Heizkessels. Im ersten Teil habe ich geschildert, daß es hier ein, zwei Mal vorgekommen ist, daß die Saugentnahme nicht so gut funktioniert hat, weil sich der Maulwurf irgendwo in einer ungünstigen Ecke verlaufen hatte.
Hier wollte ich eine Art „Früherkennung“ bauen, die eine Meldung erzeugt, bevor es zu einer Störung kommt. Dazu berechne ich zu Beginn jeder Saugentnahme aus dem Rest-Füllstand des Zwischenbehälters einen Schätzwert für die Saugdauer. Die Ermittlung der Formel dazu ist in Teil 1 beschrieben. Die tatsächlich benötigte Dauer wird gemessen und in der Variable `Pellets_RA-Dauer_letzte` gespeichert. Aus diesen Werten wird die Abweichung der tatsächlichen Saugdauer von der berechneten Dauer ermittelt und als Prozentwert in `Pellets_RA-Dauer_Abweichung` abgelegt. Über diesen Wert wird eine Schwelle gelegt, die bei grober Abweichung eine Meldung ausgibt.

Einige Werte sind sowohl als Absolut- als auch als Prozentwert verfügbar. Der Prozentwert wird dabei meistens für die grafische Darstellung in der App verwendet.

Das kommt rein

Als Eingangssignale werden, wie im zweiten Teil beschrieben, die drei Inputs des Wired-I/O-Moduls verwendet.
Sie sind entsprechend der Signale der Kesselsteuerung benannt. Ein vorangestelltes *n* zeigt an, daß die Signale negiert sind. Der Signalzustand *aus* entspricht also in Realität dem aktiven elektrischen Zustand und umgekehrt. Das muß bei der Programmierung im Hinterkopf behalten werden.

Peleo-nStoerung	HMW-IO-12-Sw14-DR		Wired RS485 I/O-Modul 12 Eingänge, 14 Ausgänge, Hutschienenmontage
Peleo-nRA	HMW-IO-12-Sw14-DR		Wired RS485 I/O-Modul 12 Eingänge, 14 Ausgänge, Hutschienenmontage
Peleo-nRES1	HMW-IO-12-Sw14-DR		Wired RS485 I/O-Modul 12 Eingänge, 14 Ausgänge, Hutschienenmontage

Für die oben beschriebene Kontrolle der Saugaustragung muß die Zeitdauer des Signals *Peleo-nRA* gemessen werden.
Dazu verwende ich eine Funktion des CUx-Daemons, nämlich das *State-Monitor Device*.
Dieses virtuelle Gerät ermöglicht es auf einfache und ressourcenschonende Weise, Ein- und Ausschaltdauer eines Gerätes zu überwachen.
Hier wird es mit dem Signal *Peleo-nRA* verbunden und bekommt den Namen Peleo_RA_StateMon. In der Geräteliste stellt es sich dann als virtuelles Gerät so dar:

Peleo_RA_StateMon	HM-Sen-Wa-Od	
Peleo_RA_StateMon:1 Schaltaktor	HM-Sen-Wa-Od	
Peleo_RA_StateMon:2	HM-Sen-Wa-Od	
Peleo_RA_StateMon:3	HM-Sen-Wa-Od	

Das Gerät muß zuerst im CUx-Daemon erzeugt (Universal Wrapper Device, Funktion „State“) und dann in der CCU konfiguriert:

Name	Typenbe-zeichnung	Bild
Peleo_RA_StateMon	HM-Sen-Wa-Od	

Geräteparameter

USE_HMDATAPT	☑	
HMSERIAL	[redacted]:16	SERIAL:X
HMDATAPT	STATE	
HSS_TYPE	DIGITAL_INPUT	
Fehler	OK!	
TRIGGER_SET	0.0	(-99999.0-99999.0)
TRIGGER_CMP	EQ	
TIMER_SET	0	s (0-99999)
SUM_CALC	☑	
TIME_ON_SUM	250.8	min (0.0-9999999.0)
SWITCH_SUM	95	(0-9999999)

HMSERIAL ist die Seriennummer des zu überwachenden Geräts bzw. Kanals, das ist hier die Seriennummer des I/O-Moduls und Kanal 16 für den I/O-Pin des nRA - Signals.
TRIGGER_SET wird auf 0 und die Vergleichsbedingung TRIGGER_CMP auf EQ gestellt.
Der State-Monitor betrachtet den Zustand des Kanals damit als aktiv, wenn er „gleich 0“ entspricht.
Durch die Invertierung des Signals entspricht das dem Zustand, wenn der Motor des Maulwurfs läuft.
Dies wird im State-Monitor Device als „Schaltzustand ein“ dargestellt.

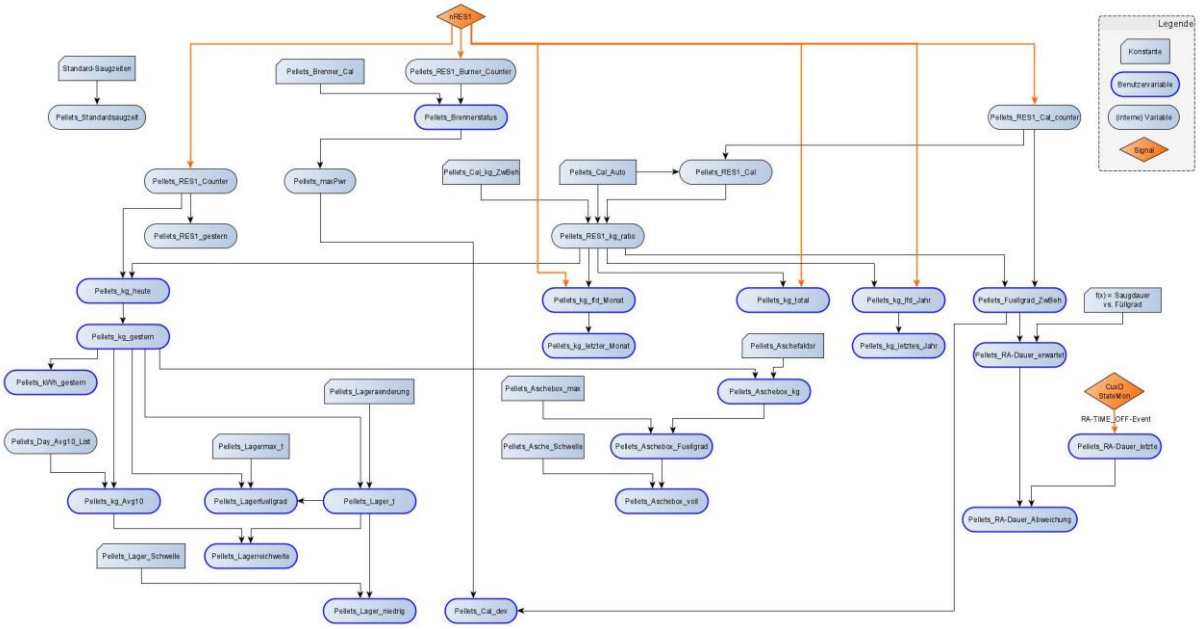
Name	Raum	Gewerk	Letzte Änderung	Control
Filter	Filter	Filter		
Peleo_RA_StateMon:1 Schaltaktor	Heizraum	Heizung	20.01.2021 00:06:00	Schaltzustand: aus [SWITCH_1H]: 0.00 [TIME_ON_1H]: 0.00 min [TIME_OFF_1H]: 60.00 min [TIME_ON]: 2.40 min [TIME_OFF]: 287.93 min [TIME_ON_SUM]: 250.80 min [SWITCH_SUM]: 95.00 [TIME_ON_EVENT]: false [TIME_OFF_EVENT]: false [TIME_STATE=FALSE]

Der Kanal :1 des State-Monitor Device beinhaltet die für uns später relevante Information über die Einschaltdauer des letzten Zeitintervalls mit Schaltzustand *ein*.
Im Bild oben war dies z.B. [TIME_ON] 2.40 min.
Dieser Wert wird dann später weiterverarbeitet.

Variablen, Variablen….

Bevor es mit dem Programmieren losgeht, müssen die Systemvariablen angelegt und ggf. mit Vorgabewerten belegt werden.
Neben den *Benutzervariablen*, die oben in der Liste aufgeführt sind und (mehr oder weniger) sinnvolle Werte anzeigen, gibt es noch eine Reihe interner Variablen, die Zwischenwerte speichern oder Zähler für die Systemevents (Signalwechsel etc.) darstellen.

Außerdem werden einige Konstanten benötigt, die Anlagenparameter bzw. -konfigurationen speichern. Diese sind zum Teil auch in Systemvariablen abgelegt, damit sie einfach zu ändern und nachzujustieren sind.
Auch die Konstanten müssen zunächst als Systemvariablen definiert und mit Vorgabewerten belegt werden. Ein paar Konstanten sind gar nicht so konstant, sondern können ggf. auch automatisch nachkonfiguriert werden oder über die grafische Oberfläche eingestellt werden (z.B. die Lager- und Ascheschwelle).
Bei den Konstanten gibt es eine wichtige Ausnahme, nämlich der Zusammenhang zwischen Füllgrad_Zwischenbehälter und erwarteter Saugdauer. Dies ist eine Formel in Form einer linearen Funktion $f(x) = a \cdot x + b$, wobei die Parameter a und b entsprechend der Heizungsanlage bestimmt werden müssen (siehe Teil 1). Die Formel muß an zwei Stellen von Hand in die Skripte eingetragen werden.



Abhängigkeiten der Variablen und Konstanten

Das Diagramm oben zeigt die Abhängigkeiten der Variablen voneinander. Benutzervariablen sind blau umrandet, interne Variablen haben einen schwarzen Rand. Die Rechtecke mit Eselsohr symbolisieren Konstanten. Mit den orangen Pfeilen habe ich einmal die direkte Beeinflussung von Variablen durch Signale dargestellt. Zum Beispiel wird der *Pellets_RES1_Counter*, von dem aus praktisch alle Verbrauchswerte ermittelt werden, durch das Signal *nRES1* erhöht.

Die Bezeichnung aller Systemvariablen beginnt mit *Pellets_* .
Innerhalb der Skripte war ich leider nicht so systematisch mit den Bezeichnern, hier hat die Benennung eine gewisse Evolution erfahren.

Im Anhang findet sich ein Datenfussdiagramm, daß – sagen wir mal mit etwas Übung einen Art von – Überblick über die Variablen und ihre Verwendung in und Manipulation durch die Programme und Skripte gibt. Man sollte sich dadurch aber nicht erschrecken lassen – die Übersicht leidet möglicherweise dann doch ein wenig an der Menge der Datenpunkte.
Die Diagramme wurden übrigens mit dem yEd Graph Editor erstellt, der für solche Aktionen wirklich klasse ist.

Konstanten

Im folgenden sind zunächst alle Konstanten mit jeweils einer kurzen Erläuterung aufgeführt.
Das Setzen der Werte habe ich manuell über die Funktion „Skript testen“ durchgeführt, das war am einfachsten, da der Funktionsumfang ja nach und nach gewachsen ist.

Pellets_Lagermax_t
Zahl: Fassungsvermögen (max. Füllmenge) des Pelletlagers in t
Wertebereich: 0…50 t
verwendet in: Peleo Tagesstatistik (Verwendung)
Konstante, muß ausgemessen und manuell gesetzt werden. Wird verwendet, um - den Lagerfüllstand in % zu berechnen Beispiel: `dom.GetObject("Pellets_Lagermax_t").State(7.0);`

Pellets_Lager_Schwelle
Zahl: Lagerstand in kg, ab dem eine Warnmeldung „Lagerstand niedrig“ ausgegeben wird
Wertebereich: 0…65000 kg
verwendet in: Peleo Tagesstatistik (Verwendung)
Eine Konstante, die z.B. per Slider in der AIO Neo – Bedienoberfläche angepaßt werden kann. Muß manuell gesetzt werden. Beispiel: `dom.GetObject("Pellets_Lager_Schwelle").State(1000);`

Pellets_Cal_kg_ZwBeh
Zahl: max. Füllmenge kg des Zwischenbehälters. Verwendet zur Kalibrierung der Pelletmenge in kg
Wertebereich: 0…100 kg
verwendet in: Peleo-RA geplant (Verwendung) Peleo-RA early (Verwendung)
Konstante! Muß ausgemessen und manuell gesetzt werden. Wird verwendet, um - die Austragungsmenge der Kugelschleuse zu kalibrieren - den Füllgrad des Zwischenbehälters zu berechnen Wert aus Peleo-Bedienungsanleitung: 32kg und für meine 16 kW-Kesselversion manuell überprüft. Beispiel: `dom.GetObject("Pellets_Cal_kg_ZwBeh").State(32.0);`

Pellets_Cal_Auto
Logikwert: Automatische Kalibrierung Volumen Kugelschleuse aktiviert / deaktiviert
Wertebereich: ein / aus
verwendet in: Peleo-RA geplant (zur Berechnung) Peleo-RA early (zur Berechnung)
Der Wert der Statusvariablen wird manuell eingestellt: ***ein***: Das Verhältnis des Volumens der Kugelschleuse zum Volumen des Zwischenbehälters wird automatisch kalibriert (Wert der Variablen *Pellets_RES1_Cal*). Dies geschieht: a) in der Routine *Peleo-RA early* (der Zwischenbehälter ist „ganz leer“, damit ist die Anzahl der Kugelschleusen-Umläufe bekannt, um den Zwischenbehälter vollständig zu leeren (*Pellets_RES1_Cal_counter*). b) in der Routine *Peleo-RA geplant* (der Zwischenbehälter ist nicht ganz leer, aber die Anzahl der Kugelschleusen-Umläufe ist jetzt schon höher, als der gesetzte Wert. Also muß der Wert größer sein und wird auf den aktuellen Wert angehoben (sukzessive Annäherung). ***aus***: Die automatische Kalibrierung ist deaktiviert. Sobald ein stabiler (und ggf. verifizierter) Kalibrierungswert vorliegt, kann es Sinn machen, diesen einzufrieren, damit er nicht versehentlich verschlechtert wird. Das kann insbesondere dann passieren, wenn der Pelletskessel neu eingeschaltet wird und dann außerhalb der Standardsaugzeiten den Zwischenbehälter auffüllt. Generell sollte bei jeder außerplanmäßigen Befüllung des Zwischenbehälters die Autokalibrierung ausgeschaltet werden, sofern diese nicht durch einen leeren Zwischenbehälter ausgelöst wird. Beispiel: `dom.GetObject("Pellets_Cal_Auto").State(0);`

Pellets_Aschefaktor
Zahl: Aschemenge in kg, die bei Verbrennung von 1 kg Pellets anfällt
Wertebereich: 0…65000
verwendet in: Peleo Tagesstatistik (Verwendung)
Muß manuell gesetzt werden. Der Wert muß am besten ausgemessen werden oder als Faustformel (z.B. 0,5% aus Norm) angenommen werden Beispiel: `dom.GetObject("Pellets_Aschefaktor").State(0.005);`

Pellets_Aschebox_max
Zahl: maximale Füllmenge der Aschebox in kg
Wertebereich: 0…255 kg
verwendet in: Peleo Tagesstatistik (Verwendung)
Muß manuell ausgemessen und gesetzt werden. Aus Hersteller-Unterlagen: 7,5 kg Beispiel: `dom.GetObject("Pellets_Aschebox_max").State(7.5);`

Pellets_Asche_Schwelle
Zahl: Füllgrad der Aschebox in %, bei der eine Warnmeldung ausgegeben wird
Wertebereich: 0…100 %
verwendet in: Peleo Tagesstatistik (Verwendung)
Muß manuell gesetzt werden. Kann auch über die AIO Neo – Oberfläche eingestellt werden. Beispiel: `dom.GetObject("Pellets_Asche_Schwelle").State(90);`

Pellets_Lageraenderung
Zahl: Vorgabewert für Lagerzu- oder -abgang in kg
Wertebereich: 0…65000 kg
verwendet in: Peleo Tagesstatistik (Verwendung)
kg-Wert, der bei Lagerzu- oder abgang verbucht wird. Der Wert kann z.B. über einen Slider in AIO Neo eingestellt werden, bevor die Zu- oder Abgangsfunktion aufgerufen wird. Auf diese Weise läßt sich der Wert, um den der Lagerstand verändert werden soll, leicht über die grafische Oberfläche einstellen. Muß manuell gesetzt / initialisiert werden. Beispiel: dom.GetObject("Pellets_Lageraenderung").State(1000);

Pellets_Brenner_Cal	ab Version: B2
Zahl: Kalibrierungsfaktor für Leistungsanzeige Brenner	
Wertebereich: 0.0 … 100.0	
verwendet in: Peleo_Brennerstatus_berechnen (Berechnung / Verwendung)	
Kalibrierung für die Brenner-Leistungsanzeige. Gibt an, wie viele nRES1-Pulse pro 15 min als 100% Brennerleistung angezeigt werden. Konstante, die z.B. aus CCU-Historian-Diagramm ausgezählt werden kann. Der Wert muß manuell gesetzt werden (für Auto-Cal vorgesehen). Beispiel: dom.GetObject("Pellets_Brenner_Cal").State(15.0);	

Konstanten, die in Programmen festgelegt werden:

Am Kessel eingestellte Standard-Saugzeiten
Zeitfenster – muß im Programm gesetzt werden
verwendet in: Peleo Standardsaugzeit (Einstellung und zur Berechnung)
Wird verwendet, um außerplanmäßige Lageraustragung zu erkennen. In diesem Fall wird der Zwischenbehälter als vollständig entleert angenommen. Dient zur Kalibrierung der Austragungseinheiten der Kugelschleuse. Die Zeitfenster entsprechend der am Pelletkessel eingestellten geplanten Saugzeiten einstellen. Empfehlung: +/- 15 min. Beispiel: geplante Saugzeit = 12:15 Uhr -> Zeitfenster 12:00 – 12:30 Uhr (Wird weiter unten bei den Programmen beschrieben.)

