// Integrierte Heizungsthermostatsteuerung

// Autor: Looxer01 -------------------------

// Version 0.7 12.04.2017 - Initialversion

// Erweiterte Version 2.0 28.12.2017

// Autoren: Looxer01 - Apollon77 (Triggerkonzept)

//

//

// Erweiterte Funktionen im Vergleich zu 1.0

//- ExcludeHMSensors eingebaut - Damit lassen sich einzelne Sensoren ausschliessen ohne das Gewerk zu Aendern

//- wenn man "cron=0" setzt läuft das ganze Ding über Trigger. Vom Skript selbst erzeugte State-Changes werden aussortiert. Debug Log zeigt was passiert

//- Es werden für die eingestellten Planzeiten Schedules gesetzt, sodass Änderungen immer zum korrekten Zeitpunkt kommen. Das geht auch crongesteuert

//- Es gibt einen neuen Datenpunkt "Source\_NextTemp" der die nächste Temperatur enthält vom nächsten Schedule-Punkt. Noch nicht vollkommen korrekt weil Feuertage/Urlaub blöd sind. Muss ich nochmal ran. Aber im Notfall würde mir das reichen.

//- Es ist jetzt möglich eine manuell gesetzte Solltemperatur zum nächsten Schedulewechsel zurückzusetzen (statt Ablaufzeit in Minuten). Ablaufzeit in Minuten ist weiterhin möglich. Auch die Verhinderung von manuellen Temps ist möglich

//- viele Optimierungen und Bugfixes

//- Source\_last\_Program\_Run Datenpunkt auch pro Raum angelegt, generelles Source\_last\_Program\_Run wird immer geschrieben wenn etwas abgearbeitet wird

//- Es wird ein neuer View zur Verfügung gestellt. Der bisherige View kann aber auch weiterhin genutzt werden. (Austausch des Datenpunktes Source\_last\_Program\_Run auf Raumebenen ist empfehlenswert )

//

// Das Programm dient zur Steuerung von Heizungsthermostaten. (siehe Funktionsliste in der Doku)

// Es synchronisiert alle Thermostate eines Raumes - aehnlich zur Gruppenfunktion der Homematic

// Direktverknüpfungen bzw Gruppen werden unterstützt. Letztendlich funktionieren die Thermostate aber auch ohne DV und Gruppen, so als wären sie mit DV bzw Gruppen gesteuert

// Empfehlung: Modus des thermostates auf MANU setzen. "AUTO" ist zwar auchmoeglich, führt aber zur Erkennung einer manuellen Temperatur

//

// Ein Thermostat/Sensor darf nicht mehreren Raeumen zugeordnet sein

// Alle Sensoren und Thermostate muessen je einem Gewerke zugeordnet sein. Dieses Gewerk wird vom Heizungsscript ausgewertet

// Hinweis: bei aelteren ioBroker Installationen sind u.U. Raumdefinitionen in ioBroker inkonsistent was zu Fehlern im Programm fuehrt (siehe ioBroker Admin Tab: Aufzaehlungen)

// Falls das auftritt muessen die enum.rooms mit den Raeumen der CCU verglichen worden. Falsche Definitionen muessen aus iobroker manuell geloescht werden - Restart Adapter REGA erforderlich

// Falls es keine eigenen Definition in ioBroker gibt koennen auch einfach die enums gelöscht und durch REGA Neusynch neu angelegt werden

// Fuer nicht HM-Geräte: Die angegebenen Räume muessen in enums (Aufzählungen) von ioBroker vorhanden sein -bzw. können dort angelegt werden

//

// Vorbereitungen

// die relevanten Thermostate und Sensoren muessen dem angegebenen Gewerk zugeordnet sein

// Checken, dass alle Raeume bei den Aufzaehlungen mit der Raumliste der CCU uebereinstimmen

// Die Views muessen angelegt werden. Hierzu sind die folgenden Schritte notwendig

// a. Ersetzen der folgenden Variablen aus dem mitgelieferten View. Das muss je Raum und Profil wiederholt werden

// - Ersetze Schlafzimmer mit den eigenen jeweiligen Raumnamen entsprechend der eigenen Raumliste - Achtung Raumnamen duerfen keine Blanks enthalten - Blanks muessen mit "\_" aufgefuellt werden

// - Ersetze Profil-1 durch Profil-2 etc je anzulegendes Profil

// b. Importieren der so geaenderten views in VIS (ueber VIS Funktionen)

// c. Ordne die richtigen Datenpunkte fuer Solltemperatur und Isttemperatur zu

// ACHTUNG: Im View muss das Raumprofil aktiviert werden. Es ist anfangs nicht aktiviert damit nicht alle Raeume mit dem Standardprofil beim ersten Programmlauf aktiviert sind. Daher manuelle notwendig

// Import des Views geschicht über VIS - VIEWS - View imporiteren. Hier öffnet sich ein Fenster. Das Coding des Views muss dort hineinkopiert werden.

//

// Bitte die Dokumentation lesen - alle Konfigschritte und auch die Funktionsweise des Programmes sind dort erlaeutert

//

// ab hier ChangeLog // Aktuelle Version 2.0b01 (erste Beta) 28.12.2017

// Version 2.00b02 05.01.2018 - zweite Beta

//.............................Technische Coding Aenderungen (ueberfluessige log eintragungen und doppel coding entfernt)

//.............................Kein Trigger bei Aenderung von An/Abwesenheit und Feiertagen gefixt

//.............................Bei Einstellung der Duration Manuelle Temp kleiner Null wurde bei einer Thermostataenderung am Thermostat keine Rückstellung auf schedule vorgenommen

//.............................Delay Time (notwendig für alte Thermostate) nach Fensteröffnung wieder aktiviert - 2 Minuten Verzögerung nach Fensterschliessung

//.............................Sensorstatusermittlung fuer HM-Geraete verallgemeinert (keine Speziallogik mehr notwendig. Konfig in der Sensortypetab reicht aus) / logging Eintraege fuer Sensor Aenderungen hinzugefuegt

//.............................Bei Einschalten der Heizperiode wurden die Temperaturen nicht sofort auf die geplanten Temperaturen gesetzt

// Version 2.00b03 02.04.2018 - dritte Beta

//.............................Manuelle Temperaturen werden bei Scriptstart ignoriert/zurückgesetzt

//.............................Thermostabtypetab Position 4 auf Position 8 (nach den Wandthermosteten) verschoben

//.............................NoneHMTab - Fuellen der Position 12 in Controltab falsch (mit 0 ersetzt)

//.............................Bei gleichen Zeiten im schedule von verschiedenen Räumen kam es dazu, dass nicht geschaltet wurde. Eine Zeitverzögerung eingebaut

//.............................externe Dateiausgabe bei manuellen Aenderungen hinzugefügt (writelog)

//.............................Fehler in Routine Sensor Change bei direktvernuepften Fenstersensoren beseitigt.