Zusammenhang Saugdauer zu Füllstand Pelletsberechnung
Formel – muß ermittelt und in 2 Skripten eingesetzt werden
verwendet in: Peleo-RA geplant (zur Berechnung) Peleo-RA early (zur Berechnung)
Die Formel muß aus gemessenen RA-Dauern und zugehörigen Restfüllständen ermittelt werden (z.B. Excel x/y-Diagramm). Die Werte können z.B. über den CCU-Historian einfach abgelesen werden. Sollte eine lineare Funktion sein. Die Ermittlung der Funktion wird in Teil 1 beschrieben. Wird verwendet, um Abweichungen der realen Saugzeit zu erkennen (Probleme bei der Lageraustragung frühzeitig erkennen). Beispiel: `ra_dauer_estim = (-0.0408*fuellgrad_zwbeh)+4.0278;` Achtung: Die Formel muß in zwei Skripten angepaßt werden (siehe oben).

Variablen

Nun geht es an die Definition und Erklärung der Variablen.
In diesem Abschnitt sind die Benutzervariablen mit einem blauen Kopf gekennzeichnet, die internen Variablen in grau.

Pellets_RES1_Counter
Zahl: Anzahl der Umläufe der Kugelschleuse pro Tag
Wertebereich: 0…65000
verwendet in: Peleo-RES1 (Aktualisierung) Peleo Tagesstatistik (Verwendung, Rücksetzen)
Dieser Zähler ist der Ausgangspunkt für die meisten Mengenberechnungen. (außer den langfristigen Zählern Monat, Jahr und Total, die separat laufen). Der Zähler wird bei jedem Umlauf der Kugelschleuse erhöht und täglich um Mitternacht zurückgesetzt.

Pellets_RES1_gestern
Zahl: Anzahl der Umläufe der Kugelschleuse gestern total
Wertebereich: 0…65000
verwendet in: Peleo Tagesstatistik (Berechnung)
Diese Variable ist eine Kopie des RES1_Counters, die täglich um Mitternacht aktualisiert wird. Sie ist für's Debugging verwendet worden und hat keinen echten Nutzwert.

Pellets_kg_heute
Zahl: Aktueller Tagesverbrauch Pellets in kg (wird Mitternacht zurückgesetzt)
Wertebereich: 0…5000 kg
verwendet in: Peleo-RES1 (Berechnung) Peleo Tagesstatistik (Verwendung, Rücksetzen)
Wird aus `Pellets_RES1_Counter` und der pro Umlauf transportierten Pelletmenge berechnet. Die Variable wird bei jedem Umlauf aktualisiert und spiegelt daher den Verbrauch des laufenden Tages aktuell wieder.

Pellets_kg_gestern
Zahl: Tagesverbrauch Pellets gestern in kg (Tagessumme, Berechnung Mitternacht)
Wertebereich: 0…5000 kg
verwendet in: Peleo Tagesstatistik (Berechnung)
Verbrauchssumme des vergangenen Tages. Wird um Mitternacht durch Kopie von `Pellets_kg_heute` erzeugt. Der Wert eignet sich zur Darstellung des Tagesverbrauchs in Diagrammen und Statistiken, z.B. mit dem CCU Historian (siehe Beispiel-Diagramm auf der ersten Seite).

Pellets_kWh_gestern
Zahl: Tagesverbrauch gestern umgerechnet in kWh (Tagessumme, Berechnung Mitternacht)
Wertebereich: 0…65000 kWh
verwendet in: Peleo Tagesstatistik (Berechnung)
Der Wert wird durch eine grobe Pi-x-Daumen – Umrechnung aus `Pellets_kg_gestern` gewonnen. Er soll ein besseres Gefühl für den Verbrauch in einer gängigen Energie-Einheit geben. Da weder der Energiegehalt der Pellts noch der Wirkungsgrad des Kessels exakt bekannt (und wahrscheinlich auch variabel) sind, ist das lediglich ein grobes Schätzeisen. Die Umrechnung erfolgt über die Formel: 4.8 kWh pro Kilogramm Pellets * 0.9 Wirkungsgrad = 4.32 kWh/kg

Pellets_kg_lfd_Monat
Zahl: Pelletverbrauch laufender Monat in kg
Wertebereich: 0…65000 kg
verwendet in: Peleo-RES1 (Erhöhung) Peleo Tagesstatistik (Verwendung, Rücksetzen)
Die Variable gibt den aktuellen Verbrauch in diesem Monat an. Der Wert wird bei jedem Umlauf der Kugelschleuse (nRES1) aktualisiert. Am Monatsende wird der Wert bei Berechnung der Statistiken um Mitternacht zurückgesetzt. Da durch die laufende Aktualisierung sehr viele Werte entstehen, sollte bei Aufzeichnung durch den CCU Historian unbedingt eine Kompressionsmethode eingeschaltet werden.

Pellets_kg_letzter_Monat
Zahl: Pelletverbrauch letzter Monat in kg
Wertebereich: 0…65000 kg
verwendet in: Peleo Tagesstatistik (Berechnung)
Verbrauchssumme des vergangenen Monats. Wird am 1. eines Monats nach Mitternacht durch Kopie von `Pellets_kg_lfd_Monat` erzeugt. Der Wert eignet sich zur Darstellung des Tagesverbrauchs in Diagrammen und Statistiken, z.B. mit dem CCU Historian (siehe Beispiel-Diagramm auf der ersten Seite).

Pellets_kg_lfd_Jahr
Zahl: Pelletverbrauch laufendes Jahr in kg
Wertebereich: 0…65000 kg
verwendet in: Peleo-RES1 (Erhöhung) Peleo Tagesstatistik (Verwendung, Rücksetzen)
Aktueller Verbrauch im laufenden Jahr. Dieser Wert wird analog zum Monatsverbrauch separat berechnet und bei jedem RES1-Signal aktualisiert.

Pellets_kg_letztes_Jahr
Zahl: Pelletverbrauch letztes Jahr in kg
Wertebereich: 0…65000 kg
verwendet in: Peleo Tagesstatistik (Berechnung)
Verbrauchssumme des vergangenen Jahres für die Statistik. Wird am 1.1. nach Mitternacht aus `Pellets_kg_lfd_Jahr` umkopiert.

Pellets_kg_total
Zahl: Verbrauch Pellets total in kg (fortlaufender Zähler)
Wertebereich: 0…65000 kg
verwendet in: Peleo-RES1 (Berechnung)
„Ewiger“ Zähler Pelletsverbrauch. Wird mit jedem RES1-Signal aktualisiert und läuft unabhängig von der übrigen Berechnung. Mit den Werten dieses Zählers aus einer Datenbank kann im Prinzip sehr einfach der Verbrauch zwischen beliebigen Zeitpunkten bestimmt werden. Achtung: Datenkompression verwenden!

Pellets_kg_Avg10
Real: durchschnittlicher Tagesverbrauch der letzten 10 Tage in kg
Wertebereich: 0…65000 kg
verwendet in: Peleo Tagesstatistik (Berechnung, Verwendung)
Diese Variable gibt den Durchschnittsverbrauch der letzten 10 Tage an. Der Wert wird mittels gleitendem Durchschnitt in der Tagestatistik um Mitternacht berechnet. Die dazu benötigten Verbräuche der letzten 9 Tage werden in der Zeichenketten-Variablen `Pellets_Day_Avg10_List` verwaltet. Der Durchschnittsverbrauchswert wird für die Lager-Reichweitenprognose verwendet.

Pellets_Day_Avg10_List
Zeichenkette: kommagetrennte Liste der letzten Tagesverbräuche für gleitenden Durchschnitt
Wertebereich: - / kg
verwendet in: Peleo Tagesstatistik (Verwendung, Aktualisierung)
Diese interne Variable hält in der Form einer Zeichenkette die Tages-Verbrauchswerte der letzten 9 Tage. Diese werden für die Berechnung des 10-Tages-Durchschnittsverbrauchs benötigt. Da die HomeMatic keine Variablen-Arrays kennt und die Verwendung von 9 Systemvariablen für diesen Zweck weder ressourcenschonend noch übersichtlich ist, behelfe ich mich hier mit einer Zeichenketten-Variablen, in der die Werte durch Kommas separiert abgelegt sind. Beispiel: `29.07,30.78,40.33,43.13,40.42,44.84,48.17,40.42,45.08` wobei der neueste Wert links steht. Zur Berechnung wird die Liste durch String-Operationen wieder in die einzelnen Werte zerlegt und auch entsprechend aktualisiert.

Pellets_Lager_t
Zahl: Füllstand Pelletslager in Tonnen
Wertebereich: -1…50 t
verwendet in: Peleo Tagesstatistik (Berechnung, Verwendung)
Der Wert gibt den aktuellen Füllstand des Pelletslagers in Tonnen an. Der Wert wird einmal täglich durch Abzug des jeweiligen Tagesverbrauchs `Pellets_kg_gestern` aktualisiert.

Pellets_Lagerfuellgrad
Zahl: Füllgrad Pelletslager in Prozent
Wertebereich: 0…100 %
verwendet in: Peleo Tagesstatistik (Berechnung)
Wird analog der Füllstands in Tonnen berechnet. Der Prozentwert wird für die grafische Darstellung als Balkendiagramm in der AIO Remote verwendet

Pellets_Lager_niedrig
Logikwert: wahr, wenn Lagerstand die Meldeschwelle unterschritten hat
Wertebereich: ist wahr / ist falsch
verwendet in: Peleo Tagesstatistik (Berechnung / Verwendung)
Zeigt an, daß der Lagerstand unter dem Schwellwert liegt. Bleibt aktiv, bis der Lagerstand den Schwellwert wieder erreicht oder übersteigt. Die Berechnung erfolgt durch Vergleich des Lagerfüllstands in Tonnen mit dem Wert der Meldeschwelle `Pellets_Lager_Schwelle`.

Pellets_Lagerreichweite
Zahl: Reichweiten-Prognose des Pellet-Lagers (Tage)
Wertebereich: 0…65000 d
verwendet in: Peleo Tagesstatistik (Berechnung)
Die Reichweitenprognose ist ein Schätzwert, wie viele Tage der aktuelle Lagerstand noch reicht, unter Voraussetzung des Durchschnittsverbrauchs der letzten 10 Tage. Berechnet aus Füllstand Lager und dem Durchschnittsverbrauch der letzten 10 Tage. Der Durchschnittsverbrauch über 10 Tage wurde gewählt um einerseits kurzfristige Tagesschwankungen zu glätten, andererseits sollte möglichst die aktuelle Verbrauchssituation berücksichtigt werden. Der Wert ist ja meistens dann interessant, wenn die Lagerreichweite schon niedrig ist und man sich überlegt, wann es Zeit wird, Pellets nachzubestellen. Dann macht der jahreszeitlich bedingte aktuelle Verbrauch mehr Sinn, als z.B. der Jahresdurchschnitt.

Pellets_Aschebox_kg
Zahl: errechnete Füllmenge der Aschebox in kg
Wertebereich: 0…65000 kg
verwendet in: Peleo Tagesstatistik (Berechnung, Verwendung)
Die Variable gibt den ungefähren Füllstand der Aschebox in kg an. Der Wert ist auch eher eine Schätzeisen-Berechnung aus der verbrannten Pelletmenge. Dazu wird die verbrauchte Tagesmenge Pellets mit der Konstanten `Pellets_Aschefaktor` multipliziert. Der Aschefaktor gibt das Verhältnis kg Pellets zu kg Asche an, also wieviel kg Asche nach dem Verbrennen von kg Pellets anfällt. Diesen Wert kann man z.B. auswiegen.

Pellets_Aschebox_Fuellgrad
Zahl: errechneter Füllgrad der Aschebox in Prozent
Wertebereich: 0…100 %
verwendet in: Peleo Tagesstatistik (Berechnung, Verwendung)
Im Grunde die gleiche Information wie `Pellets_Aschebox_kg`, nur über die Konstante `Pellets_Aschebox_max` als Füllgrad in Prozent umgerechnet. Wie beim Lagerstand wird dies zur Visualisierung als Balkendiagramm in der App genutzt. Außerdem wird der Wert für die Meldeschwelle verwendet, weil das Einstellen eines Schwellwerts für die Asche in Prozent wahrscheinlich intuitiver ist als ein kg-Wert.

Pellets_Aschebox_voll
Logikwert: wahr, wenn Aschestand über Schwellwert
Wertebereich: wahr / falsch
verwendet in: Peleo Tagesstatistik (Berechnung / Verwendung)
Zeigt an, daß der Aschestand über dem Schwellwert liegt. Bleibt aktiv, bis der Füllstand zurückgesetzt wird.

Pellets_RES1_kg_ratio
Zahl: Austragungsmenge pro Umlauf der Kugelschleuse (Kalibrierwert)
Wertebereich: 0…10 kg
verwendet in: Peleo-RA early (Berechnung) Peleo-RA geplant (Berechnung) Peleo-RES1 (Verwendung)
Im Variablendiagramm schön zu erkennen ist diese Variable sehr zentral für viele Berechnungen. Der Wert gibt an, wie viele kg Pellets bei einem Umlauf der Kugelschleuse (ein Impuls des Signals nRES1) zugeführt, also verbraucht werden. Dieser Wert muß möglichst genau bestimmt werden, da sich Ungenauigkeiten hier stark aufaddieren. Mögliche Methoden, diesen Wert zu bestimmen, sind im ersten Teil beschrieben worden. Eigentlich müßte diese Variable in die Kategorie „Konstante“ fallen. Da trotz der eher weniger befriedigenden Ergebnisse die Möglichkeit besteht, den Wert durch das Programm automatisch bestimmen zu lassen, ist sie hier unter Variable verbucht. Das Programm versucht auf Wunsch, diesen Parameter automatisch zu ermitteln (siehe **Pellets_Cal_Auto**). Das setzt voraus, daß bei Aufruf des Skripts *Peleo-RA early* der Zwischenbehälter vollständig geleert ist. Geschieht dies nicht (oder findet aufgrund des Verbrauchs keine vorzeitige Lagerentnahme statt), muß der Wert manuell ermittelt werden. Alternativ kann auch davon ausgegangen werden, daß typischerweise 10%-15% Pellets im Zwischenbehälter verbleiben, wenn eine vorzeitige Auffüllung ausgelöst wird und den Wert entsprechend korrigieren (z.B. in die Berechnung einen Faktor 0,87 einfügen). Für eine manuelle Ermittlung den Behälter möglichst weit entleeren lassen. Stand von **Pellets_RES1_Cal_counter** notieren. Pellets manuell in den Zwischenbehälter einfüllen und dabei abwiegen (z.B. mit Eimer und Kofferwaage). Das Verfahren wurde im ersten Teil genauer beschrieben. Hierbei auch unbedingt die Sicherheitshinweise beachten! Der Variablenwert kann wie folgt berechnet werden: *Pellets_RES1_kg_ratio* = eingefüllte kg / *Pellets_RES1_Cal_counter* Beispiel: *Pellets_RES1_kg_ratio* = 24,95 kg / 525 = 0,0475 kg Der Wert kann dann manuell gesetzt werden: dom.GetObject("Pellets_RES1_kg_ratio").State(0.0475); Hinweis: Bei manueller Ermittlung muß `Pellets_Cal_Auto` auf `falsch` gestellt werden, damit der Wert nicht überschrieben wird.

Pellets_RES1_Cal_counter
Zahl: Anzahl der Umläufe der Kugelschleuse seit der letzten Befüllung des Zw.-Behälters
Wertebereich: 0…65000
verwendet in: Peleo-RES1 (Aktualisierung, Verwendung) Peleo-RA geplant (Rücksetzen) Peleo-RA early (Verwendung, Rücksetzen)
Dieser Zähler läuft analog zu `Pellets_RES1_Counter`, wird aber anders zurückgesetzt. Er wird nach dem Befüllen des Zwischenbehälters wieder auf 0 gesetzt und zählt also die Anzahl der Entnahmen durch die Kugelschleuse seit der Zwischenbehälter komplett befüllt wurde. Der Wert wird für zwei Berechnungen verwendet: (a) Für die automatische Kalibrierung von `Pellets_RES1_kg_ratio` (b) Für die Bestimmung des Füllgrads des Zwischenbehälters in Prozent, was wiederum für die Bewertung der Lageraustragung verwendet wird.

Pellets_RES1_Cal
Zahl: Anzahl der Umläufe der Kugelschleuse, um Zw.-Behälter vollständig zu leeren (Kalibrierungswert)
Wertebereich: 0…65000
verwendet in: Peleo-RA early (Berechnung / Verwendung) Peleo-RA geplant (Berechnung / Verwendung)
Speichert den Maximalwert von `Pellets_RES1_Cal_counter` . Der Wert wird bei Beginn jeder Befüllung des Zwischenbehälters geprüft und ggf. erhöht. Er ist ein Zwischenwert für die Berechnung des `Pellets_RES1_kg_ratio`. Ein endgültiger Wert für diese Variable kann nur ermittelt werden, wenn der Zwischenbehälter ganz geleert wurde („*earlyRA*“).

Pellets_Fuellgrad_ZwBeh
Zahl: berechneter Füllstand des Zwischenbehälters in %
Wertebereich: 0…100 %
verwendet in: Peleo-RES1 (Berechnung) Peleo-RA early (Verwendung) Peleo-RA geplant (Verwendung)
Wird berechnet aus den Umläufen der Kugelschleuse seit der letzten Befüllung des Zwischenbehälters. Dazu wird die Anzahl der Umläufe der Kugelschleuse („Austragungseinheiten“) mit dem `Pellets_RES1_kg_ratio` multipliziert und von der maximalen Füllmenge abgezogen. Dieser Wert wird in Prozent umgerechnet. Wird verwendet für die Visualisierung des Füllstands in der App und zur Vorhersage der Lageraustragungsdauer.