//.............................Fehler bei den Subscriptions fuer Feiertage fuehrte zu Warnmeldungen, wenn kein Feiertagsadapter genutzt wurde

// Version 2.00a01 08.10.2018 - erste Alpha

//.............................Fehler bei nicht direkten Sensoren behoben: Temperatur wurde nicht abgesenkt

//.............................Raum-Statusanzeigen (Abweichungen vom Heizplan) wurde überarbeitet - keine Datenstrukturanpassung nötig- zentrale Routine eingefügt

//.............................Routine zur Überprüfung direktverknüpfter Sensoren nicht notwendig - entfernt

//.............................Voreinstellungen Parameter für Verschlusssensoren angepasst

//.............................Raumstatus (geoeffnet oder geschlossen ) Datenpunkt je Raum eingerichtet und Logik zum Raumstatus update fuer alle Sensoren des raumes abgebildet

//.............................Sensor aufgenommen: HMW-IO-12-Sw14-DR', 'Schließerkontakt HMW' , wired

// Version 2.00a02 14.10.2018 - zweite Alpha

//.............................Fehler bei nicht direkten Sensoren behoben: Temperatur wurde nicht abgesenkt / Code wieder aktiviert

// Version 2.00 26.10.2018 - erste Stable

//.............................bei direkt verknuepften Sensoren wurde bei geoeffnetem Fenster ein moeglicher Schedule Wechsel nicht berüksichtig

//Version 2.10 15.12.2018......Fehler behoben nach updae auf JS 4.0.1 !fs.existsSync funktioniert nicht mehr ausgetausch mit fs.readFile

//.............................WT HmiP-BWTH hinzugefügt

//.............................Subscription fuer ICAL Profile hinzugefügt

//.............................Delay für HM Geräte gefixt und für Nicht-HM Geräte hinzugefügt

//.............................ICAL Pfad umgestellt für ICAL Adapter Version 1.7.0

//.............................ICAL- Profile mit Selection Prio für die Profil-Selektion hizugefügt - unterstützt jetzt mehrere gleichzeitig aktive ICAL Events

//.............................Subscription für ICAL-Raumprofile und Selektion hinzugefügt (analog zu den globalen Profilen)

//.............................nach update auf JS 4.0.2 !fs.existsSync wieder aktiviert

//.............................SoftBoost Funktion hinzugefügt (analog dem HardBoost der Thermostate)

//.............................ICAL Events können jetzt in den Views aktiviert/deaktiviert werden

//.............................Manuelle Aenderungen stabilisiert

//Version 2.1.01 18.12.........Testraum erzeugte Fehler - Entfernt

//.............................Synchen bei nicht direkt verknüpften Thermostaten wieder hergestellt (bei Absenktemperatur)

//

//

//------------------------------------------------------------------------------

// Beginn Generelle Einstellungen

// werden an dieser Stelle benoetigt - Einstellungen sind nur in Spezialfaellen notwendig

//------------------------------------------------------------------------------

// Anpassung nur wenn unbedingt notwendig. Hier ist der Ansatz um z.B. Einliegerwohnungen separat zu steuern

var JSPath = "javascript.0."; // JS- Pfad

var path = JSPath + 'Heizung.Heizplan'; // Pfad fuer create states

var Gparameterpath = path + ".GlobaleParameter"; // Pfad in die Globalen Parameter

var ICALPath = "ical.0.events.0.now"; // Pfad zu den ICAL events zur Profilauswahl // ab ICAL Adapter version 1.7.0

// var ICALPath = "ical.0.events"; // Pfad zu den ICAL events zur Profilauswahl // bis ICAL Version kleiner als 1.7.0

//------------------------------------------------------------------------------

// Ende Generelle Einstellungen

// Usereinstellungen sind Einstellungen, die ueblicherweise gemacht werden

//------------------------------------------------------------------------------

//------------------------------------------------------------------------------

// Beginn USER Einstellungen

//------------------------------------------------------------------------------

// Gewerke - wichtige Einstellung, da nur die Geraete eingelesen werden, die im Gewerk vorhanden sind

// Das Gewerk muss alle Thermostate bzw Sensoren enhalten

var HeizungGewerk = "Heizung"; // diesem Gewerk muessen alle Thermostate zugeordnet sein.

var SensorGewerk = "Verschluss"; // diesem Gewerk muessen alle Verschlusssensoren zugeordnet sein.

// Alle x Minuten wird gecheckt ob die SollTemp angepasst werden muss - Empfehlung wenn cron dann 5

// Wenn Wert = 0 ist dann läuft das Skript über Events (empfohlener Weg)

var cron = 0;

// Raumliste - empfohlen zu benutzen fuer kleine Systeme

// UseRoomList heisst, dass die nur hier gelisteten Raeume angelegt und abgearbeitet werden - somit werden nicht sofort alle Datenpunkte aller Räume angelegt (ca. 100 pro Raum und Profil)

// dies dient hauptsaechlich zur Anlage der Daten - So koennen Raum fuer Raum alle States angelegt wrden

// Das sollte genutzt werden mit langsamen Rechnern wie Raspi mit SD karte

var UseRoomList = false; // Wenn testmodus werden nur die Angegebenen Raeume abgearbeitet

var RoomList = [];

RoomList[0] = ['Wohnzimmer']; // Liste der Raeume die gesteuert werden soll zum Testen

RoomList[1] = ['initial'];

RoomList[2] = ['Raum2'];

RoomList[3] = ['Raum3'];

RoomList[4] = ['Raum4'];

RoomList[5] = ['Raum5'];

RoomList[6] = ['Raum6'];

RoomList[7] = ['Raum7'];

RoomList[8] = ['Raum8'];

RoomList[9] = ['Raum9'];

// Liste der Auszuschliessenden Homematic Sensoren

// Falls ein Raum Sensoren hat die zu "Verschluss" gehören aber pot. nichts mit einem Öffnungszustand zu tun ChckAbsenkung

// kann man diese ausschliessen

var ExcludeHMSensors = [];

ExcludeHMSensors[0] = ''; // Liste des STATE-Datenpunkts (z.B. hm-rpc.0.NEQXXXXX.1.STATE) von Sensoren die nicht beachtet werden sollen

ExcludeHMSensors[1] = '';

// Anzahl der Profile. i.d.R. sollten maximal 3 Profile genuegen - Profile werden z.B. fuer Events aus ICAL verwendet

var MaxProfile = 2; // Maximal genutzte Profile pro Raum (gering halten ) Zahl zwischen 1 und 9

// Das ist die Temperatur, die eingestellt wird, wenn erkannt wird, dass ein Verschluss eines Raumes geoeffnet ist (z.B bei nicht direktverknuepften Geraeten)

var VerschlussAbsenkungsGrenze = 12;

// Soft-Boost Temperatur zum kurzfristigen Aufheizen eines Raumes - Sollte nicht hoeher sein als die High-Temp der Thermostate

var SoftBoostTemp = 30;

// erweitertetes Logging im ioBroker log bei true

var debug = true;

// Logging in externe Datei - Achtung der Pfad muss fuer MS-Windows bzw IOS angepasst werden

var LogFlag = false; // logging enabled

var LogPath = "/opt/iobroker/iobroker-data/HeizungsthermostatLOG.csv"; // Pfad und Dateiname des externen Logs

var OnlyChanges = false; // bei true wird nur geloggt wennn eine neue Solltemperatur geschrieben wird

// ICAL Einstellungen

// ICAL kann in den Views aktiviert werden - zuerst muss aber ICAL konfiguriert werden (Adapter ICAL Events) diehe dazu FAQs

// die Events muessen entsprechend in ICAL angelegt werden, sonst gibt es Warnmeldungen im Log

// Die Eventnamen koennen angepasst werden. Bitte die Logkik von ICAL unbeding beachten. (siehe Doku im Kapitel ICAL)

// var UseEventsGlobalParameter = false; //nicht mehr aktiv - kann gelöscht werden

// var UseEventsGlobalProfilSelect = true; //nicht mehr aktiv - kann gelöscht werden

// var UseEventsRaumProfilSelect = true; //nicht mehr aktiv - kann gelöscht werden

var EventG\_UrlaubAbwesend = "Urlaub\_Abwesend"; // dieses Event muss in ICAL angelegt werden wenn ICAL genutzt wird

var EventG\_UrlaubAnwesend = "Urlaub\_Anwesend"; // dieses Event muss in ICAL angelegt werden wenn ICAL genutzt wird

var EventG\_Party = "Party"; // dieses Event muss in ICAL angelegt werden wenn ICAL genutzt wird

var EventG\_Gaeste = "Gaeste"; // dieses Event muss in ICAL angelegt werden wenn ICAL genutzt wird

var EventG\_Abwesend = "Keiner\_DA"; // dieses Event muss in ICAL angelegt werden wenn ICAL genutzt wird

var EventG\_Feiertag = "Feiertag"; // dieses Event muss in ICAL angelegt werden wenn ICAL genutzt wird

// Die folgenden EVENT Texte muessen in ICAL angelegt werden. Sobald die Texte im google Kalender

// aktiv sind wird das Event fuer die Heizungsthermostatsteuerung ausgewertet.

// Achtung die Zeichen <> und der Text innerhalb dieser Klammer duerfen nicht geaendert werden

// siehe Dokumentation fuer mehr infos

var UseEventG\_Profil = "Profil\_<ProfilNummer>"; // Events mit denen das Profil umgeschaltet werden kann - muss in ICAL angelegt werden wenn UseEventP\_Profil = true ist

var UseEventR\_Profil = "<Raumname>\_<ProfilNummer>"; // Events mit denen das Raumprofil umgeschaltet werden kann - muss in ICAL angelegt werden wenn mit Eents gearbeitet wird

// folgende Beispiele, die in den ICAL Events des ICAL Adapters angelegt werden muessen

// Globale Events

// Urlaub\_Anwesend

// Party

// Globale Profil Events (fuer alle Räume gültig)

// Profil\_1

// Profil\_2

// Raumprofil Events (fuer einzelne Räume gültige Events)

// Schlafzimmer\_1

// Schlafzimmer\_2

// Kinderzimmer\_1

// Kinderzimmer\_2

// Integration zur Anwesenheitsermittlung -

var UseAnwesenheitserkennung = false; // wenn true, dann wird die o.g. Anwesenheitsvariable genutzt - Empfehlung erst im zweiten Schritt aktivieren

var StateAnwesenheitFunction = JSPath + "Anwesenheitssteuerung.Userlist.JemandDa"; // Wenn UseAnwesenheitserkennung = true, dann muss der Pfad angepasst werden

// Integration zum Feiertagskalender -

var UseFeiertagskalender = false; // wenn der Kalender genutzt wird bitte auf true setzen - Empfehlung: Feiertagsadapter installieren und auf true setzen

var StateFeiertagHeuteAdapter = "feiertage.0.heute.boolean"; // wenn UseFeiertagskalender, dann wird dieser Pfad verwendet

var StateFeiertagMorgenAdapter = "feiertage.0.morgen.boolean"; // wenn UseFeiertagskalender, dann wird dieser Pfad verwendet wenn es darum geht den nächsten Schaltpunkt zu ermitteln wenn dieser am nächsten Tag liegt