Pellets_RA-Dauer_erwartet
Zahl: geschätzte Dauer der kommenden Raumaustragung in Minuten
Wertebereich: 0…200 min
verwendet in: Peleo-RA geplant (Berechnung) Peleo-RA early (Berechnung) Peleo-RA-beendet (Verwendung)
Wird aus dem Füllgrad des Zwischenbehälters zu Beginn der Lageraustragung errechnet. Der Wert wird am Ende mit der gemessenen, tatsächlich benötigten Zeit verglichen, um Störungen bei der Pelletsförderung erkennen zu können.

Pellets_RA-Dauer_letzte
Zahl: Dauer der letzten Raumaustragung in Minuten
Wertebereich: 0…200 min
verwendet in: Peleo-RA-beendet (kopieren aus CUxD-Datenpunkt)
Gemessene Zeitdauer der letzten Lageraustragung. Der Wert wird vom CUxD StateMon-Device automatisch gemessen. Nach Ende der Lageraustragung wird der Wert in eine Systemvariable kopiert, um ihn in der AIO Remote zugänglich zu machen (die Datenpunkte des CUxD-Device sind nicht direkt in AIO importierbar).

Pellets_RA-Dauer_Abweichung
Zahl: Abweichung Dauer der letzte Raumaustragung vom Schätzwert in %
Wertebereich: -100000…100000 %
verwendet in: Peleo-RA-beendet (Berechnung)
Nach Ende der Lageraustragung wird die tatsächlich benötigte Zeit mit der geschätzten Zeit ins Verhältnis gesetzt und als Prozentwert angegeben. Das Ergebnis kann positiv (hat länger gedauert) oder negativ (ging schneller als erwartet) sein. Der Wert wird auf +500% begrenzt, falls bei fehlgeschlagener Austragung sehr große Saugzeiten entstehen.

Pellets_RES1_Burner_Counter
Zahl: Zähler zur Ermittlung des Brennerstatus
Wertebereich: 0…65000
verwendet in: Peleo-RES1 (Erhöhung) Peleo_Brennerstatus_berechnen (Verwendung, Rücksetzen)
Zählt die nRES1-Impulse in einem vorgegebenen Zeitraum. Wird verwendet, um Brennerleistung aus der zugeführter Pelletmenge abzuschätzen

Pellets_Brennerstatus
Zahl: Brennerleistung in Prozent
Wertebereich: 0…100 %
verwendet in: Peleo_Brennerstatus_berechnen (Berechnung)
Berechnet aus Pelletszufuhr die mittlere Brennerleistung der letzten 15 min in Prozent. Das verwendete Verfahren stellt eine Tiefpaß-Glättung der nRES1-Impulse dar. Das Zeitintervall von 15 min wurde gewählt, da es sich bei der Verbrennung der Pellets um einen eher trägen Prozeß handelt. So brennt das Feuer im Brenner auch nach Ende der Pelletszufuhr noch etwa 15 bis 20 Minuten weiter (Kesselstatus „Nachlauf“). Hier würde sich zur Berechnung natürlich auch der gleitende Mittelwert anbieten. Aufgrund des Rechenaufwands und um Ressourcen der CCU zu schonen, wurde das nicht umgesetzt, sondern eine sequentielle, einfache Mittelwertberechnung alles 15 min. Die Berechnung beinhaltet eine Konstante zur Normierung auf 100%, die ggf. individuell angepaßt werden muß. Dazu bei 100% Leistung die Anzahl der RES1-Umläufe pro 15 min abzählen und als Konstante einsetzen.

Pellets_Standardsaugzeit
Logikwert: wahr, wenn aktuelle Uhrzeit im Zeitfenster Standard-Saugzeit
Wertebereich: wahr / falsch
verwendet in: Peleo Standardsaugzeit (Berechnung) Peleo-RA geplant (Prüfung) Peleo-RA early (Prüfung)
Last but least das eigentlich überflüssig gewordene Flag der Standardsaugzeit. Der Zustand wahr zeigt an, daß eine geplante Auffüllung des Zwischenbehälters ansteht. Dementsprechend wird beim Zustand *wahr* das Programm *Peleo-RA geplant* gestartet, wenn das Signal nRA aktiv wird. Beim Zustand *falsch* wird dagegen *Peleo-RA early* gestartet. Da die beiden Programme inzwischen praktisch dasselbe machen, ist die Umschaltmechanik eigentlich eher ein Relikt.

Pellets_maxPwr	ab Version: B2
Zahl: Maximalwert Brennerleistung für autom. Leistungskalibrierung	
Wertebereich: 0.0 … 200.0	
verwendet in: Peleo_Brennerstatus_berechnen (Berechnung) Peleo Tagesstatistik (Verwendung)	
Maximalwert der Brennerleistung für die Kalibrierung über die Brenner-Leistungsanzeige.	

Pellets_PwrAbw	ab Version: B2
Zahl: letzte gespeicherte Abweichung der Brennerleistung für autom. Leistungskalibrierung	
Wertebereich: 0.0 … 200.0	
verwendet in: Peleo Tagesstatistik (Berechnung)	
Abweichung der ermittelten max. Brennerleistung von 100% zur Kalibrierungskontrolle Beispiel:	

Pellets_Cal_dev	ab Version: B2
Logikwert: wahr, wenn mögliche Kalibrierungsabweichung ermittelt wurde	
Wertebereich: wahr, falsch	
verwendet in: Peleo Tagesstatistik (Berechnung) Peleo-RA-beendet (Berechnung) Peleo-RES1 (Berechnung)	
Wird gesetzt bei erkannter Abweichung einer der zu Kalibrierungskontrolle benutzten Parameter: - max. Brennerleistung weicht von 100% ab - Füllgrad Zwischenbehälter ist bei early-RA zu hoch (sollte leer sein) - Füllgrad Zwischenbehälter fällt zu stark unter 0% Dient zur Anzeige in der Bedienoberfläche. Das Flag wird nicht automatisch zurückgesetzt.	

Übersicht über alle Konstanten und Variablen

Pellets_Asche_Schwelle	Meldeschwelle Füllgrad Aschebox	Zahl	Minimalwert: 0 Maximalwert: 100	%
Pellets_Aschebox_Fuellgrad	Füllgrad Aschebox in %	Zahl	Minimalwert: 0 Maximalwert: 100	%
Pellets_Aschebox_kg	errechneter Füllstand Aschebox	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_Aschebox_max	Fassungsvermögen Aschebox kg	Zahl	Minimalwert: 0 Maximalwert: 255	kg
Pellets_Aschebox_voll	Statusmeldung Aschebox	Logikwert	wahr = wahr falsch = falsch	
Pellets_Aschefaktor	Verhältnis kg Asche : kg Pellets	Zahl	Minimalwert: 0 Maximalwert: 65000	
Pellets_Brenner_Cal	Kalibrierungswert für Brennerleistung	Zahl	Minimalwert: 0 Maximalwert: 100	
Pellets_Brennerstatus	ermittelte Brennerleistung aus Pelletszufuhr	Zahl	Minimalwert: 0 Maximalwert: 100	%
Pellets_Cal_Auto	Freischaltung autom. RES1-Kalibrierung	Logikwert	wahr = ein falsch = aus	
Pellets_Cal_dev	Warnung Kalibrierungsabweichung	Logikwert	wahr = wahr falsch = falsch	
Pellets_Cal_kg_ZwBeh	max. Füllmenge kg Zwischenbehälter	Zahl	Minimalwert: 0 Maximalwert: 100	kg
Pellets_Day_Avg10_List	Werteliste letzte 10 Tagesverbräuche	Zeichenkette		kg
Pellets_Fuellgrad_ZwBeh	Füllstand Zwischenbehälter %	Zahl	Minimalwert: 0 Maximalwert: 100	%
Pellets_kg_Avg10	durchschn. Pelletverbrauch letzte 10 d	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_kg_gestern	Pelletverbrauch kg gestern	Zahl	Minimalwert: 0 Maximalwert: 5000	kg
Pellets_kg_heute	Pelletsverbrauch heute aus RES1	Zahl	Minimalwert: 0 Maximalwert: 5000	kg
Pellets_kg_letzter_Monat	Pelletverbrauch letzter Monat	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_kg_letztes_Jahr	Pelletverbrauch letztes Jahr	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_kg_lfd_Jahr	Pelletverbrauch laufendes Jahr	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_kg_lfd_Monat	Pelletverbrauch kg laufender Monat	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_kg_total	Pelletverbrauch Summe kg	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_kWh_gestern	Pelletverbrauch in kWh	Zahl	Minimalwert: 0 Maximalwert: 65000	kWh
Pellets_Lager_niedrig	Lagerstand Pellets hat Schwellwert unterschritten	Logikwert	wahr = ist wahr falsch = ist falsch	
Pellets_Lager_Schwelle	Schwellwert Lagerstand niedrig in kg	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_Lager_t	Füllmenge Lager t	Zahl	Minimalwert: -1 Maximalwert: 50	t
Pellets_Lageraenderung	Vorgabewer Lagerzu- / abgang	Zahl	Minimalwert: 0 Maximalwert: 65000	kg
Pellets_Lagerfuellgrad	Lagerfüllung in %	Zahl	Minimalwert: 0 Maximalwert: 100	%
Pellets_Lagermax_t	max. Lagermenge t	Zahl	Minimalwert: 0 Maximalwert: 50	t
Pellets_Lagerreichweite	Reichweite auf 10d-Mittel	Zahl	Minimalwert: 0 Maximalwert: 65000	d
Pellets_maxPwr	maximale berechnete Brennerleistung	Zahl	Minimalwert: 0 Maximalwert: 200	%
Pellets_PwrAbw	Abweichung max Leistung Kalibrierungskontrolle	Zahl	Minimalwert: -1000 Maximalwert: 1000	%

Pellets_RA-Dauer_Abweichung	Abweichung der Saugdauer	Zahl	Minimalwert: -100000 Maximalwert: 100000	%
Pellets_RA-Dauer_erwartet	Schätzung der kommenden RA-Dauer aus Füllstand	Zahl	Minimalwert: 0 Maximalwert: 200	min
Pellets_RA-Dauer_letzte	Dauer der letzten Raumaustragung in min	Zahl	Minimalwert: 0 Maximalwert: 200	min
Pellets_RES1_Burner_Counter	Zähler für Brennerstatus	Zahl	Minimalwert: 0 Maximalwert: 65000	
Pellets_RES1_Cal	Anzahl RES1 pro Behälter	Zahl	Minimalwert: 0 Maximalwert: 65000	
Pellets_RES1_Cal_counter	Zählvariable für Kalibrierung Austragung	Zahl	Minimalwert: 0 Maximalwert: 65000	
Pellets_RES1_Counter	Zähler Zwischenbehälter täglich	Zahl	Minimalwert: 0 Maximalwert: 65000	
Pellets_RES1_gestern	Zähler Zwischenbehälter gestern	Zahl	Minimalwert: 0 Maximalwert: 65000	
Pellets_RES1_kg_ratio	Kalibrierwert kg pro RES1-Umlauf	Zahl	Minimalwert: 0 Maximalwert: 10	kg
Pellets_Standardsaugzeit	wahr wenn Standard-Saugzeit	Logikwert	wahr = wahr falsch = falsch	

Die Programme

Die folgende Tabelle gibt einen Überblick über die verwendeten Programme:

Name	Beschreibung	Bedingung	Aktivität	bedienbar
Peleo Standardsaugzeit	setzt Variable Saugzeitfenster	Zeit: Täglich von 12:00 Uhr bis 12:30 Uhr beginnend am 08.12.2020 zu Zeitpunkten auslösen	Systemzustand: Pellets_Standardsaug-zeit sofort auf wahr setzen	
Peleo Tagesstatistik	Berechnung Tagesstatistik Pellets	Zeit: Täglich um 00:02 Uhr beginnend am 10.12.2020 zu Zeitpunkten auslösen	Skript: ... sofort ausführen	
Peleo-RA early	RA Zwischenbehälter ganz leer	Kanalzustand: Peleo-nRA bei Schaltzustand: aus bei Änderung auslösen	Skript: ... sofort ausführen	
Peleo-RA geplant	RA im Saugzeitfenster	Kanalzustand: Peleo-nRA bei Schaltzustand: aus bei Änderung auslösen	Skript: ... sofort ausführen	
Peleo-RA-beendet	RA beendet	Kanalzustand: Peleo_RA_StateMon:1 bei Schaltzustand: aus bei Änderung auslösen	Skript: ... sofort ausführen	
Peleo-RES1	Zähler Zwischenbehälter-Austragung	Kanalzustand: Peleo-nRES1 bei Schaltzustand: aus bei Änderung auslösen	Skript: ... sofort ausführen	
Peleo-Störung	Störmeldung Pelletskessel	Kanalzustand: Peleo-nStoerung bei Schaltzustand: aus bei Änderung auslösen	Skript: ... sofort ausführen	
Peleo_Brennerstatus_berechnen	aktuelle Brennerleistung schätzen	Zeit: Periodisch Ganztägig beginnend am 28.12.2020 zu Zeitpunkten auslösen	Skript: ... sofort ausführen	
Peleo_Lager_abbuchen	Pellets_Lageraenderung kg dem Lager abbuchen	Systemzustand: CCU_Bootstate bei nur prüfen CCU_up	Skript: ... sofort ausführen	X
Peleo_Lager_zubuchen	Pellets_Lageraenderung kg dem Lager zubuchen	Systemzustand: CCU_Bootstate bei nur prüfen CCU_up	Skript: ... sofort ausführen	X
Peleo-Aschestand Reset	Pellets Zähler Aschebox zurücksetzen	Systemzustand: CCU_Bootstate bei nur prüfen CCU_up	Skript: ... sofort ausführen	X

Man sieht, daß praktisch alle Programme auf der Ausführung von Skripten basieren.
Die letzten drei Programme sind für den Anwender bedienbar, bei allen anderen Programmen ist das Häkchen bei „bedienbar“ entfernt. Ob man die Programme sichtbar läßt oder nicht ist, wie ich finde, Geschmackssache, wieviel „Technik“ sichtbar bleiben soll.

In der folgenden Beschreibung der Programme ist die Reihenfolge etwas anders, damit die Zusammenhänge besser aufeinander aufbauen.
Die Skripte verarbeiten nur Systemvariablen und greifen nicht direkt auf Geräte zu.
Ausnahme: Das Umkopieren der Austragungszeit aus dem CUxD-StateMon in eine Systemvariable im Skript *Peleo-RA-beendet*.
Bei der Benennung der lokalen Variablen innerhalb eines Skripts war ich leider nicht von Anfang an konsequent genug, ein festes System zu verwenden. Dadurch haben die Variablennamen der Skripte eine gewisse Evolution erfahren. Irgendwann hat sich die Konvention herausgebildet, daß eine lokale Variable, die eine Systemvariable übernimmt, den gleichen Namen ohne führendes „*Pellets_*“ bekommt. Zur Unterscheidung sind lokale Variablen (oft) in Kleinbuchstaben geschrieben.
(Vorzeige-) Beispiel:
```
ra_dauer_erwartet = dom.GetObject("Pellets_RA-Dauer_erwartet").Value();
```

Außer den erwähnten Datenpunkten der Geräte, die hauptsächlich in den Programmen vorkommen und natürlich den Systemvariablen, ist das System softwareseitig ziemlich unabhängig.
Folgende Systemvoraussetzungen der CCU werden von den Programmen und Skripten benutzt:

- **System-Anwesenheit**
  In den Bedingungsprüfungen der Programme wird häufig der Systemzustand
  „Wenn Systemzustand CCU_Bootstate *bei* CCU_up nur prüfen“
  abgefragt.
  CCU_Bootstate ist dabei die berühmte CCU – Anwesenheitsvariable, mit der die Ausführung von Programmen beim Booten der CCU verhindert werden kann. Der Zustand *CCU_up* bedeutet, der Bootvorgang ist abgeschlossen. Hier gibt es natürlich unterschiedliche Bezeichnungen für die Variable.

- **CUxD**
  Die Verwendung des CUx-Daemon wurde schon weiter oben ausführlich beschrieben.

- **Signalgeber „MP3 Funkgong mit LED“ HM-OU-CFM-TW**
  Dieses Ding benutze ich für Ansagen und Mittelungen die nicht nur aufs Handy geschickt, sondern gleich im Haus angesagt werden sollen. In diesem Fall wird die Störmeldung des Pelletskessels mit einer entsprechenden Ansage gemeldet.

- **Pushover**
  Für den Versand von Statusmitteilungen, z.B. bei einer Kessel-Störungsmeldung, verwende ich Pushover. Hier kann natürlich jeder andere Nachrichtendienst eingesetzt werden. Die Aufrufe sind in den Skripten mit […] gekennzeichnet.

- **Email-Addon**
  Die Skripte benutzen das Email-Addon zum Versenden von „stillen“ Informationen. Das sind vor allem Benachrichtigungen, die nachts während der Tagesstatistik erzeugt werden.
  In meiner Konfiguration werden diese Email-IDs aufgerufen:

ID	Betreff
03	HomeMatic Zentrale: Pellet-Lagerstand niedrig
04	HomeMatic Zentrale: Peleo Aschebox voll
05	HomeMatic Zentrale: Kalibrierungskontrolle

Der Nachrichtenkörper ist jeweils:

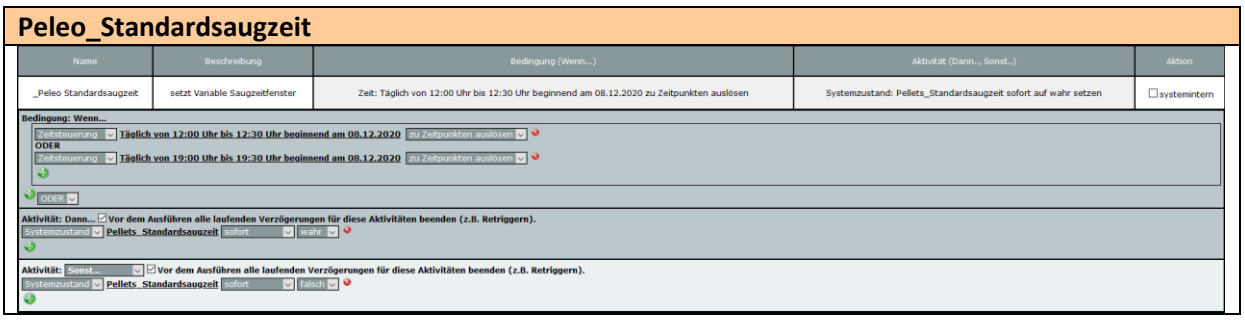
```
----- HomeMatic-Statusmeldung -----

$status
```

Nun aber zu den Programmen. In den Screenshots ist teilweise vor dem Programmnamen ein Unterstrich _ zu sehen. Der diente während der Programmierung dazu, daß die Programme in der Programmtabelle alle am Anfang stehen, also Faulheit beim Scrollen.
Hier muß man nur später aufpassen, die Programme umzubenennen, bevor sie im AIO Neo importiert werden. Das betrifft nur die Programme, die bedient werden.
Vielleicht fällt auch auf, daß die Skripte in den Screenshots anders beginnen, als im Listing darunter. Das liegt ganz einfach daran, daß die Skripte nach den Screenshots teilweise noch weiterentwickelt und auch etwas aufgehübscht wurden. Die Formatierung der Skripte ist mit Notepad++ erfolgt.