//------------------------------------------------------------------------------

// Ende USER Einstellungen

// Usereinstellungen sind Einstellungen, die ueblicherweise gemacht werden

//------------------------------------------------------------------------------

//------------------------------------------------------------------------------

// Beginn Experteneinstellungen

// Experteneinstellungen sollten nur geamcht werden, wenn die Logik des Programmes bekannt ist

//------------------------------------------------------------------------------

// Pfad zum Anwesenheitsflag der Hz-Steuerung - wird parallel zum Adapter gehalten

var StateAnwesenheit = JSPath + "Heizung.Heizplan.GlobaleParameter.Anwesenheit";

// Pfad zum Feiertagskennzeichen der Hz-Steuerung - wird parallel zum Adapter gehalten

var StateFeiertagHeute = JSPath +"Heizung.Heizplan.GlobaleParameter.Feiertag\_Heute";

// die States sollten moeglichst so belassen werden - das Programm laesst aber Aenderungen zu

var StatePartyjetzt = Gparameterpath + ".Partyjetzt"; // ID Party Jetzt flag

var StateGaesteDa = Gparameterpath + ".GaesteDa"; // ID Gaeste da flag

var StateUrlaubAnwesend = Gparameterpath + ".Urlaub\_Anwesend"; // Wenn kein Arbeitstag, dann wird der Tag wie ein Sonntag behandelt

var StateUrlaubAbwesenheit = Gparameterpath + ".Urlaub\_Abwesend"; // Temperaturabsenkung wenn laengerer Urlaub eingetragen ist

var StateHeizperiode = Gparameterpath + ".Heizperiode"; // Wenn Heizperiode false werden alle Ventile geschlossen

// Die ThermostatTypeTab definiert die Thermostat Typen.

// Achtung zentrale Steuerungen muessen immer zuerst eingetragen sein.

// Steuerung zentral heisst, dass dieses Geraet evt abhaengige Geraete steuert, wenn false, dann werden abhaengige Geraete gleich behandelt

// Wenn mit Direktverbindungen gearbeitet wird dann MUSS zentrale Steuerung auf true stehen

var ThermostatTypeTab = [];

// 0.RPC-Pfad 1.GeraeteType 2. Beschreibung, 3. Type 4.DP-SollTemp 5.nicht verwendet ID 6.DP MANU/AUTO Schaltung 7.Steuerung DV 8. IstTemp 9-Check-MANU-Mode 10-Ventilstellung wenn nicht Heizperiode 11. Delay nach Verschluss zu

ThermostatTypeTab[0] = ['hm-rpc.0.', 'HM-TC-IT-WM-W-EU', 'Wandthermostat (neu)' ,'WT', '2.SET\_TEMPERATURE' , false, '2.MANU\_MODE', true, '1.TEMPERATURE', '2.CONTROL\_MODE', 12, 0];

ThermostatTypeTab[1] = ['hm-rpc.0.', 'HM-CC-TC' , 'Wandthermostat (alt)' ,'WT', '2.SETPOINT' , false, false, false, '1.TEMPERATURE', false, 12, 0];

ThermostatTypeTab[2] = ['hm-rpc.0.', 'HM-CC-RT-DN' , 'Heizkoerperthermostat(neu)' ,'HT', '4.SET\_TEMPERATURE' , false, '4.MANU\_MODE', true, '4.ACTUAL\_TEMPERATURE', '4.CONTROL\_MODE', 12, 0];

ThermostatTypeTab[3] = ['hm-rpc.1.', 'HmIP-eTRV' , 'Heizkoerperthermostat(HMIP)','IPHT', '1.SET\_POINT\_TEMPERATURE', false, '1.CONTROL\_MODE', false, '1.ACTUAL\_TEMPERATURE', '1.CONTROL\_MODE', 12, 0];

ThermostatTypeTab[4] = ['hm-rpc.1.', 'HmIP-WTH' , 'Wandthermostat(HMIP)' ,'IPWT', '1.SET\_POINT\_TEMPERATURE', false, '1.CONTROL\_MODE', true, '1.ACTUAL\_TEMPERATURE', '1.CONTROL\_MODE', 12, 0];

ThermostatTypeTab[5] = ['hm-rpc.1.', 'HmIP-WTH-2' , 'Wandthermostat(HMIP)' ,'IPWT', '1.SET\_POINT\_TEMPERATURE', false, '1.CONTROL\_MODE', false, '1.ACTUAL\_TEMPERATURE', '1.CONTROL\_MODE', 12, 0];

ThermostatTypeTab[6] = ['hm-rpc.1.', 'HmIP-STH' , 'Wandthermostat(HMIP)' ,'IPWT', '1.SET\_POINT\_TEMPERATURE', false, '1.CONTROL\_MODE', true, '1.ACTUAL\_TEMPERATURE', '1.CONTROL\_MODE', 12, 0];

ThermostatTypeTab[7] = ['hm-rpc.1.', 'HmIP-STHD' , 'Wandthermostat(HMIP)' ,'IPWT', '1.SET\_POINT\_TEMPERATURE', false, '1.CONTROL\_MODE', true, '1.ACTUAL\_TEMPERATURE', '1.CONTROL\_MODE', 12, 0];

ThermostatTypeTab[8] = ['hm-rpc.1.', 'HmIP-eTRV-2' , 'Heizkoerperthermostat(HMIP)','IPHT', '1.SET\_POINT\_TEMPERATURE', false, '1.CONTROL\_MODE', false, '1.ACTUAL\_TEMPERATURE', '1.CONTROL\_MODE', 12, 0];

ThermostatTypeTab[9] = ['hm-rpc.2.', 'HmIP-eTRV-B' , 'Heizkoerperthermostat(HMIP)','IPHT', '1.SET\_POINT\_TEMPERATURE', false, '1.SET\_POINT\_MODE', true, '1.ACTUAL\_TEMPERATURE', '1.SET\_POINT\_MODE', 12, 0];

ThermostatTypeTab[10] = ['hm-rpc.1.', 'HmiP-BWTH' , 'Heizkoerperthermostat(HMIP)','IPWT', '1.SET\_POINT\_TEMPERATURE', true, '1.CONTROL\_MODE', false, '1.ACTUAL\_TEMPERATURE', '1.CONTROL\_MODE', 12, 0];

// Tabelle fuer Nicht HM Thermostate - Details finden sich in der Dokumentation

// wurde zum Testen verwendet, da auch virutelle Thermostate verwaltet werden koennen.

// Wenn nicht HM Geraete korrekt in ioBroker angebunden sind sollten diese auch ueber die Tabelle ThermostatTypeTab konfigurierbar sein

//Spalte 1 = Raumname wie in der CCU hinterlegt

//Spalte 2 = Erster Teil des Datenpunktpfades mit Instance wie z.B. "hm-rpc.0"

//Spalte 3 = Zweiter Teil des Datenpunktpfades mit der ID des Geraetes z.B. "MEQ0183268"

//Spalte 4 = Dritter Teil des Datenpunktpfades mit dem Datenpunkt der die Solltemperatur des Geraetes einstellt z.B. "4.SET\_TEMPERATUR"

//Spalte 5 = Delay nach schliessen des Raumes bis zum Start des nächsten Programmlaufes

//

var NoneHMTab = [];

// 0 = Raum 1 = Datenpunkt bis vor Geraet 2=Datenpunkt Geraet 3=Datenpunkt SollTemp 4= Ventilstellung bei NichtHeizperiode 5. Delay nach Verschluss zu

NoneHMTab[0] = ['initial', 'javascript.0.Heizung', 'zwave1', '4.SET\_TEMPERATURE', 12, 0];

NoneHMTab[1] = ['HZ\_Kilian', 'zwave.0', 'NODE2', 'THERMOSTAT\_SETPOINT.Heating\_1\_1', 12, 0];

NoneHMTab[2] = ['HZ\_Salon', 'zwave.0', 'NODE3', 'THERMOSTAT\_SETPOINT.Heating\_1\_1', 12, 0];

NoneHMTab[3] = ['HZ\_Bad', 'zwave.0', 'NODE4', 'THERMOSTAT\_SETPOINT.Heating\_1\_1', 12, ];

// Typen-Tabelle der Verschlusssensoren fuer Homematic Geräte

// 6 = Verschlussstatus = false ist gechlossen

var SensorTypeTab = [];

// 0.RPC-Pfad 1.GeraeteType 2. Beschreibung, 3.Type 4.DP Status 5.nicht verwendet 6. Verschlussstatus 7. direktverknuepft

SensorTypeTab[0] = ['hm-rpc.0.', 'HM-Sec-SCo' , 'Fenstersensor (neu)' , 'HM', '1.STATE' , false, false, true ];

SensorTypeTab[1] = ['hm-rpc.0.', 'HM-Sec-SC' , 'Fenstersensor (alt)' , 'HM', '1.STATE' , false, false, true ];

SensorTypeTab[2] = ['hm-rpc.0.', 'HM-Sec-RHS' , 'Fenster-Drehgriffkontakt', 'HM', '1.STATE' , false, 0, true ];

SensorTypeTab[3] = ['hm-rpc.0.', 'HM-Sec-SC-2', 'Fenstersensor-2 (alt)' , 'HM', '1.STATE' , false, false, true ];

SensorTypeTab[4] = ['hm-rpc.1.', 'HMIP-SWDO' , 'Fenstersensor (HMIP )' , 'IPSE', '1.STATE' , false, 0, true ];

SensorTypeTab[5] = ['hm-rpc.2.', 'HMW-Sen-SC-12-DR', 'Schließerkontakt HMW' , 'HM', '1.STATE' , false, false, false ];

SensorTypeTab[6] = ['hm-rpc.1.', 'HMW-IO-12-Sw14-DR', 'Schließerkontakt HMW' , 'HM', '1.STATE' , false, false, false ]; // wired

SensorTypeTab[7] = ['hm-rpc.1.', 'HmIP-SWDO-I' , 'Fenstersensor (HMIP )' , 'IPSE', '1.STATE' , false, 0, true ]; // IP innenliegender Sensor

SensorTypeTab[8] = ['hm-rpc.1.', 'HmIP-SWDM' , 'Fenstersensor (HMIP )' , 'IPSE', '1.STATE' , false, 0, true ]; // IP

// Tabelle der Verschlusssensoren fuer NichtHomematic Geräte

// 5 = wenn script die Absenktemperatur setzen soll, dann false

var NoneHMSenorTab = [];

// 0= Raum 1 = Datenpunkt vis vor Geraet 0.RPC-Pfad 2. Datenpunkt Geraet 3. Datenpunkt FensterstatusGeraeteType 4.Verschlussstatus bei geschlossen 5. TempAbsenkung automatisch

NoneHMSenorTab[0] = ['initial', 'javascript.0.Heizung.Heizplan', 'Wohnzimmer' , 'TestZusaetzlichesFenster' , false, false,];

NoneHMSenorTab[1] = ['initial', 'fhem.0', 'OG\_DU\_TF' , 'state' , 'closed', false];

NoneHMSenorTab[2] = ['initial', 'maxcube.0.devices', '', 'contact\_0a9d75', 'opened', false];

NoneHMSenorTab[3] = ['initial', 'javascript.0.Heizung', 'zwave1', 'isOpen', 'false', false];

NoneHMSenorTab[4] = ['initial', 'javascript.0.Heizung', 'zwave1', '4.Fenster', 'false', false];

// Mit der Tabelle OverruleTab kann die Logik der Temperaturanpassungen beeinflusst werden (Sobald eine Anpassung erfolgt wird der Vorgang Overrule abgebrochen)

// Die Tabelle kann als Prioritätenliste verstanden werden, wenn mehrere Parameter gleichzeitig zutreffen sollten.