Peleo_Störung

Als erstes zur Einstimmung das Programm für die Kessel-Störmeldung – ja, das kommt auch schon nicht ohne Skript aus. Das Skript dient hier lediglich dazu, die Pushnachricht zu senden.
Zu beachten ist hier die invertierte Logik des Signals *Peleo-nStörung*, weswegen das Programm bei *Schaltzustand: aus* ausgelöst wird.

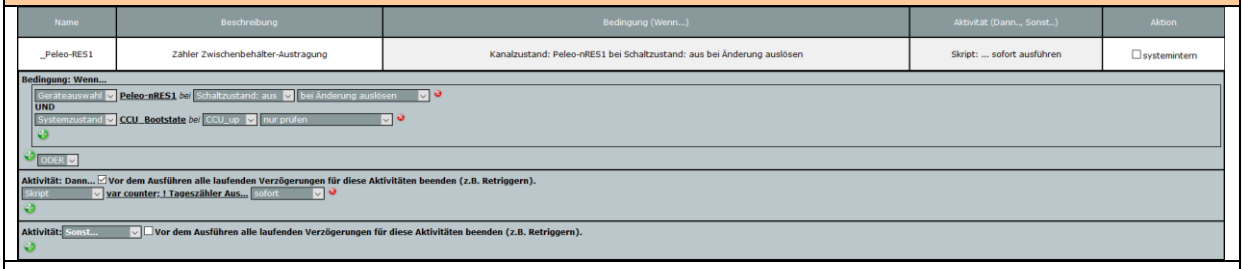
Peleo_Standardsaugzeit



Hurra, ein Programm ohne Skript! (Dafür das einzige mit einem *Sonst*-Zweig.) Hier werden die Zeitfenster definiert, die als Standard-Saugzeiten am Pelletskessel eingestellt sind.
Im Programm oben sind das zwei Zeitpunkte, um 12:15 Uhr und um 19:15 Uhr. Die Zeitfenster werden mit +/- 15 min Toleranz angegeben.

Das Programm macht nichts weiter, als den Wert der Variablen **`Pellets_Standardsaugzeit`** entsprechend einzustellen.
So lassen sich die Zeiten flexibel konfigurieren, ohne in den Skripten oder in Systemvariablen Werte ändern zu müssen.

Peleo-RES1




```
!- --- Skript Peleo-RES1 ---
!- bei Impuls Kugelschleuse (nRES1)

var counter;                !- Tageszähler Austragungen
var cal_counter;            !- Zähler zur Kalibrierung Austragung
var burner_counter;         !- Zähler für Brenner-Aktivität
var fuellgrad_zwischenbeh;  !- Füllstand Zwischenbehälter in Prozent
var fuellstand_zw_max;      !- Maximalwert Zwischenbehälter in kg
var pellets_RES1_kg_ratio;  !- Kalibrierwert Pelletsmenge pro RES1-Umlauf
var pellets_kg_heute;       !- Tagesverbrauch Pellets in kg
var pellets_kg_total;       !- Gesamtverbrauch Pellets in kg
var pellets_kg_monat;       !- Verbrauch Pellets in kg laufender Monat
var pellets_kg_jahr;        !- Verbrauch Pellets in kg laufendes Jahr
var pellets_cal_dev;        !- Flag für Kalibrierungsabweichung

!- Variablen einlesen
pellets_RES1_kg_ratio = dom.GetObject("Pellets_RES1_kg_ratio").Value();
pellets_cal_dev = dom.GetObject("Pellets_Cal_dev").Value();

!- Tageszähler erhöhen
counter = dom.GetObject("Pellets_RES1_Counter").Value();
counter = counter+1;
dom.GetObject("Pellets_RES1_Counter").State(counter);

!- Kalibrierungszähler erhöhen
cal_counter = dom.GetObject("Pellets_RES1_Cal_counter").Value();
cal_counter = cal_counter+1;
dom.GetObject("Pellets_RES1_Cal_counter").State(cal_counter);

!- Brennerzähler erhöhen
burner_counter = dom.GetObject("Pellets_RES1_Burner_Counter").Value();
burner_counter = burner_counter+1;
dom.GetObject("Pellets_RES1_Burner_Counter").State(burner_counter);

!- Füllstand Zwischenbehälter berechnen
fuellstand_zw_max = dom.GetObject("Pellets_Cal_kg_ZwBeh").Value();
fuellgrad_zwischenbeh = (fuellstand_zw_max - (cal_counter * pellets_RES1_kg_ratio)) / fuellstand_zw_max;

!- Kalibrierungskontrolle
if ((fuellgrad_zwischenbeh < -0.05) && (!pellets_cal_dev))
{ !- der Füllgrad kann nicht negativ werden (m. 5% Toleranz). Nur einmal benachrichtigen.
 string MailText;
 string sDate = system.Date("%d.%m.%Y"); !- per Mail benachrichtigen
 MailText = "Kalibrierungskontrolle Pelletssystem vom " # sDate # ".\r\n";
 MailText = MailText # "\r\n Abweichung Zwischenbehaelter-Inhalt: ";
 MailText = MailText # (fuellgrad_zwischenbeh * 100).ToString(1);
 MailText = MailText # " %).(Sollwert 0...-5%)\r\n";
 MailText = MailText # "\r\n Bitte Kalibrierung kontrollieren";
 string stdout;
 string stderr;
 system.Exec("/etc/config/addons/email/email 05 '"+MailText+"'",&stdout,&stderr);
 dom.GetObject("Pellets_Cal_dev").State(1); !- Statusflag für Bedienoberfläche setzen
}
```

```
!- Ergebnis für Anzeige aufs Intervall beschneiden
if (fuellgrad_zwischenbeh > 1)
{fuellgrad_zwischenbeh= 1;}
else
{if (fuellgrad_zwischenbeh < 0)
 {fuellgrad_zwischenbeh = 0;}
}
fuellgrad_zwischenbeh = fuellgrad_zwischenbeh * 100;
dom.GetObject("Pellets_Fuellgrad_ZwBeh").State(fuellgrad_zwischenbeh);

!- Tagesverbrauch Pellets in kg berechnen
pellets_kg_heute = counter * pellets_RES1_kg_ratio;
dom.GetObject("Pellets_kg_heute").State(pellets_kg_heute);

!- Gesamtverbrauch Pellets in kg berechnen
pellets_kg_total = dom.GetObject("Pellets_kg_total").Value();
pellets_kg_total = pellets_kg_total + pellets_RES1_kg_ratio;
dom.GetObject("Pellets_kg_total").State(pellets_kg_total);

!- Monatsverbrauch Pellets in kg aktualisieren
pellets_kg_monat = dom.GetObject("Pellets_kg_lfd_Monat").Value();
pellets_kg_monat = pellets_kg_monat + pellets_RES1_kg_ratio;
dom.GetObject("Pellets_kg_lfd_Monat").State(pellets_kg_monat);

!- Jahresverbrauch Pellets in kg aktualisieren
pellets_kg_jahr = dom.GetObject("Pellets_kg_lfd_Jahr").Value();
pellets_kg_jahr = pellets_kg_jahr + pellets_RES1_kg_ratio;
dom.GetObject("Pellets_kg_lfd_Jahr").State(pellets_kg_jahr);
```

Das Programm ***Peleo-RES1*** wird bei jedem Umlauf der Kugelschleuse aufgerufen, also wenn das Signal *Peleo-nRES1* auf aktiv (=aus) wechselt.
Die Hauptaufgabe ist, die diversen Verbrauchszähler zu erhöhen.
Bei den Kurzzeit-Zählern, die direkt in diverse Tagesberechnungen eingehen, wird jeweils eine Zählvariable für die Anzahl der Umläufe inkrementiert und an anderer Stelle bei der Verwendung in die Pelletmenge (z.B. in kg) umgerechnet. Eine Änderung der Umrechnungsfaktoren hat so auch rückwirkend Einfluß auf die Werte dieses Zählers und ist in den Ausgaben praktisch direkt zu sehen.
Bei langfristigen Zählern (z.B. Monatsverbrauch) ist das anders gelöst, hier wird jeweils die aktuelle Menge Pelltes aufaddiert. Umrechnungsfaktoren, die in der Vergangenheit gegolten haben, werden damit nicht rückwirkend verändert.
Des weiteren wird in diesem Skript der Füllgrad des Zwischenbehälters in Prozent jeweils neu berechnet.
Hier ist auch eine Kalibrierungskontrolle eingebaut: Wenn alle Parameter passen, sollte der Füllgrad nicht nennenswert unter 0% fallen. Geschieht dies doch, kann das daran liegen, daß der Wert **`Pellets_RES1_kg_ratio`** zu groß ist (es werden tatsächlich weniger Pellets pro Umlauf entnommen, als eingestellt).
Zur Anzeige wird der Wert des Füllgrads auf das Intervall 0…100% beschnitten, damit es in der Anzeige nicht unschön aussieht.

Peleo-RA early

```
!- --- Skript Peleo-RA early ---
!- Kalibrierung Anzahl Umläufe Zwischenbehälter-Motor um Zwischenbehälter ganz zu leeren

var cal;               !- Kalibrierungsvariable Umläufe für ganzen Zwischenbehälter-Inhalt
var counter_cal;       !- Zähler für Umläufe zur Kalibrierung
real pellets_kg_ratio; !- kg-Umrechnung aus Anzahl Umläufe und kg-Fassungsverm. Zwischenbeh.
real cal_kg_zwbeh; !- Konstante zur Umrechnung Umläufe in kg (kg-Fassungsverm. Zwischenbeh.)
real fuellgrad_zwbeh;  !- Füllgrad Zwischenbehältzer zu Beginn der RA
real ra_dauer_estim;   !- erwartete Dauer der RA aus Füllgrad Zwischenbehälter

!- Variablen einlesen
cal = dom.GetObject("Pellets_RES1_Cal").Value();
counter_cal = dom.GetObject("Pellets_RES1_Cal_counter").Value();
cal_kg_zwbeh = dom.GetObject("Pellets_Cal_kg_ZwBeh").Value();
fuellgrad_zwbeh = dom.GetObject("Pellets_Fuellgrad_ZwBeh").Value();

!- wenn Auto-Cal freigegeben: RES1-Umlaufvolumen kalibrieren
!- wenn weniger als 400 Umläufe ist etwas schiefgegangen (Problem bei Saugförderer etc.)
if (dom.GetObject("Pellets_Cal_Auto").Value())
{
 if (counter_cal > 400)
 { cal = counter_cal; !- sonst den Zähler als neuen Kalibrierungswert setzen
   pellets_kg_ratio = cal_kg_zwbeh / cal; !- Füllmenge Zwischenbehälter (float!) / Anz.
Umläufe um diesen zu leeren
   dom.GetObject("Pellets_RES1_kg_ratio").State(pellets_kg_ratio);
 }
}

counter_cal = 0; !- Kalibrierungszähler rücksetzen

!- Schätzung vorraussichtliche RA-Dauer berechnen
ra_dauer_estim = (-0.0408*fuellgrad_zwbeh)+4.0278; !- *** Formel individuell anpassen ***
if (ra_dauer_estim < 0)
{ ra_dauer_estim =0; } !- falls Kurvenschnitt x-Achse ungenau

!- Variablen zurückschreiben
dom.GetObject("Pellets_RES1_Cal").State(cal);
dom.GetObject("Pellets_RES1_Cal_counter").State(counter_cal);
dom.GetObject("Pellets_RA-Dauer_erwartet").State(ra_dauer_estim);
```

Peleo-RA early wird aufgerufen, wenn eine Lageraustragung (angezeigt durch das Signal *Peleo_nRA* aktiv low) außerhalb des Standardsaugzeitraums erfolgt. Das ist eigentlich ein Ausnahmefall, der im Normalbetrieb selten vorkommen sollte.
Im Prinzip bedeutet diese Situation, daß so viele Pellets verbraucht worden sind, daß die Vorratsmenge im Zwischenbehälter nicht bis zum nächsten programmierten Auffüllzeitpunkt ausgereicht hat. Der Kessel erkennt dies daran, daß der untere Induktivgeber nach Ablauf eines Kugelschleusenumlaufs immer noch keine Pellets an der Brennerschnecke meldet.
Diese Situation ist deshalb von Vorteil, weil man hier (halbwegs genau) weiß, wieviel kg Pellets seit dem letzten Auffüllen des Zwischenbehälters entnommen wurden (nämlich das gesamte Behältervolumen).
Der Zähler **`Pellets_RES1_Cal_counter`** hält die Anzahl der Schleusenumläufe seit dem letzten Auffüllen (er wird in ***Peleo-RES1*** erhöht und in ***Peleo-RA early*** sowie ***Peleo-RA geplant*** zurückgesetzt). Da die ausgetragene Menge bekannt ist (z.B. 32 kg Behältervolumen), kann die Menge pro Schleusenumlauf leicht berechnet werden:

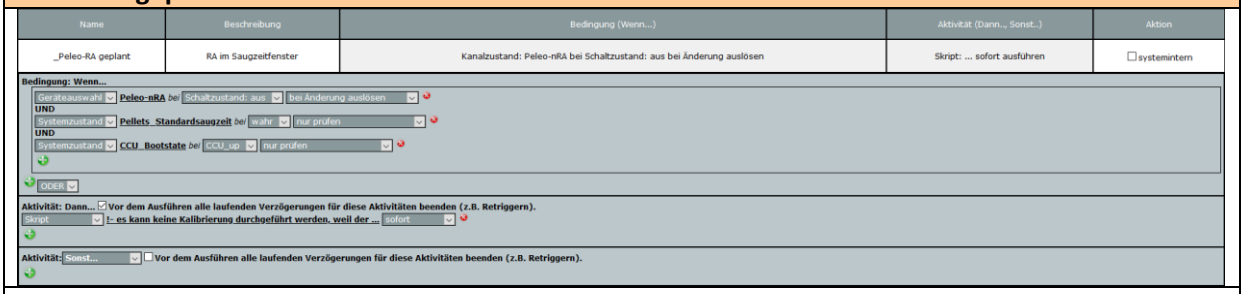
```
pellets_kg_ratio = cal_kg_zwbeh / cal;
```

In der Praxis ist es leider so, daß die Lageraustragung schon angeworfen wird, bevor der Behälter vollständig geleert ist. Vermutlich ist der Druck der verbleibenden Pellets auf die Förderschnecke nicht mehr ausreichend, um diese in die Umlaufschleuse zu fördern. So können z.B. 10% … 15%

Restmenge verbleiben, was das Ergebnis entsprechend verfälscht, wenn man es nicht von Hand korrigiert.
Der zur Kalibrierung gehörende Zählerstand wird außerdem in der Variable **`Pellets_RES1_Cal`** gespeichert, während der eigentliche Zähler zurückgesetzt wird, da der Behälter nun wieder befüllt wird.

Des weiteren wird für die Kontrolle der gerade beginnenden Lageraustragung die erwartete Förderdauer aus dem aktuellen Restfüllgrad berechnet.
Dazu wird eine individuell auf das Fördersystem angepaßte Geradengleichung verwendet, deren Bestimmung im ersten Teil beschrieben wurde.

Peleo-RA geplant

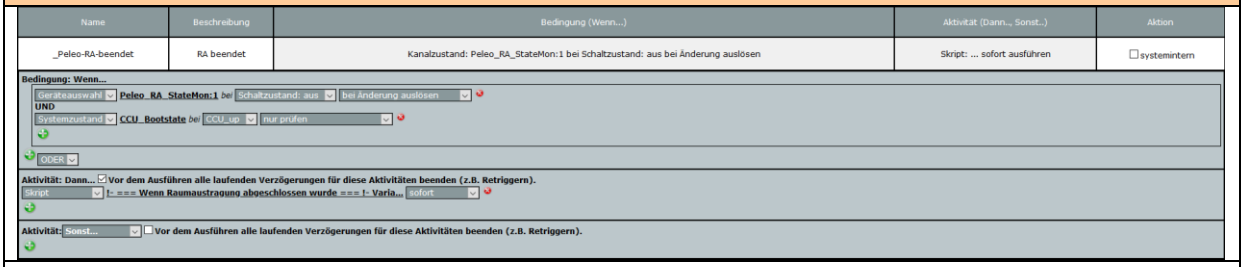



```
!- --- Skript Peleo-RA geplant ---
!- es kann keine Kalibrierung durchgeführt werden, weil der Behälter nicht leer ist
!- nur den Kalibrierungszähler rücksetzen für nächsten Turnus
!-
!- Falls jedoch mehr Umläufe als bisheriger Kalibrierwert und der Behälter ist trotzdem noch
nicht geleert,
!- dann war der alte Kalibrierwert wohl zu niedrig und kann zumindest auf den aktuellen Wert
erhöht werden
!- (sukzessive Annährung an realen Wert, der nur bei RA_early bestimmt werden kann.)

var cal;                !- Kalibrierungsvariable Umläufe für ganzen Zwischenbehälter-Inhalt
var counter_cal;        !- Zähler für Umläufe zur Kalibrierung
real pellets_kg_ratio; !- kg-Umrechnung aus Anzahl Umläufe und kg-Fassungsverm. Zwischenbeh.
real cal_kg_zwbeh;   !- Konstante zur Umrechnung Umläufe in kg (kg-Fassungsverm. Zwischenbeh.)
real fuellgrad_zwbeh;   !- Füllgrad Zwischenbehälter zu Beginn der RA
real ra_dauer_estim;    !- erwartete Dauer der RA aus Füllgrad Zwischenbehälter

!- Variablen einlesen
cal = dom.GetObject("Pellets_RES1_Cal").Value();
counter_cal = dom.GetObject("Pellets_RES1_Cal_counter").Value();
cal_kg_zwbeh = dom.GetObject("Pellets_Cal_kg_ZwBeh").Value();
fuellgrad_zwbeh = dom.GetObject("Pellets_Fuellgrad_ZwBeh").Value();

!- wenn Auto-Cal aktiv ist, ggf. Kalibrierung nachführen:
!- falls mehr Umläufe als bisheriger Kalibrierwert, dann war dieser wohl zu niedrig
!- Vorsicht Sonderfälle? (Brennerschnecke leer etc.)
if (dom.GetObject("Pellets_Cal_Auto").Value())
{
 if (counter_cal > cal)
 { cal = counter_cal; !- dann den Kalibrierungswert auf den aktuellen Wert erhöhen
   pellets_kg_ratio = cal_kg_zwbeh / cal; !- Füllmenge Zwischenbehälter (float!) / Anz.
Umläufe um diesen zu leeren
   dom.GetObject("Pellets_RES1_kg_ratio").State(pellets_kg_ratio);
 }
}

counter_cal = 0; !- Kalibrierungszähler rücksetzen

!- Schätzung vorraussichtliche RA-Dauer berechnen
ra_dauer_estim = (-0.0408*fuellgrad_zwbeh)+4.0278; !- === Formel individuell anpassen ===
if (ra_dauer_estim < 0)
{ ra_dauer_estim =0; } !- falls Kurvenschnitt x-Achse ungenau
```

```
!- Variablen zurückschreiben
dom.GetObject("Pellets_RES1_Cal").State(cal);
dom.GetObject("Pellets_RES1_Cal_counter").State(counter_cal);
dom.GetObject("Pellets_RA-Dauer_erwartet").State(ra_dauer_estim);
```

Peleo-RA geplant wird aufgerufen, wenn eine Lageraustragung (angezeigt durch das Signal *Peleo_nRA* aktiv low) während des Standardsaugzeitraums erfolgt.
Das Skript ist weitgehend identisch zu ***Peleo-RA early*** und unterscheidet sich nur in der Kalibrierungsoption.
Falls die automatische Kalibrierung aktiviert ist, versucht das Skript eine bestmögliche Anpassung. Da bei einer geplanten Raumaustragung der Zwischenbehälter in der Regel nicht ganz geleert wurde, ist die verbrauchte Pelletmenge unbekannt und es kann keine vollständige Kalibrierung erfolgen. Falls noch keine vollständige Kalibrierung über ***Peleo-RA early*** erfolgt ist, kann man dennoch davon ausgehen, daß die bislang *maximale* Anzahl von Schleusenumläufen die bestmögliche Annäherung darstellt. Die Routine vergleicht daher, ob die aktuell aufgezeichnete Anzahl von Umläufen **`Pellets_RES1_Cal_counter`** größer war als der bislang bekannte Wert **`Pellets_RES1_Cal`** und versucht damit, die Schätzung besser anzunähern.

Peleo-RA-beendet




```
!- --- Skript Peleo-RA-beendet ---
!- wenn Raumaustragung abgeschlossen wurde

!- Variablen definieren
real fuellgrad_zwischenbeh;   !- Füllstand Zwischenbehälter in Prozent
real ra_dauer_erwartet;       !- errechnete Austragungsdauer aus Füllgrad
var pellets_standardsaugzeit; !- wahr wenn Raumentnahme in programmierter Zeitspanne
var ra_dauer_abweichung;      !- Abweichung von erwarteter Austragungsdauer in Prozent

!- Laufzeit Raumaustragungsmotor für AIO Remote umkopieren in Sytemvariable
var ra_time_on = (datapoints.Get("CUxD.CUX9001001:1.TIME_ON")).Value();
dom.GetObject("Pellets_RA-Dauer_letzte").State(ra_time_on);

!- Variablen einlesen
fuellgrad_zwischenbeh = dom.GetObject("Pellets_Fuellgrad_ZwBeh").Value();
ra_dauer_erwartet = dom.GetObject("Pellets_RA-Dauer_erwartet").Value();
pellets_standardsaugzeit = dom.GetObject("Pellets_Standardsaugzeit").Value();

!- Kalibrierung verifizieren
if (!pellets_standardsaugzeit)
{                                  !- bei einer early-RA...
 if (fuellgrad_zwischenbeh > 20.0)
 {                                  !- ...sollte der Reststand im Behälter max. 15% sein
  string MailText;
  string sDate = system.Date("%d.%m.%Y"); !- per Mail benachrichtigen
  MailText = "Kalibrierungskontrolle Pelletssystem vom " # sDate # ".\r\n";
  MailText = MailText # "\r\n Abweichung Zwischenbehaelter-Inhalt bei early RA: ";
  MailText = MailText # fuellgrad_zwischenbeh.ToString(1);
  MailText = MailText # " %).(Sollwert 0..15%)\r\n";
  MailText = MailText # "\r\n Bitte Kalibrierung kontrollieren";
  string stdout;
  string stderr;
  system.Exec("/etc/config/addons/email/email_05 '"+MailText+"'",&stdout,&stderr);
```

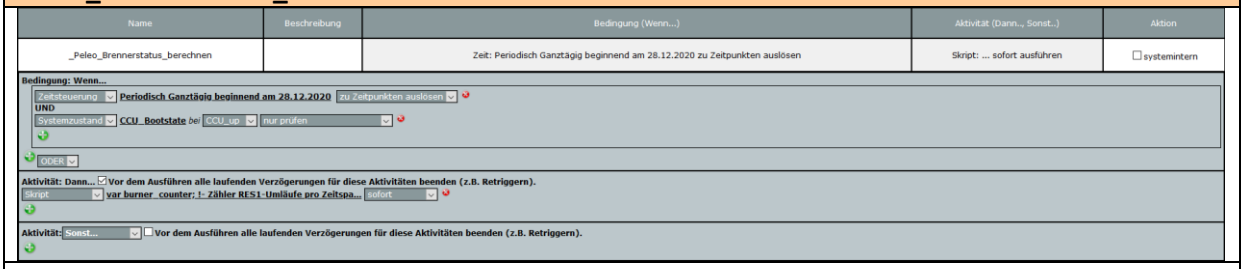
```
   dom.GetObject("Pellets_Cal_dev").State(1); !- Statusflag für Bedienoberfläche setzen
  }
}

!- Füllgrad auf 100% setzen
fuellgrad_zwischenbeh =100;
dom.GetObject("Pellets_Fuellgrad_ZwBeh").State(fuellgrad_zwischenbeh);

!- === Abweichung Raumaustragungsdauer berechnen ===
if (ra_dauer_erwartet == 0)
{ra_dauer_erwartet = 0.0167;} !- falls 0 dann 1 sek. um DIV!0 zu vermeiden
ra_dauer_abweichung = (ra_time_on - ra_dauer_erwartet) / ra_dauer_erwartet;
ra_dauer_abweichung = ra_dauer_abweichung *100; !- Formatierung in Prozent
if (ra_dauer_abweichung > 500)
{ra_dauer_abweichung = 500;} !- Begrenzung bei übermäßiger Abw. um Diagramm nicht zu versauen

!- Variablen schreiben
dom.GetObject("Pellets_RA-Dauer_Abweichung").State(ra_dauer_abweichung);

!- === Pushnachricht bei auffälliger Saugdauer ===
if (ra_dauer_abweichung > 100)
{
 string nachricht = "Hinweis: Peleo hohe Saugdauer-Abweichung";
 !- […] !— Pushnachricht senden
}
```

Nach Beendigung der Lageraustragung wird ***Peleo-RA-beendet*** aufgerufen.
Das Triggerereignis für dieses Programm ist der vom CUxD – State-Monitor ermittelte Schaltzustand. Da die logische Invertierung des Signals schon im StateMonitor definiert wurde, wird in diesem Programm auf
Peleo_RA_StateMon:1 bei Schaltzustand aus
geprüft.
Das Skript kopiert zuerst die von StateMonitor gemessene Förderdauer (Laufzeit des Raumaustragungsmotors) in eine Systemvariable, damit der Wert auch in die AIO Neo – Oberfläche importiert werden kann.
Dann wird an dieser Stelle wieder versucht zu überprüfen, ob die Kalibrierung des Faktors **`Pellets_RES1_Cal_counter`** noch stimmig ist.
Dazu folgende Überlegung: Falls die Kalibrierung stimmt, sollte zu Beginn einer außerplanmäßigen Behälterauffüllung der Füllstand nicht 0% (oder, wegen der oben erwähnten Ungenauigkeit, größer 15%) sein, so könnte es sein, daß die Austragungsmenge der Kugelschleuse zu gering angenommen wurde. Tatsächlich werden also möglicherweise mehr Pellets entnommen, so daß der Behälter real schon leer ist, obwohl in der digitalen Abbildung noch ein höherer Füllstand angenommen wird.
In diesem Fall wird eine E-Mail-Benachrichtigung versandt.

Da die Befüllung des Zwischenbehälters nun abgeschlossen ist, wird der Füllgrad wieder auf 100% gesetzt.

Als nächstes wird die tatsächliche Zeitdauer für die Behälterbefüllung aus dem Lager mit der zu Beginn des Vorgangs in ***Peleo-RA early*** / ***geplant*** berechneten Zeit verglichen und für die Anzeige in der Bedienoberfläche gespeichert.
Sollte sich hier eine starke Abweichung ergeben (aktuell bei einer mehr als doppelt so langen Zeitdauer), wird eine Nachricht per Pushover versandt.

Peleo_Brennerstatus_berechnen




```
!- --- Skript Peleo_Brennerstatus_berechnen ---

var burner_counter;    !- Zähler RES1-Umläufe pro Zeitspanne
real burner_cal;       !- Kalibrierungswert Brennerleistung
real burner_power;     !- emittelte Brennerleistung
real burner_pwr_last;  !- Vorfilter für maximale Brennerleistung
real burner_maxPwr;    !- maximale ermittelte Brennerleistung für autom. Kalibrierung

!- Variablen einlesen
burner_counter = dom.GetObject("Pellets_RES1_Burner_Counter").Value();
burner_cal = dom.GetObject("Pellets_Brenner_Cal").Value();
burner_pwr_last = dom.GetObject("Pellets_Brennerstatus").Value();
burner_maxPwr = dom.GetObject("Pellets_maxPwr").Value();

!- Brennerleistung berechnen
burner_power = burner_counter / burner_cal; !- Anpassungsfaktor Umläufe pro 15 min bei 100%
Leistung
burner_power = burner_power * 100; !- Umrechnen in Prozent und hübsch machen
burner_power = burner_power.ToInteger();

!- Maximalwert zur Überprüfung der Kalibrierung speichern
if (burner_power < 0) { burner_power = 0;}
if (burner_power == burner_pwr_last)
{                  !- nur wenn der gleiche Wert 2x hintereinander kommt(Glitch-Unterdrückung)
if (burner_power > burner_maxPwr)
{
burner_maxPwr = burner_power; !- die errechnete max. Brennerleistung ohne Bereichsanpassung.
sollte 100% werden wenn richtig kalibriert
dom.GetObject("Pellets_maxPwr").State(burner_maxPwr);
}
}
if (burner_power > 100) { burner_power = 100;}
burner_counter = 0; ! Brennerzähler rücksetzen

!- Variablen zurückschreiben
dom.GetObject("Pellets_RES1_Burner_Counter").State(burner_counter);
dom.GetObject("Pellets_Brennerstatus").State(burner_power);
```

Hier ist eigentlich der Name des Programms nicht korrekt, denn es wird nicht der Zustand des Brenners (an / aus), sondern eine geschätzte Brennerleistung ermittelt.
Ursprünglich war das nur für die grafische Anzeige gedacht, damit man sehen kann, ob der Pelletskessel gerade heizt oder nicht. Das läßt sich daran festmachen, ob in letzter Zeit Brennstoff eingetragen wurde, also das Signal nRES1 des Kugelschleusen-Motors aktiv war.
Mit nur geringem Mehraufwand kann man daraus auch eine Leistungsanzeige bauen, indem man die Anzahl der Umläufe pro (fester) Zeiteinheit betrachtet.
Entsprechend der relativ trägen Veränderungsgeschwindigkeit der Brennerleistung habe ich einen Zeitraum von 15 min gewählt. Das Zeitmodul ruft entsprechend periodisch das Programm **Peleo_Brennerstatus_berechnen** auf.
Die im `Pellets_RES1_Burner_Counter` in diesem Zeitraum abgezählten Impulse müssen nun noch mit Hilfe eines Kalibrierungsfaktors so umgerechnet werden, daß eine Prozentanzeige entsteht.

Zur Ermittlung des Kalibrierungsfaktors **Pellets_Brenner_Cal** sucht man sich einen Zeitraum, in dem der Kessel längere Zeit auf 100% Leistung läuft und zählt dann ab, wie viele Impulse auf *nRES1* in den 15 min *davor* aufgelaufen sind. Waren das z.B. 13 Impulse, dann ist der Kalibrierungsfaktor **Pellets_Brenner_Cal** = 13.
Bevor der Wert für die Anzeige aufgehübscht wird, bietet sich wieder eine Möglichkeit zur Kontrolle der Kalibrierung (warum ich an allen möglichen Stellen Kalibrierungskontrollen eingebaut habe, erkläre ich am Ende des Artikels).
Zur Kontrolle wird hier zunächst der Maximalwert der ermittelten Brennerleistung ermittelt und gespeichert. Idealerweise sollte dieser ja genau 100% betragen (vorausgesetzt, der Heizung wird genug Wärme abgefordert also im Winter). Verändern sich die Gegebenheiten, z.B. durch andere Brennstoffqualität, dann wird möglicherweise mehr oder weniger Brennstoff für die gleiche Leistung benötigt und die Kalibrierung der Brennerleistung (aber wahrscheinlich auch die der Verbrauchsmessungen) stimmt nicht mehr.
Im Ergebnis wird bei real 100% Kesselleistung ein zu hoher oder zu niedriger Prozentsatz angezeigt. Dies wird in diesem Mechanismus zur Kontrolle benutzt. Die Auswertung erfolgt täglich in der *Tagesstatistik*.
Es hat sich gezeigt, daß die reine Auswertung des maximalen Werts unzuverlässig sein kann, da es gelegentlich einzelne Peaks gibt, die aus der Meßreihe herausstechen und nicht repräsentativ sind. Um zu vermeiden, daß diese das Ergebnis verfälschen, ist das Skript so aufgebaut, daß nur zwei hintereinander auftretende gleich hohe Werte bei der Suche nach dem Maximum berücksichtigt werden.
Wenn man das Verfahren zur Leistungsbestimmung kritisch betrachtet, stellt man einige Unzulänglichkeiten fest:

- Es wird nur zu festen Zeitpunkten (alle 15 min) ein neuer Wert generiert
- Durch das 15-minütige Meßintervall liegt das Ergebnis verzögert vor und kann nicht ohne weiteres z.B. mit der Anzeige am Kessel verglichen werden
- Die Auflösung der Werte ist relativ grob, z.B. bei einem Kalibrierungswert von 13 entspricht eine Stufe rund 7,7%

Diese Nachteile lassen sich durch (erheblich) aufwändigere Verfahren reduzieren. Zum Beispiel könnte der Einsatz eines gleitenden Mittelwerts wesentlich häufiger neue Ergebnisse liefern.
Weil der Brennerstatus hauptsächlich als Spaß-Funktion konzipiert war, wurde ein möglichst ressourcenschonendes Verfahren gewählt. Mit Blick auf die Kalibrierungskontrolle wäre vor allem eine bessere Auflösung wünschenswert. Das läßt sich wegen der geringen Frequenz der Schleusenumläufe aber nur mit noch größeren Meßzeiträumen realisieren.