// Die Logik wird fuer jeden Raum ausgefuehrt

var OverruleTab = [];

OverruleTab[0] = ["Abwesenheit"]; // Bei Abwesenheit wird die Temperatur der entsprechend Eisntellung abgesenkt

OverruleTab[1] = ["UrlaubAnwesend"]; // Urlaubsanwesenheit / beeinflusst nicht direkt die Solltemp - ist aber wichtig fuer die Schedule Findung

OverruleTab[2] = ["UrlaubAbwesend"]; // Urlaubsabwesenheit -

OverruleTab[3] = ["Gaeste"]; // Temperatur Anhebung

OverruleTab[4] = ["Party"]; // Partyabsenkung

// Bei Verwendung des Widgets Select-value List werden die Temperaturen nicht als Grad gespeichert

var VerwendungSelectValue = true;

// Weitere Pfade fuer die globalen Parameter - Empfehlung ist keine Aenderung vorzunehmen

var StatePP\_PartyAbsenkung = "ProfilParameter\_PartyAbsenkung";

var StatePP\_GaesteAnhebung = "ProfilParameter\_GaesteAnhebung";

var StatePP\_AbwesenheitAbsenkung = "ProfilParameter\_AbwesenheitAbsenkung";

var StatePP\_UrlaubAbsenkung = "ProfilParameter\_UrlaubAbsenkung";

var StatePP\_UrlaubWieFeiertag = "ProfilParameter\_UrlaubWieFeiertag";

var StatePP\_MinimaleTemperatur = "ProfilParameter\_MinimaleTemperatur";

// UserExit Einstellungen

// UserExits koennen genutzt werden, um die manuelle Temperatur von selbstdefinierten Abhängigkeiten zu steuern

// Beipiel Steuerung von ElektroKonvektoren abhängig vom Energieertrag einer PV

// Es koennen beliebig viele Eintragungen gemacht werden

//

// Die Datenpunkte werden nicht angelegt sondern muessen separat angelegt wreden

// UserEexitTab Tabellendefinition:

// 0 = Datenpunkt = Pfad Datenpunkt aufgrund dessen eine Reaktion erfolgen soll und Definition bei welchem Ereignis eine Reaktion erfolgen soll

// 1 = Name der Routine - Routine muss im Userexit definiert sein

// 2 = Operand - zulaessige Operanden sind

// groesser = "valGt"

// groesser gleich = "valGe"

// kleiner = "valLt"

// kleiner gleich = "valLe"

// gleich = "val"

// ungleich = "valNe"

// beliebiege Aenderung = "Any"

// 3 = Wert - Vergleichswert der die Routine auslöst (im Zusammenhang mit dem Operanden

// Sobald eine Bedingung zutrifft wird der UserExit aufgerufen. Die zugehoerige Routine wird zu anfang des UserExits ermittelt und kann dann weiterverarbeitet werden

// Rueckgabe des UserExits ist ein Raumname sowie eine manuelle Temperatur und Gueltigkeit in Minuten. Die Temperatur wird dann entsprechend mit der Gueltigkeit gesetzt

// ist die manuelle Temperatur = 0 wird eine evt. vorher eingestellte manuelle Temperatur gelöscht und die neue SollTemperatur wird anhand des schedules ermittelt

var UserExitTab = [];

// 0 = Datenpunkt 1= Routine 2=Operand 3= Vergleichswert

UserExitTab[0] = ['initial', 'TriggerHeatingOn', 'valGt', 100 ];

UserExitTab[1] = ['initial', 'TriggerHeatingOff', 'valLt', 0 ];

// es sind 5 globale Tabellen vordefiniert. Werden diese im UserExit befuellt bleiben die Werte erhalten für den nächsten Aufruf

var UserExitValueTab1 = [];

var UserExitValueTab2 = [];

var UserExitValueTab3 = [];

var UserExitValueTab4 = [];

var UserExitValueTab5 = [];

//------------------------------------------------------------------------------

// Ende Experteneinstellungen

//------------------------------------------------------------------------------

if (SoftBoostTemp === undefined) { // falls nicht definiert / abwärtskompatibilität

var SoftBoostTemp = 30;

}

// Variablendefinition

var ControlTab = []; // Zentrale Tabelle der Thermostate

var SensorList = []; // Liste der Verschlusssensoren

var DelayAfterClose = []; // Liste fuer Thermostate, die einen Delay nachdem die Verschluesse geschlossen werden, benötigen (z.B. alte HM Thermostate)

var fs = require('fs'); // enable write fuer externes log

var cronjob = "\*/" + cron + " \* \* \* \*"; // CRON pattern aufgrund der Vorgabe in den Einstellungen

var SubscribeThermBlock = []; // dieses Array dient dazu doppelte Ausführungen zu vermeiden, wenn die Solltemp geaendert wird, da Trigger auf den Solltemps liegen

// Diese Variablen sind ein Findungsnachweis fuer die gesetzte Temperatur - Sie werden am Ende im Raum gespeichert und dienen auch zur ermittlung von manuell eingestellten Temperatur

var Source\_Profil;

var Source\_ICALEvent;

var Source\_ManualAdjustment;

var Source\_GlobalParameter;

var Source\_SchedulePoint;

var Source\_LastTemp;

var Source\_Timestamp;

var Source\_CurrentSollTemp;

var Source\_NextSollTemp;

var Source\_last\_Program\_Run;

var Source\_ManTempSet;

// Object Deklarationen fuer Subscriptionsteuerungen (und flexiblen schedules)

var ManTimeouts = {};

var NextSchedules = {};

var ownStateChanges = {};

// Time ScriptStart

var TimeScriptStart = new Date().getTime(); // Aktuelle Zeit in Millisekunden seit 1970 wird benötigt um manuelle Temperaturen bei script start zu vermeiden

// Funktionsaufruf zur Einsteuerung der Subscriptions oder Schedule

initializeData();

//------------------------------------------------------------------------------

// Diese Funktion initialisiert alles

//------------------------------------------------------------------------------

function initializeData() {

// Devicetabellen aufbauen

getDevices(); // Liste der Thermostate generieren und States anlegen

setTimeout(function() { // Rest Initialisierung leicht verzögern damit States angelegt sind vorher

//-----------------------------------------------------------------------------------------------------

// Job zur Ausfuehrung der Kernfunktion (Temperatur Setting)