Peleo Tagesstatistik


```
!- --- Skript Peleo Tagesstatistik ---
!- Tagesstatistik Pellets berechnen um Mitternacht

var cal;                        !- Kalibrierungsvar. Umläufe für ganzen Zwischenbeh.-Inhalt
var counter_daily;              !- RES1-Zähler Tagessumme
var counter_yesterday;          !- Umläufe Zwischenaustragung gestern
var pellets_kg_heute;           !- aktueller Stand Pelletverbrauch in kg
var pellets_kg_gestern;         !- Tagesverbrauch Pellets gestern in kg
```

```
real pellets_kwh_gestern;        !- Tagesverbrauch Pellets gestern in kWh
var pellets_kg_lfd_mon;          !- Pelletverbrauch laufender Monat in kg
var pellets_kg_letzter_mon;      !- Pelletverbrauch letzter Monat in kg
var pellets_kg_lfd_jahr;         !- Pelletverbrauch laufendes Jahr in kg
var pellets_kg_letztes_jahr;     !- Pelletverbrauch letztes Jahr in kg
real pellets_lager_t;            !- Füllstand Pelletslager in t
real pellets_lagermax_t;         !- Fassungsvermögen Pelletslager in t
real pellets_lagerfuellgrad;     !- Füllgrad Pelletslager in Prozent
integer lagerreichweite;         !- Reichweitenprognose in Tagen
var pellets_lager_schwelle;      !- Schwellwert für Füllstandswarnung in kg
var pellets_lager_niedrig;       !- Schwellwert unterschritten
var pellets_aschebox_max;        !- Fassungsvermögen Aschebox kg
var pellets_aschebox_kg;         !- Füllstand Aschebox kg
var pellets_aschebox_fuellgrad; !- Füllstand Aschebox in %
var pellets_aschefaktor;         !- Faktor zur Schätzung der Aschemenge
var pellets_asche_schwelle;      !- Schwellwert für Warnung Aschebox
var pellets_aschebox_voll;       !- Statusvariable Aschebox voll
real pellets_max_pwr;            !- max. Brennerleistung zur Kalibrierungsprüfung
real pwr_dev;                    !- Abweichung Leistungsanzeige von 100%

!- Variablen einlesen
cal = dom.GetObject("Pellets_RES1_Cal").Value();
counter_daily = dom.GetObject("Pellets_RES1_Counter").Value();
counter_yesterday = dom.GetObject("Pellets_RES1_gestern").Value();
pellets_kg_heute = dom.GetObject("Pellets_kg_heute").Value();
pellets_lager_t = dom.GetObject("Pellets_Lager_t").Value();
pellets_lagermax_t = dom.GetObject("Pellets_Lagermax_t").Value();
pellets_kg_lfd_mon = dom.GetObject("Pellets_kg_lfd_Monat").Value();
pellets_kg_lfd_jahr = dom.GetObject("Pellets_kg_lfd_Jahr").Value();
pellets_lager_schwelle = dom.GetObject("Pellets_Lager_Schwelle").Value();
pellets_lager_niedrig = dom.GetObject("Pellets_Lager_niedrig").Value();
pellets_aschebox_max = dom.GetObject("Pellets_Aschebox_max").Value();
pellets_aschebox_kg = dom.GetObject("Pellets_Aschebox_kg").Value();
pellets_aschefaktor = dom.GetObject("Pellets_Aschefaktor").Value();
pellets_asche_schwelle = dom.GetObject("Pellets_Asche_Schwelle").Value();
pellets_aschebox_voll = dom.GetObject("Pellets_Aschebox_voll").Value();
pellets_max_pwr = dom.GetObject("Pellets_maxPwr").Value();

!- --- Tageswert sichern und rücksetzen
counter_yesterday = counter_daily;
counter_daily = 0; ! Tageswert rücksetzen

!- --- Tagesverbrauch kg speichern und rücksetzen
pellets_kg_gestern = pellets_kg_heute;
pellets_kg_heute =0;
pellets_kwh_gestern = pellets_kg_gestern * 4.32; !- Umrechnung in geleistete kWh

!- --- Ascheanfall berechnen
pellets_aschebox_kg = pellets_aschebox_kg + (pellets_kg_gestern * pellets_aschefaktor);
pellets_aschebox_fuellgrad = 0;
if (pellets_aschebox_max > 0)
{
 pellets_aschebox_fuellgrad = (pellets_aschebox_kg / pellets_aschebox_max) * 100;
}
if (pellets_aschebox_fuellgrad > 100)
{
 pellets_aschebox_fuellgrad = 100;
}

!- Pruefung, ob Aschebox über Schwellwert
!- wenn voll nur einmal benachrichtigen
if ((pellets_aschebox_fuellgrad > pellets_asche_schwelle) && (!pellets_aschebox_voll))
{
 string MailText;
 string sDate = system.Date("%d.%m.%Y");
 MailText = "Pruefung der Aschebox vom " # sDate # ".\r\n";
 MailText = MailText # "\r\n [!] Der Aschestand betraegt ";
 MailText = MailText # pellets_aschebox_fuellgrad.ToString(0);
 MailText = MailText # " %.\r\n";
 string stdout;
 string stderr;
 system.Exec("/etc/config/addons/email/email 04 '"+MailText+"'",&stdout,&stderr);
 pellets_aschebox_voll = 1;
}
```

```
!- falls die Aschebox geleert wurde
if ((pellets_aschebox_fuellgrad < pellets_asche_schwelle) && (pellets_aschebox_voll))
{
 pellets_aschebox_voll = 0;
}

!- --- Lagerstand berechnen
pellets_lager_t = pellets_lager_t - (pellets_kg_gestern / 1000);
if (pellets_lager_t < 0)
{pellets_lager_t = 0.0;} ! Unterlauf abfangen
pellets_lagerfuellgrad = (pellets_lager_t / pellets_lagermax_t) *100; ! Füllgrad in % für
Anzeige

!- --- gleitenden Durchschnitt der Tagesverbräuche der letzten 10 Tage berechnen (kg)
integer n = 9;          !- legt die max. Anzahl der Elemente in der Speicherliste fest
string list_in;         !- String mit Speicherliste. Muss von Systemvariable befüllt werden
string list_out = "";   !- String mit neuer Speicherliste. Muss in Systemvariable geschrieben
werden
real new_element;       !- Neues Element, was dem gleitenden Durchschnitt hinzugefügt wird
(z.B. aus Systemvariable)
real kg_Avg10 = 0.0;    !- Gleitender Durchschnitt aus n Listenelementen und dem neuen Element
string element;
real summand;
real sum = 0.0;
new_element = pellets_kg_gestern;                    !- neuen Tageswert der Liste hinzufügen
list_in = dom.GetObject("Pellets_Day_Avg10_List").Value(); !- Speicherliste einlesen
integer i = 0;
foreach(element, list_in.Split(","))
{
 summand = element.ToFloat();
 sum= sum + summand;
 i = i + 1;
 if (i < n)
 {
  list_out = list_out + "," + summand.ToString(2);
 }
}
kg_Avg10 = (sum + new_element) / (i + 1);
list_out = new_element.ToString(2) + list_out;
dom.GetObject("Pellets_Day_Avg10_List").State(list_out) !- Speicherliste zurückschreiben

!- --- Lagerreichweitenprognose berechnen
lagerreichweite = 0;
if (kg_Avg10 > 0)
{
 lagerreichweite = (pellets_lager_t *1000) / kg_Avg10;
}
lagerreichweite = lagerreichweite.ToInteger();

!- -- Monatsverbrauch kg speichern und rücksetzen
if (system.Date("%d").ToInteger() == 1) {
 pellets_kg_letzter_mon = pellets_kg_lfd_mon;
 pellets_kg_lfd_mon = 0;
 dom.GetObject("Pellets_kg_letzter_Monat").State(pellets_kg_letzter_mon);
 dom.GetObject("Pellets_kg_lfd_Monat").State(pellets_kg_lfd_mon);
}

!- -- Jahresverbrauch kg speichern und rücksetzen
if ((system.Date("%m").ToInteger() == 1) && (system.Date("%d").ToInteger() == 1)) {
 pellets_kg_letztes_jahr = pellets_kg_lfd_jahr;
 pellets_kg_lfd_jahr = 0;
 dom.GetObject("Pellets_kg_letztes_Jahr").State(pellets_kg_letztes_jahr);
 dom.GetObject("Pellets_kg_lfd_Jahr").State(pellets_kg_lfd_jahr);
}

!- Pruefung, ob Lagerstand die Meldeschwelle unterschreitet
!- wenn Lagerstand niedrig nur einmal benachrichtigen
if (((pellets_lager_t * 1000) < pellets_lager_schwelle) && (!pellets_lager_niedrig))
{
 string MailText;
 string sDate = system.Date("%d.%m.%Y");
 MailText = "Prüfung des Pelletslager vom " # sDate # ".\r\n";
 MailText = MailText # "\r\n [!] Der Lagerstand ist niedrig (";
```

```
 MailText = MailText # pellets_lager_t.ToString(3);
 MailText = MailText # " t).\r\n";
 MailText = MailText # "\r\n Prognose Rest-Reichweite: ";
 MailText = MailText # lagerreichweite.ToString(0);
 MailText = MailText # " Tage.\r\n";
 string stdout;
 string stderr;
 system.Exec("/etc/config/addons/email/email 03 '"+MailText+"'",&stdout,&stderr);
 pellets_lager_niedrig = 1;
 dom.GetObject("Pellets_Lager_niedrig").State(pellets_lager_niedrig);
}

!- falls sich das Lager durch ein Wunder gefüllt hat (oder die Schwelle verändert wurde)
if (((pellets_lager_t * 1000) > pellets_lager_schwelle) && (pellets_lager_niedrig))
{
 pellets_lager_niedrig = 0;
 dom.GetObject("Pellets_Lager_niedrig").State(pellets_lager_niedrig);
}

!- Prüfung der Kalibrierung über die Brennerleistung
pwr_dev = 0;
if (pellets_max_pwr > 0)
{
 pwr_dev = pellets_max_pwr - 100.0; !- Abweichung von der Maximalleistung 100%
 if ((pwr_dev > 10) || (pwr_dev < -10))
 {
  string MailText;
  string sDate = system.Date("%d.%m.%Y");
  MailText = "Kalibrierungskontrolle Pelletssystem vom " # sDate # ".\r\n";
  MailText = MailText # "\r\n Abweichung der Brennerleistung: ";
  MailText = MailText # pwr_dev.ToString(3);
  MailText = MailText # " %).\r\n";
  MailText = MailText # "\r\n Bitte Kalibrierung kontrollieren";
  string stdout;
  string stderr;
  system.Exec("/etc/config/addons/email/email 05 '"+MailText+"'",&stdout,&stderr);
  dom.GetObject("Pellets_Cal_dev").State(1); !- Statusflag für Bedienoberfläche setzen
 }
 pellets_max_pwr = 0; !- neue Prüfung am nächsten Tag
 dom.GetObject("Pellets_maxPwr").State(pellets_max_pwr);
}

!- --- Variablen zurückschreiben
dom.GetObject("Pellets_RES1_gestern").State(counter_yesterday);
dom.GetObject("Pellets_RES1_Counter").State(counter_daily);
dom.GetObject("Pellets_kg_heute").State(pellets_kg_heute);
dom.GetObject("Pellets_kg_gestern").State(pellets_kg_gestern);
dom.GetObject("Pellets_kWh_gestern").State(pellets_kwh_gestern);
dom.GetObject("Pellets_Lager_t").State(pellets_lager_t);
dom.GetObject("Pellets_Lagerfuellgrad").State(pellets_lagerfuellgrad);
dom.GetObject("Pellets_kg_Avg10").State(kg_Avg10);
dom.GetObject("Pellets_Lagerreichweite").State(lagerreichweite);
dom.GetObject("Pellets_Aschebox_kg").State(pellets_aschebox_kg);
dom.GetObject("Pellets_Aschebox_Fuellgrad").State(pellets_aschebox_fuellgrad);
dom.GetObject("Pellets_Aschebox_voll").State(pellets_aschebox_voll);
dom.GetObject("Pellets_PwrAbw").State(pwr_dev);
```

Kurz nach Mitternacht, wenn alle schlafen, schlägt die Stunde der ***Peleo Tagesstatistik***.
Dieses relativ umfangreiche Skript berechnet, wie der Name schon vermuten läßt, alle möglichen Verbrauchs- und statistikwerte und kümmert sich außerdem darum, danach zu schauen, ob bald mal wieder Pellets bestellt werden sollten oder die Aschebox ausgeleert werden will.
Für alle, die bis hierhin durchgehalten haben: Das ist die letzte aufwändigere Funktion, danach kommt nur noch Kleinkram.
Eine Aufgabe, die in diesem Skript erledigt wird, ist die Speicherung der „statistischen Verbrauchswerte“. Dabei wird der jeweils aktuelle Verbrauchswert in eine Variable umkopiert und danach zurückgesetzt. Dies passiert z.B. gleich am Anfang mit dem Tagesverbrauch, der als „Tagesverbrauch gestern“ gespeichert wird.

Weiter unten geschieht das selbe mit Monats- und Jahresverbrauch, wobei hier die Abspeicherung über die Prüfung des Tagesdatums (1.x für Monats- und 1.1. für Jahresbeginn) gesteuert wird. Da die Variablen vom CCU Historian archiviert werden, können damit recht einfach nette Auswertungen und Diagramme erstellt werden.
Weiter wird in diesem Skript einmal täglich der Ascheanfall aus der an diesem Tag verbrauchten Pelletmenge über den Umrechnungsfaktor **`Pellets_Aschefaktor`** abgeschätzt.
Der kg-Wert wird in einen Prozentwert (Füllgrad) umgerechnet und mit der vorgegebenen Meldeschwelle verglichen. Falls der Aschestand die Schwelle überschreitet, wird eine E-Mail verschickt. Dabei wird das Flag **`Pellets_Aschebox_voll`** gesetzt und in der Bedingung für den Mailversand mit geprüft, so daß die Mail-Benachrichtigung nur einmal verschickt wird.
Umgekehrt wird geprüft, ob der Schwellwert ggf. wieder unterschritten wurde, während das Flag noch aktiv ist, und in diesem Fall zurückgesetzt.

Punkt, Punkt, Komma Strich…

… gilt bei der HomeMatic nicht.

Könnte man so oder ähnlich als Merkregel dichten, um sich an die Eigenheiten der Berechnungen in Skripten zu erinnern. Die Skriptlogik kennt nicht Punkt- vor Strichrechnung und arbeitet einen Term von rechts nach links ab. Klingt komisch, ist es auch und irgendwann fällt man (also zumindest ich) dann doch wieder drauf rein, wie ich an diesem Beispiel wieder erfahren habe:

Der Füllgrad der Aschebox in Prozent errechnet sich aus dem Verhältnis der aktuellen Aschemenge in kg zur maximalen Füllmenge, skaliert mit dem Faktor 100 in den Wertebereich 0…100. Als Formel ausgedrückt:

$$Füllgrad = \frac{Füllstand\ aktuell}{Füllstand\ max} \cdot 100$$

Als Code sieht das so eigentlich nicht schlecht aus:

```
pellets_aschebox_fuellgrad = pellets_aschebox_kg / pellets_aschebox_max * 100;
```

Reine Punktrechnung, kann eigentlich nichts passieren. Trotzdem kommt überraschend ein um Zehnerpotenzen zu kleines (und damit natürlich falsches) Ergebnis raus.
Grund ist die Abarbeitung von rechts nach links, die zuerst `100 * pellets_aschebox_max` rechnet und `pellets_aschebox_kg` durch dieses Zwischenergebnis teilt.

Aus der obigen Formel wird so

$$Füllgrad = \frac{Füllstand\ aktuell}{Füllstand\ \max \cdot 100}$$

und damit leider ein falsches Ergebnis. Daher muss auch hier geklammert werden:

```
pellets_aschebox_fuellgrad = (pellets_aschebox_kg / pellets_aschebox_max) * 100;
```

Analog zur Schäzung des Aschestands wird auch der Füllstand des Pelletlagers berechnet.

Für die Reichweitenprognose, also die Schätzung, wieviel Tage mit dem aktuellen Lagerstand noch geheizt werden kann, muß zunächst ein Tagesverbrauchswert ermittelt werden. Am einfachsten wäre dafür der Verbrauch des letzten Tages zu nehmen. Der Nachteil dabei ist, daß Schwankungen im Tagesverbrauch so direkt durchschlagen und die Reichweitenprognose stark schwanken kann. Daher wird hier zunächst ein gleitender Mittelwert über die letzten 10 Tage berechnet.

Das ist im Prinzip eine Mittelwertbildung über die letzten 10 Verbrauchswerte, die jeden Tag eine Werteposition weitergeschoben wird.
Neben dem aktuellen Tageswert werden also die Verbrauchswerte der letzten 9 Tage benötigt. Dafür 9 Systemvariablen zu verwenden, wirkt ein bißchen unelegant. Andererseits gibt es in HomeMatic-Welt keine Arrays. Deshalb habe ich den Zwischenspeicher als eine Zeichenkette realisiert, in der die Werte durch Komma getrennt gespeichert werden.
Durch die Schleifenkonstruktion mit `foreach` wird der Liste `list_in` in jedem Durchlauf ein `element` entnommen und aufaddiert.
Das Element wird anschließend an eine neue Liste `list_out` angefügt, sofern es nicht das letzte Element ist. Anstelle dessen wird nach Ende der Schleife vorne das aktuelle Element hinzugefügt.

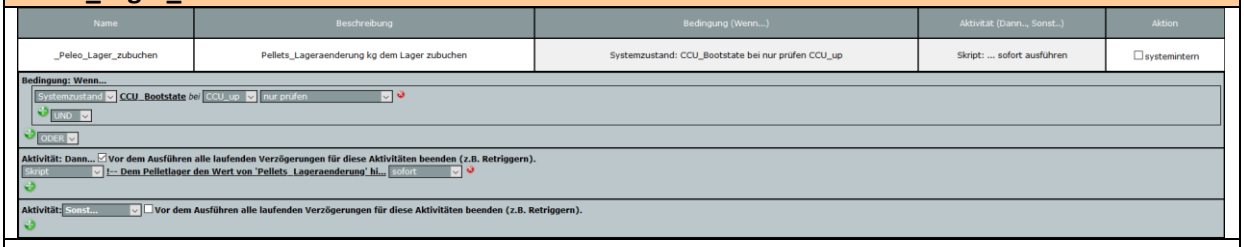
```
foreach(element, list_in.Split(","))
{
summand = element.ToFloat();
 sum= sum + summand;
 i = i + 1;
 if (i < n)
 {
  list_out = list_out + "," + summand.ToString(2);
 }
}
kg_Avg10 = (sum + new_element) / (i + 1);
list_out = new_element.ToString(2) + list_out;
```

Mit dem gebildeten Mittelwert wird dann im nächsten Schritt die Restreichweite errechnet. Dazu wird der Lagerbestandswert im kg umgerechnet und durch den mittleren Tagesverbrauch geteilt. Das Ergebnis ist die geschätzte Lagerreichweite in Tagen.