//-----------------------------------------------------------------------------------------------------

if (cron > 0) {

log("Heizungsscript verarbeitung benutzt Cron", "info");

schedule(cronjob, function() {

LoopRooms(); // Ablauflogik entlang der gefundenen Thermostate fuer alle Raeume

log("Heizungsscript verarbeitung Cron durchgelaufen", "info");

}); // Ende Job

}

//-----------------------------------------------------------------------------------------------------

// oder es läuft alles über Events und Trigger

//-----------------------------------------------------------------------------------------------------

else {

log("Heizungsscript verarbeitung benutzt Trigger und Events", "info");

var subscribeIdList = [

StateFeiertagHeuteAdapter,

StateAnwesenheitFunction,

StateHeizperiode

];

if (UseFeiertagskalender ) {

subscribeIdList.push(StateFeiertagHeuteAdapter);

}

if (UseFeiertagskalender === false) {

subscribeIdList.push(ICALPath + "."+ EventG\_Feiertag);

}

// Globale Parameter

if (getState(Gparameterpath + ".ICAL-Events\_Aktiv").val === true) {

subscribeIdList.push(ICALPath + "."+ EventG\_Party);

subscribeIdList.push(ICALPath + "."+ EventG\_UrlaubAbwesend);

subscribeIdList.push(ICALPath + "."+ EventG\_UrlaubAnwesend);

subscribeIdList.push(ICALPath + "."+ EventG\_Gaeste);

}

// ICAL Profil Subscriptions

var ProfilName;

if (getState(Gparameterpath + ".ICAL-Events\_Aktiv").val === true) {

for (var i = 1; i <= MaxProfile; i++) {

ProfilName = UseEventG\_Profil.replace("<ProfilNummer>", i);

subscribeIdList.push(ICALPath + "." + ProfilName);

}

// ICAL Profile je Raum

for (var roomName in rooms) {

for (var i = 1; i <= MaxProfile; i++) {

ProfilName = UseEventR\_Profil;

ProfilName = UseEventR\_Profil.replace("<Raumname>", roomName);

ProfilName = ProfilName.replace("<ProfilNummer>", i);

subscribeIdList.push(ICALPath + "." + ProfilName);

}

}

}

// Auf Änderungen einiger spezieller States reagieren

on({id: subscribeIdList, change: 'ne'}, function(state) {

if (debug) { log("Trigger nach Änderung für State " + state.id, "info");}

TriggerUpdate();

});

// Auf Änderungen der Globalen Parameter reagieren

on({id: new RegExp("^" + Gparameterpath .replace(/[-[\]{}()\*+?.,\\^$|#\s]/g, '\\$&') + '\\.'), change: 'ne'}, function(state) {

if (ownStateChanges[state.id]) {

ownStateChanges[state.id]--;

if (ownStateChanges[state.id] === 0) delete ownStateChanges[state.id];

if (debug) {log("Ignoriere Trigger nach Änderung für State " + state.id, "info");}

return;

}

if (debug) {log("Trigger nach Änderung für State " + state.id, "info");}

TriggerUpdate();

});

// Pro Raum auf Änderungen registrieren

for (var roomName in rooms) {

roomName = roomName.replace(/\s/g, "\_");

on({id: new RegExp("^" + path.replace(/[-[\]{}()\*+?.,\\^$|#\s]/g, '\\$&') + '\\.' + roomName + '\\.'), change: 'ne'}, function(state) {

var roomName = state.id.substring(path.length + 1);

roomName = roomName.substring(0, roomName.indexOf('.'));

if (ownStateChanges[state.id]) {

ownStateChanges[state.id]--;

if (ownStateChanges[state.id] === 0) delete ownStateChanges[state.id];

if (debug) {log("Ignoriere Trigger nach Änderung für State " + state.id, "info");}

return;

}

if (debug) {log("Trigger nach Änderung für State " + state.id + " und Raum " + roomName, "info");}

TriggerUpdate(roomName);

});

}

}

LoopRooms();

}, 2000);

// Book Subsriptions

subscribeUserTab();

}

//-----------------------------------------------------------------------------------------------------

// Diese Funktion wird bei jeder registrierten Änderung aufgerufen und sorgt für

// eine Abarbeitung des/der Räume mit kleiner Verzögerung falls mehrere Werte nacheinander geändert

// wurden, sodass die Logik nur einmalig ausgeführt wird.

// Parameter: Raumname wenn bekannt oder "undefined" und Flag für manuelle Änderungen

//-----------------------------------------------------------------------------------------------------

var roomUpdateDelay = {}; // globale Variable

function TriggerUpdate(room, manChange) {

if(debug ) { log("Routine TriggerUpdate: TriggerUpdate gestartet", "info"); }

if (manChange === undefined) manChange = false;

if (manChange && room && ManTimeouts[room]) { // Funktion wurde durch manuellen Timeout getriggert, zurücksetzen

ManTimeouts[room] = null;

}

var delayRoom = room;

if (!room) { // Kein Raum gesetzt

delayRoom = 'all';

}

else if (roomUpdateDelay.all) { // Raum gesetzt aber ein Timeout für alle räume läuft schon, also bleibt es dabei

if (debug) { log("Sonderfall Raum war " + room + " ... reset to all", "info"); }

room = undefined;

delayRoom = 'all';

}

if (roomUpdateDelay[delayRoom]) { // Wenn für aktuellen Raum ein Timeout gesetzt ist dann beendedn

clearTimeout(roomUpdateDelay[delayRoom]);

roomUpdateDelay[delayRoom] = null;

}

if (!room) { // Falls kein Raum angegeben wurde

for (var roomName in roomUpdateDelay) { // Alle Tneouts auch beenden, muss ja nicht mehrfach laufen

if (roomUpdateDelay[roomName]) {

clearTimeout(roomUpdateDelay[roomName]);

roomUpdateDelay[roomName] = null;

}

}

}

var delayForUpdate = 1000;

if (debug) { log("TriggerUpdate für " + room + " und manChange=" + manChange, "info"); }

if (!room) {

delayForUpdate = 5000;

}

roomUpdateDelay[delayRoom] = setTimeout(function () {

roomUpdateDelay[delayRoom] = null;