Das Tagesstatistik-Skript wertet die in **Peleo_Brennerstatus_berechnen** aufgezeichnete maximale Brennerleistung aus. Die Methode wurde an dieser Stelle beschrieben. Hier erfolgt nun noch die Auswertung, ob sich am letzten Tag irgendwann einmal die Brennerleistung in einem Fenster von +/- 10% um den Wert 100% befunden hat. War dies nicht der Fall, könnte sich die Kalibrierung verschoben haben und es wird eine Benachrichtigung per Mail generiert.
Der aufgezeichnete Maximalwert wird wieder auf 0 zurückgesetzt, so daß eine neue Prüfung am kommenden Tag erfolgen kann.
Die Methode hat als Grundannahme, daß die Brennerleistung wenigstes einmal am Tag 100% erreicht. Das muss natürlich nicht unbedingt der Fall sein. Wenn im Sommer die Solarabdeckung groß ist, kann es sein, daß der Kessel gar nicht aktiv werden muß. Bei einer Brennerleistung von 0% wird deshalb nicht benachrichtigt. Es bleibt der Fall übrig, daß der Kessel zwar anspringt, aber moduliert betrieben wird, also nur auf Teilleistung zuheizt. Sollte das vorkommen, wird fälschlicherweise die Benachrichtigung generiert. Man muss sich also die Betriebssituation anschauen und die Ursache der Meldung prüfen.

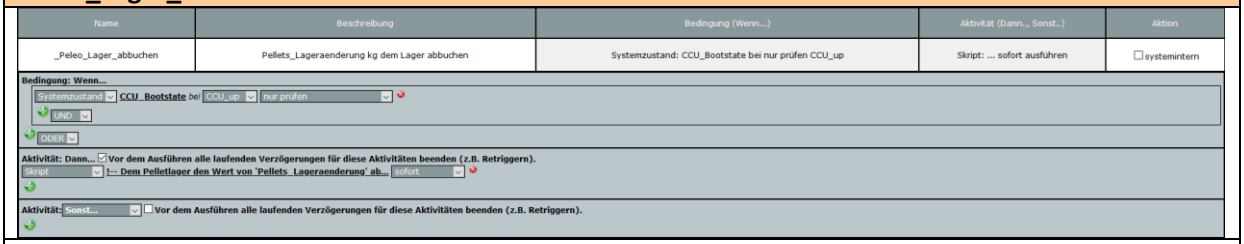
Die folgenden beiden Programme dienen dazu, den Lagerstand anzupassen. Dies soll auf einfache Weise über die AIO Neo App möglich sein.
Über einen Schieberegler, der auf die Systemvariable `Pellets_Lageraenderung` wirkt, wird die Pellets-Menge eingestellt, um die der Lagerstand erhöht oder verringert werden soll.
Die Durchführung der Änderung erfolgt über zwei Schaltflächen „zubuchen“ / „abbuchen“ , die das entsprechende Programm aufrufen. Hier geschieht im Wesentlichen die Addition / Subtraktion des Lagerfüllstands. Außerdem werden Lagerfüllgrad und die Reichweite neu berechnet, damit die entsprechenden Anzeigen aktuell sind (was ansonsten erst nachts in der Tagesstatistik passieren würde).

Peleo_Lager_zubuchen




```
!- --- Skript Peleo_Lager_zubuchen ---
!- Dem Pelletlager den Wert von 'Pellets_Lageraenderung' hinzubuchen

real pellets_lager_t;         !- Füllstand Pelletslager in t
real pellets_lagermax_t;      !- Fassungsvermögen Pelletslager in t
real pellets_lagerfuellgrad; !- Füllgrad Pelletslager in Prozent
integer lagerreichweite;      !- Reichweitenprognose in Tagen
var pellets_lager_schwelle;   !- Schwellwert für Füllstandswarnung in kg
var pellets_lager_niedrig;    !- Schwellwert unterschritten
var pellets_lageraenderung;   !- Betrag der Zu- / Abbuchung
real kg_Avg10;                !- Verbrauchsschnitt für Reichweitenberechnung

!- --- Variablen einlesen
pellets_lager_t = dom.GetObject("Pellets_Lager_t").Value();
pellets_lagermax_t = dom.GetObject("Pellets_Lagermax_t").Value();
pellets_kg_lfd_mon = dom.GetObject("Pellets_kg_lfd_Monat").Value();
pellets_kg_lfd_jahr = dom.GetObject("Pellets_kg_lfd_Jahr").Value();
pellets_lager_schwelle = dom.GetObject("Pellets_Lager_Schwelle").Value();
pellets_lager_niedrig = dom.GetObject("Pellets_Lager_niedrig").Value();
pellets_lageraenderung= dom.GetObject("Pellets_Lageraenderung").Value();
kg_Avg10 = dom.GetObject("Pellets_kg_Avg10").Value();

!- --- zubuchen
pellets_lager_t = pellets_lager_t + (pellets_lageraenderung / 1000);
if (pellets_lager_t > pellets_lagermax_t)
{
pellets_lager_t = pellets_lagermax_t;
}

!- --- Lagerfüllgrad berechnen
pellets_lagerfuellgrad = (pellets_lager_t / pellets_lagermax_t) *100; !- Füllgrad in % für
Anzeige

!- --- Lagerreichweitenprognose berechnen
lagerreichweite = 0;
if (kg_Avg10 > 0)
{
lagerreichweite = (pellets_lager_t *1000) / kg_Avg10;
}
lagerreichweite = lagerreichweite.ToInteger();

!- falls durch die Zubuchung der Schwellwert überschritten wurde
if (((pellets_lager_t * 1000) > pellets_lager_schwelle) && (pellets_lager_niedrig))
{
pellets_lager_niedrig = 0;
dom.GetObject("Pellets_Lager_niedrig").State(pellets_lager_niedrig);
}

!- --- Variablen zurückschreiben
dom.GetObject("Pellets_Lager_t").State(pellets_lager_t);
dom.GetObject("Pellets_Lagerfuellgrad").State(pellets_lagerfuellgrad);
dom.GetObject("Pellets_Lagerreichweite").State(lagerreichweite);
```

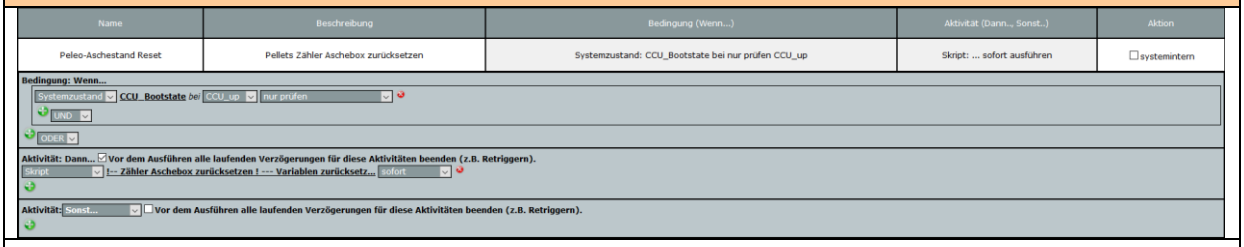
Peleo_Lager_abbuchen




```
!- --- Skript Peleo_Lager_abbuchen ---
!- Dem Pelletlager den Wert von 'Pellets_Lageraenderung' abbuchen

real pellets_lager_t;         !- Füllstand Pelletslager in t
real pellets_lagermax_t;      !- Fassungsvermögen Pelletslager in t
real pellets_lagerfuellgrad; !- Füllgrad Pelletslager in Prozent
integer lagerreichweite;      !- Reichweitenprognose in Tagen
var pellets_lager_schwelle;   !- Schwellwert für Füllstandswarnung in kg
var pellets_lager_niedrig;    !- Schwellwert unterschritten
var pellets_lageraenderung;   !- Betrag der Zu- / Abbuchung
real kg_Avg10;                !- Verbrauchsschnitt für Reichweitenberechnung

!- --- Variablen einlesen
pellets_lager_t = dom.GetObject("Pellets_Lager_t").Value();
pellets_lagermax_t = dom.GetObject("Pellets_Lagermax_t").Value();
pellets_kg_lfd_mon = dom.GetObject("Pellets_kg_lfd_Monat").Value();
pellets_kg_lfd_jahr = dom.GetObject("Pellets_kg_lfd_Jahr").Value();
pellets_lager_schwelle = dom.GetObject("Pellets_Lager_Schwelle").Value();
pellets_lager_niedrig = dom.GetObject("Pellets_Lager_niedrig").Value();
pellets_lageraenderung= dom.GetObject("Pellets_Lageraenderung").Value();
kg_Avg10 = dom.GetObject("Pellets_kg_Avg10").Value();

!- --- abbuchen
pellets_lager_t = pellets_lager_t - (pellets_lageraenderung / 1000);
if (pellets_lager_t < 0)
{
pellets_lager_t = 0;
}

!- --- Lagerfüllgrad berechnen
pellets_lagerfuellgrad = (pellets_lager_t / pellets_lagermax_t) *100; ! Füllgrad in % für
Anzeige

!- --- Lagerreichweitenprognose berechnen
lagerreichweite = 0;
if (kg_Avg10 > 0)
{
lagerreichweite = (pellets_lager_t *1000) / kg_Avg10;
}
lagerreichweite = lagerreichweite.ToInteger();

!- falls durch die Abbuchung der Schwellwert unterschritten wurde
if (((pellets_lager_t * 1000) < pellets_lager_schwelle) && (!pellets_lager_niedrig))
{
pellets_lager_niedrig = 1;
dom.GetObject("Pellets_Lager_niedrig").State(pellets_lager_niedrig);
}

!- --- Variablen zurückschreiben
dom.GetObject("Pellets_Lager_t").State(pellets_lager_t);
dom.GetObject("Pellets_Lagerfuellgrad").State(pellets_lagerfuellgrad);
dom.GetObject("Pellets_Lagerreichweite").State(lagerreichweite);
```

Das Programm **Peleo-Aschestand Reset** wird analog benutzt, um den Aschestand-Zähler zurückzusetzen, wenn die Box geleert wurde. Da man die Box wohl immer vollständig leeren wird, genügt es hier, bei Betätigung der Schaltfläche die Variablen einfach auf Null zu setzen.

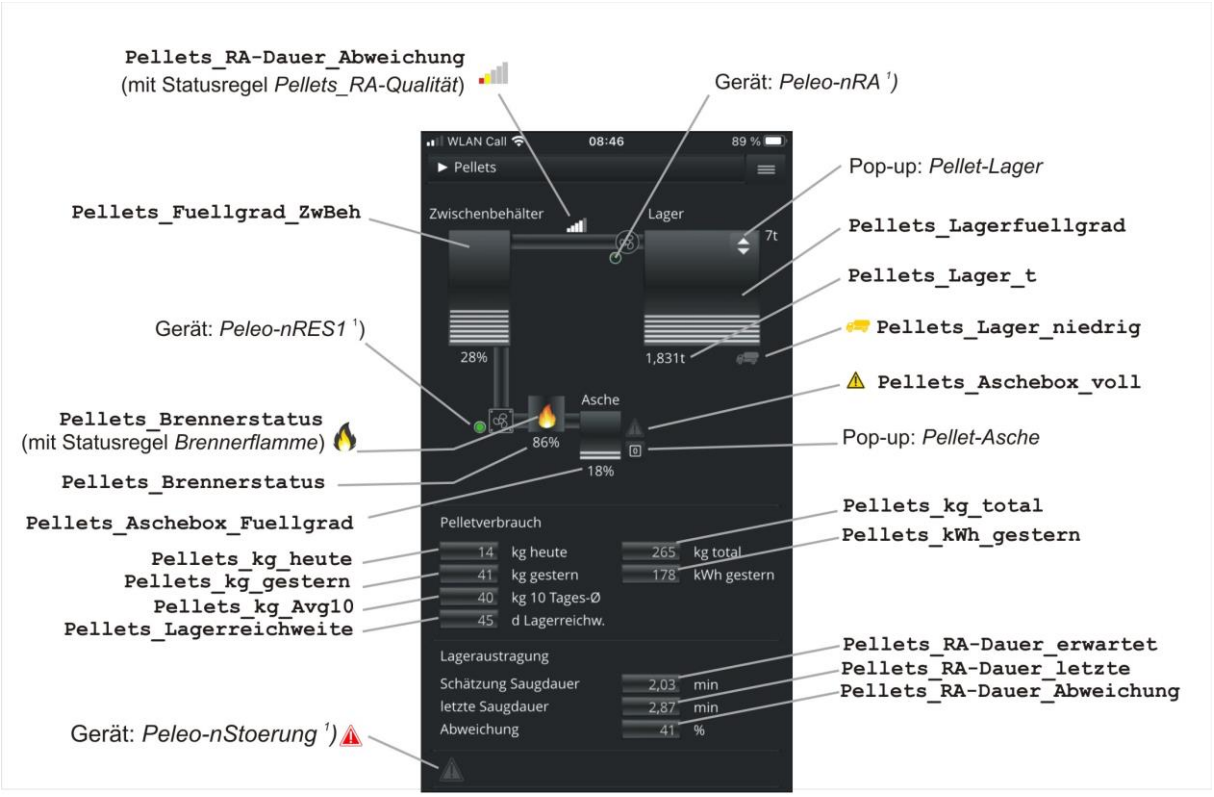
Peleo-Aschestand Reset




```
!- --- Skript Peleo-Aschestand Reset ---
!- Zähler Aschebox zurücksetzen

!- --- Variablen zurücksetzen
dom.GetObject("Pellets_Aschebox_kg").State(0.0);
dom.GetObject("Pellets_Aschebox_Fuellgrad").State(0);
dom.GetObject("Pellets_Aschebox_voll").State(0);
```

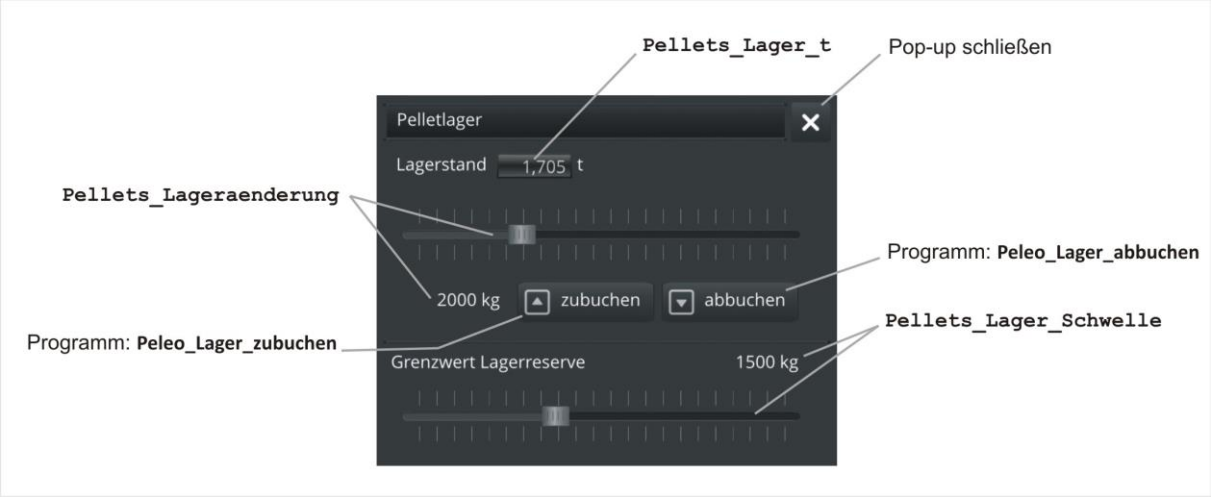
Visualisierung in der App

In den folgenden Grafiken ist die Verknüpfung der Variablen und Programme in der AIO Neo – Oberfläche dargestellt:



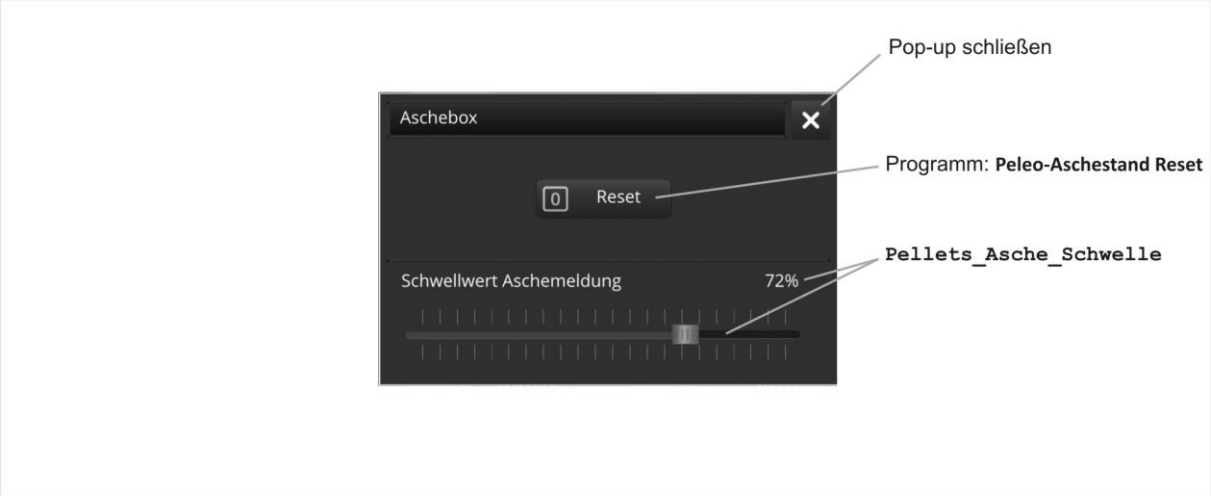
1) Icon invertiert (State off = aktives Symbol, State on = inaktives Symbol)

Für die Änderung des Lagerfüllstands und das Einstellen der Meldeschwelle für das Lager ist ein Pop-up – Fenster zuständig:



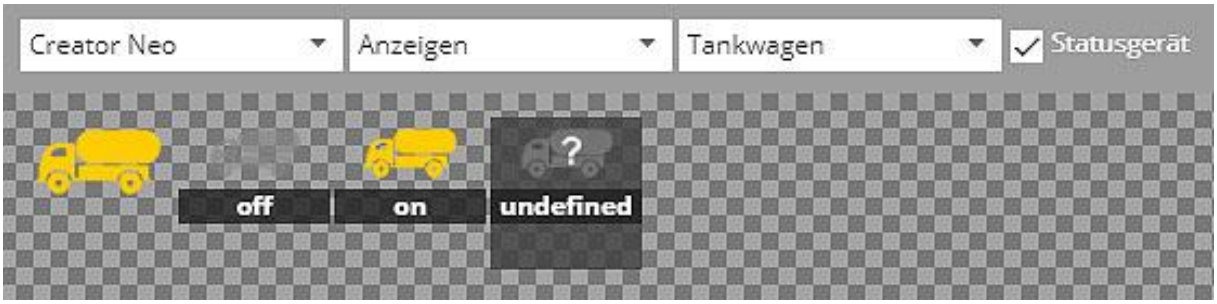
Variablenzuordnung und Programmaufrufe im Pop-up "Pelletlager"

Und mit dem folgenden Pop-up – Fenster erfolgt die Verwaltung der Aschebox:

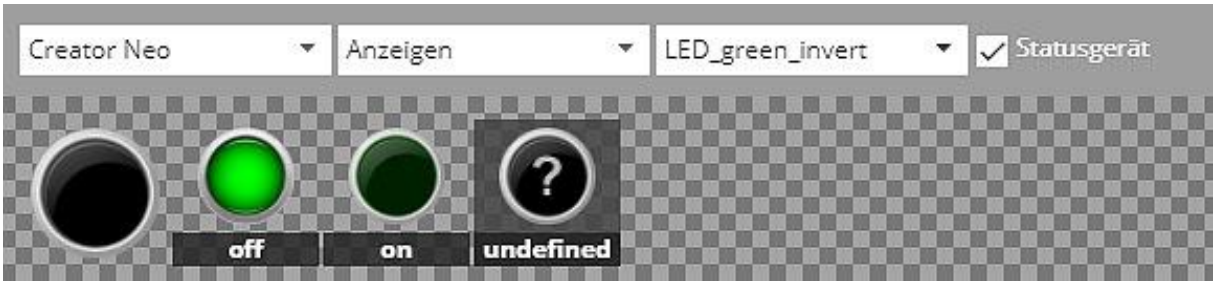


Variablenzuordnung und Programmaufrufe im Pop-up "Aschebox"

Für die Visualisierung des Systemzustands werden ein paar Statusvariablen verwendet, die folgendermaßen definiert sind:

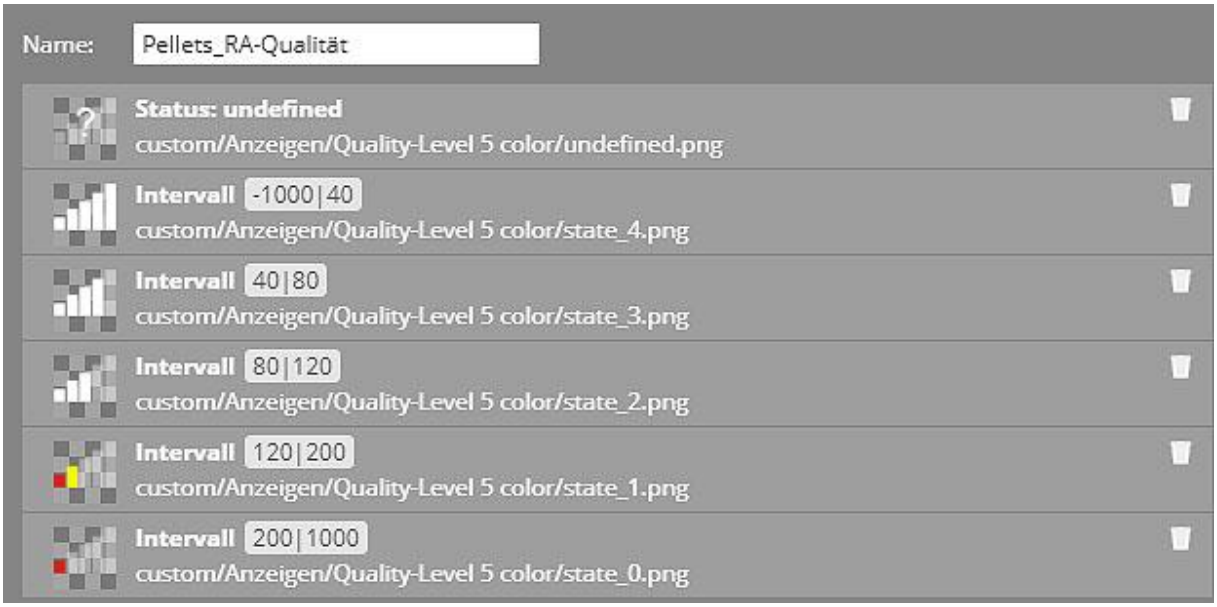


Bei den beiden folgenden Statusgeräte-Symbolen ist zu beachten, daß die Signale, die dargestellt werden sollen, invertiert sind. Daher sind die Icons so definiert worden, daß das “aktive” Symbol (z.B. LED ein) dem Zustand *aus* zugeordnet wurde und umgekehrt.
Das spart die Verwendung einer Statusregel.



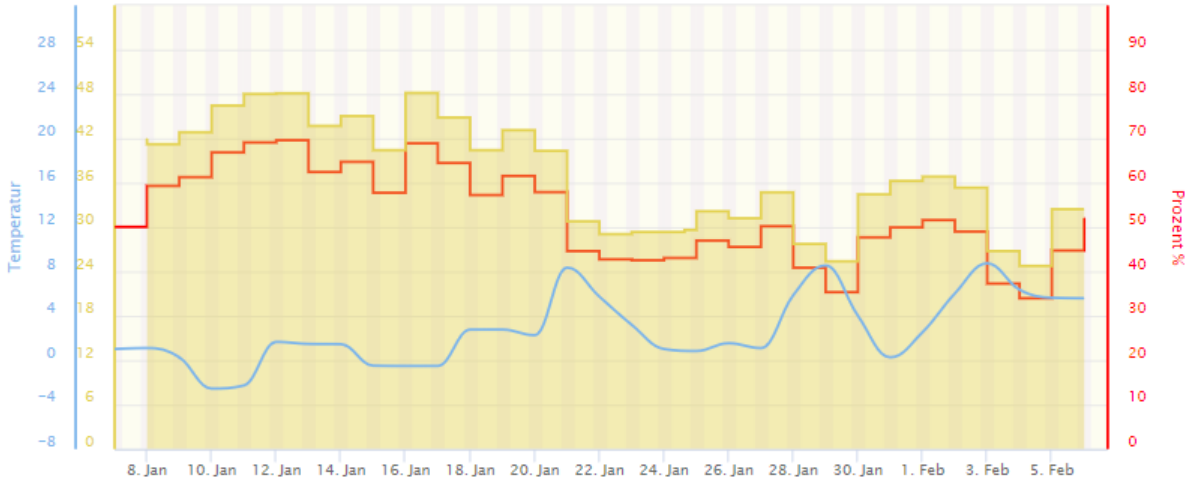
Die Grafik der Brennerflamme soll auch optisch die aktuelle Leistung widerspiegeln.
Dazu wurden fünf Grafiken mit unterschiedlicher Flammengröße angelegt und in einem Statusgerät e-Symbol zusammengefasst.
Der Wertebereich der Variable `Pellets_Brennerstatus` (0…100) wird in einer Statusregel *Brennerflamme* entsprechend in fünf Intervalle aufgeteilt, die jeweils einem der unterschiedlichen Intensitäts-Symbole zugewiesen werden:

Auf ähnliche Weise erfolgt die grafische Darstellung der Qualität der Lageraustragung. Dies sollte in einer Qualitätsanzeige mit Balken (ähnlich wie beim Handy-Empfang) visualisiert werden.
Auch hier wurden fünf Grafiken mit entsprechend unterschiedlicher Anzahl Balken erstellt und über eine Statusregel (*Pellets_RA-Qualität*) dem Wertebereich zugeordnet.
Hier muß man allerdings aufpassen, denn obwohl die Abweichung in Prozent ausgerechnet wird, ist der Maximalwert nicht 100. Es ist ja möglich, daß die Austragungsdauer ein Vielfaches länger ist, als die vorausberechnete Schätzung. Tatsächlich ist eine doppelt so lange Saugdauer noch gar nicht so dramatisch. Umgekehrt kann die Abweichung auch negativ werden (es ging schneller, als erwartet). Für die Statusregel habe ich deshalb einmal einen Wertebereich von -1000 bis 1000 angenommen und versuchsweise den Qualitätsstufen zugeordnet. (Übrigens zeigt die AIO Neo – Oberfläche das Symbol *undefined* an, wenn der Wert außerhalb des definierten Bereichs liegt.)



Erfahrungen – oder: So ganz fertig ist man nie…

So nach ein paar Monaten probieren, beobachten, weiterentwickeln und feintunen wirkte das ganze System eigentlich recht stabil und die gesammelten Werte schienen auch plausibel und verlässlich zu sein.
Im Hinterkopf hatte ich das Projekt schon abgeschlossen, da kam der 22. Januar:



braun: Pelletsverbrauch kg/Tag – rot: stündlich gemittelte Brennerleistung % - blau: Außentemperatur °C

In der Grafik kann man sehen, daß der gemessene Pelletsverbrauch plötzlich deutlich geringer wird, obwohl die Außentemperatur (bis auf eine kurze Spitze) ziemlich gleich geblieben war.
Außerdem erreichte die Anzeige der Brennerleistung nicht mehr 100%, auch wenn die Anzeige am Heizkessel diesen Wert hatte. Letzteres war eigentlich das auffälligste Zeichen, daß sich etwas verändert haben musste.
Was sich verändert hatte, war (auch für die gesamte Nachbarschaft) recht leicht auszumachen, denn um 6:45 morgens rauschten 6 t Pelletsnachschub ins Lager. Da mein Lager ein Maulwurf-Austragungssystem hat, wurden gleich auch die neuen Pellets von oben in den Heizkessel gefördert. Die neuen Pellets mussten irgendwie andere Eigenschaften haben, die meine Kalibrierung ein bisschen verschieben.
Mit Blick auf die zu niedrige Anzeige der Brennerleistung bedeutet das, daß die Umlaufschleuse nun offenbar weniger Umläufe machen muss, um die gleiche Energiemenge feuern zu können.
Dadurch erklärt sich auch, warum die gemessene Verbrauchsmenge geringer wurde, denn diese wird ja ebenfalls aus den Schleusenumläufen ermittelt.
Was konnte als Ursache in Frage kommen? Es könnte sein, daß die Pellets „besser brennen“, also die gleiche kg-Menge Pellets mehr Energie spendet. Oder aber, die Schleuse fördert bei einem Umlauf plötzlich mehr kg Pellets. Was bedeutet, es passen mehr Pellets in die Schleuse. Hier muß man sich in Erinnerung rufen, daß Pellets zum einen ein Naturprodukt sind, das gewissen Schwankungen unterlegen ist und zudem als Schüttgut auch nicht gerade dafür berühmt sind, variationsarm zu sein. Der Füllfaktor, das Verhältnis von Material zu Luft im Lager, Behälter oder in der Schleuse kann variieren.

In der Tabelle unten habe ich versucht, die Auswirkungen von verschiedenen variablen Eigenschaften auf die verschiedenen Messwerte zusammenzustellen:

variable Eigenschaft	Anzeige Brennerleistung	Verbrauchs-anzeige	Anzeige Füllgrad Zwischenbeh.	Abweichung Saugdauer
Energiegehalt	↓	-	-	-
Wichte	↓	↓	-	-
Füllfaktor	↓	↓	? (↑)	? (↑)

↓ = sinkt / zu niedrig, wenn variable Eigenschaft steigt, ↑ = steigt / zu hoch

Die Auswirkungen eines besseren Füllfaktors sind schwierig zu beurteilen. Nach meiner Beobachtung passen dann mehr Pellets in die Umlaufschleuse, so daß mit einem Umlauf mehr kg Pellets und damit mehr „Energie“ gefördert wird. Die Verbrauchsanzeige ist zu niedrig, da sie ein zu geringe Gewichtsmenge pro Umlauf annimmt und die Brenneranzeige ist zu gering, weil weniger Umläufe/Zeiteinheit als gedacht schon die Menge Pellets fördern, die für z.B. 100% Leistung benötigt werden.
Man sollte annehmen, daß auch mehr Pellets in den Zwischenbehälter passen und sich dadurch das Verhältnis zur Austragungsmenge nicht ändert – die Anzeige wäre weiter korrekt. Meine Beobachtung war aber auch eine Veränderung der Anzeige und dadurch bedingt eine Abweichung der Saugdauer, die daraus errechnet wird. Der Füllstand des Zwischenbehälters war in Wirklichkeit niedriger als errechnet, wodurch bei der Saugdauer eine positive Abweichung angezeigt wird. Möglicherweise wirkt sich eine Änderung des Füllfaktors bei der kleinen Kugelschleuse stärker aus, als beim viel größeren Zwischenbehälter.

Meine Vermutung, was zur Änderung der Werte nach der Lagerbefüllung führte ist, daß die Pellets mit höherem Druck eingeblasen wurden und dadurch mehr Bruchstücke entstanden sind. Die kleineren Stücke passen dann besser in die Kugelschleuse, was den Füllgrad verbessert.
Dazu passt ein bisschen die Situation vor Ort, denn bei mir muß recht viel Schlauch zwischen Fahrzeug und Lager verlegt werden, weswegen die Pelletsqualität beim Einblasen leiden kann.

Als Konsequenz wurden in die Programme die besprochenen Kalibrierungskontrollen eingebaut. Damit steht ein Mittel zur Verfügung, das einigermaßen automatisch auf mögliche Abweichungen hinweist. Diese müssen dann manuell nachgeprüft und die Kalibrierungswerte gegebenenfalls nachgestellt werden.
Eine komplette Kalibrierung mit Ausmessen der Pelletsmenge etc. ist dabei wohl nicht erforderlich, weil sich die angezeigten prozentualen Abweichungen quasi als Korrekturfaktor auf die Kalibrierwerte anwenden lassen. Man muss sich lediglich bewusst sein, daß z.B. die Anzeige der Brennerleistung ziemlich stufig ist, was zu entsprechend groben Korrekturwerten führt. Im Rahmen der Gesamttoleranz ist das bestimmt vertretbar, so lange man das nicht zu oft hintereinander macht, so daß sich die Fehler aufaddieren und die Genauigkeit wegdriftet.
Als weitere Methode zur Ermittelung der Korrekturwerte, die nicht zu aufwändig ist, ist die Bestimmung der Abweichung des Zwischenbehälter-Füllgrads.

Die Berechnung der beiden Methoden (über Brennerleistung und Füllgrad) habe ich einmal mit Excel durchgeführt.
Mit den konkreten Werten aus dem Januar sieht man im Bild unten, daß die Füllgrad-Methode eine Abweichung von 20% und die Brennermethode eine Abweichung von 14% ermittelt.
Die beiden Werte sind also recht plausibel, wenn man die Schrittweite der Brenneranzeige von fast 7% einkalkuliert:

	A	B	C	D	E	F	G
1	**Korrekturfaktor aus Zwischenbehälter**						
2							
3	angezeigter Füllgrad:	65	% (nur linearer Teil, bis max. 80%)				
4	tats. Pelletsstand:	22	cm vom oberen Rand gemessen				
5			(bei kleinen Werten Verzerrung durch Saugglocke beachten)				
6	theor. Stand:	17,5	cm				
7	Abweichung:	4,5	cm				
8	Abweichung in %:	20,5%					
9							
10	aktuelles RES1-kg-Ratio:	0,0475	kg				
11	neues RES1-kg-Ratio:	0,0572	kg				
12							
13							
14	**Korrekturfaktor aus Brennerleistung**						
15							
16	angezeigte Leistung	86	%				
17	tatsächliche Leistung	100	% (Achtung Zeitverzögerung)				
18							
19	Differenz:	14	%				
20							
21	aktuelles RES1-kg-Ratio:	0,0475	kg				
22	neues RES1-kg-Ratio:	0,0542	kg				
23							
24	aktueller Brenner_Cal:	15	Schrittweite:	6,67	%		
25	neuer Brenner_Cal:	12,9000					

Excel-Tabelle zur Nachführung der Kalibrierung bei Abweichungen (blau=Eingabefelder)

Zur komfortablen Einstellung der Kalibrierungswerte kann die grafische Oberfläche noch um ein Pop-up – Fenster mit entsprechenden Schiebereglern ergänzt werden.

Fazit

Der Umfang des Projekts ist deutlich größer geworden, als ursprünglich gedacht.
Im Ergebnis ist eine praxistaugliche Darstellung des Pelletsystems entstanden, besonders die einfache Überwachung durch einen Blick aufs Smartphone und die Erfassung der Verbräuche finde ich wirklich praktisch.
100% Genauigkeit ist nicht erreichbar, aber das war von vorneherein klar. Im Vergleich geben Hersteller von Pelletsheizkesseln mit einfacher Verbrauchsanzeige teils eine Genauigkeit von +/- 20% an – ich denke aus den bisherigen Erfahrungen sollte das gebaute System diesen Wert übertreffen.
Die Möglichkeit / Erforderlichkeit der Nachkalibrierung ist vielleicht ein bisschen lästig, trägt aber zu einer brauchbaren Genauigkeit bei.
Letztlich bietet das System deutlich mehr Informationen, als die käuflichen Systeme, die ich auf dem Markt gesehen habe, so daß sich das Projekt unterm Strich (abgesehen vom Spaßfaktor) gelohnt hat. ■

Teil 1: Grundlagen
Teil 2: elektrischer Aufbau
Teil 3: Programme

Anhang: Datenflussdiagramm
(erstellt mit yEd)