LoopRooms(room); // Ablauflogik entlang der gefundenen Thermostate fuer alle Raeume

log("Heizungsscript verarbeitung Trigger für Raum " + delayRoom + " durchgelaufen", "info");

}, delayForUpdate);

}

//-----------------------------------------------------------------------------------------------------

// Wird verwendet um State-Werte zu setzen, merkt sich wenn ein eigener State geändert wurde,

// um das später ignorieren zu können

//-----------------------------------------------------------------------------------------------------

function setOwnState(stateId, val) {

if (stateId.indexOf(path) !== -1 && cron === 0) {

if (! ownStateChanges[stateId]) ownStateChanges[stateId] = 1;

else ownStateChanges[stateId]++;

}

setState(stateId, val);

}

//-----------------------------------------------------------------------------------------------------

// Funktion SubscribeUserTab

// Subscriptions auf UserExit States

//-----------------------------------------------------------------------------------------------------

function subscribeUserTab() {

for (var x in UserExitTab ) {

if (UserExitTab[x][0] === "initial" ) {

continue;

}

var subscribeObj = {id: UserExitTab[x][0]};

switch (UserExitTab[x][2]) {

case "valNe":

case "valGt":

case "valGe":

case "valLt":

case "valLe":

case "val": subscribeObj[UserExitTab[x][2]] = UserExitTab[x][3];

break;

case "any": subscribeObj.change = "any";

break;

}

on(subscribeObj, function(obj) {

UserExitPrep(obj.id, obj.state.val);

}); // ende on id

} // Endfor

}

//-----------------------------------------------------------------------------------------------------

// Funktion UserExit-

// Es stehen 5 globale Tabellen zur Verfügung Die Werte dieser Tabellen gehen nicht verloren für den nächsten aufruf

// UserExitValueTab1

// UserExitValueTab2

// UserExitValueTab3

// UserExitValueTab4

// UserExitValueTab5

// Rückgabe:

// USerExitCallBack(Raum,Solltemperatur,Gueltigkeit)

//-----------------------------------------------------------------------------------------------------

function UserExit (id, value, routine) {

if (routine === "TriggerHeatingOn") {

log("Routine UserExit UserExit aufgerufen " + id + " " + value + " " + routine);

}

} // endfunction

//-----------------------------------------------------------------------------------------------------

//ab hier Haupt-Routinen

// 1. GetDevices --einlesen bei jedem Programm start -nicht schedule

// 2. Loop Rooms --wird ausgeführt durch den schedule

// 3. LoopDevices --wird durch LoopRooms ausgeführt

// 5. ThermostatChange -- bei jeder manuellen Aenderung des Thermostats wird diese Routine aufgerufen

// 6. SensorChange -- wird ausgeführt wenn ein Verschluss geoeffnet/geschlossen wird

//-----------------------------------------------------------------------------------------------------

//-----------------------------------------------------------------------------------------------------

// Funktion getDevices - Zentrale Funktionen zum Lesen von enum und objekt

//-----------------------------------------------------------------------------------------------------

function getDevices() {

// jetzt alle zugehoerigen Thermostate finden und in Crontab speichern

var roomName;

var roomNoSpace;

var idExtract;

var fullname;

var devtype;

var FinishRoom = false;

var y = 0;

var SelectorVerschluss;

var SelectorThermostat;

var hmprc;

var devTypeThermTab;

var sensortype;

var StateDP;

var x;

var hmrpc;

// Auslesen aller Raeume

var allRooms = getEnums('rooms'); // Lade alle Raeume

rooms = {};

// jetzt Thermostate einlesen

// Lade alle Raeume

for (var i in allRooms) { // loop ueber alle Raeume

roomName = allRooms[i].name;

if (!RoomListUsage(roomName)) continue; // wenn die Raumliste genutzt wird ist und der Raum in der Liste enthalten ist

if (debug) { log("Gibt es ein zugeordnetes Gerät für den Raum " + roomName + " wird jetzt ueberprueft", "info"); }

roomNoSpace = roomName.replace(/\s/g, "\_");

for (x in ThermostatTypeTab){ // loop ueber die moeglichen Thermostattypen

hmrpc = ThermostatTypeTab[x][0];

devTypeThermTab = ThermostatTypeTab[x][1].toUpperCase();

StateDP = ThermostatTypeTab[x][4];

$('channel[state.id=\*.' + ThermostatTypeTab[x][4] + '] (rooms=' + roomName + ') (functions=' + HeizungGewerk + ') ').each(function (id, i) {

idExtract = id.substr(0,id.length - StateDP.length - 1);

fullname = getObject(id).common.name;

devtype = getObject(idExtract).native.TYPE.toUpperCase();

//if (devtype.includes(devTypeThermTab)) { // hier wirden nur ähnlichkeiten verglichen. Das könnte zu Fehlern führen.

if(devtype ===devTypeThermTab) {

SelectorThermostat = $('channel[state.id=' + id + '] ');

SelectorThermostat.each(function (id, i) {

on({id: id, change: 'any'}, function(obj) { // erstelle subscription

//if (obj.state.ack) {

if (obj.state.val !== obj.oldState.val) {

ThermostatChange(id);

}

}); // endon

}); // endeach

// 0 = roomName 1 = id 2 = devtype 3 = FullName 4 ID Extract 5 = Type des Geraetes 6 = DP SollTemp 7 = Manu/Auto 8 = Steuerung zentral 9 = DP Ist-Temperatur 10 = DP MANU Check 11=Ventil Offen 12= Delay nach Verschluss zu

ControlTab[y] = [roomNoSpace, id, devtype, fullname, idExtract, ThermostatTypeTab[x][3], ThermostatTypeTab[x][4], ThermostatTypeTab[x][6], ThermostatTypeTab[x][7], ThermostatTypeTab[x][8], ThermostatTypeTab[x][9], ThermostatTypeTab[x][10], ThermostatTypeTab[x][11]];

y++;

rooms[roomName] = true;

if (debug) {log("Routine getdevice fuer " + roomNoSpace,"info");}

CreateStates(roomName, devtype); // Lege die states an

if (ThermostatTypeTab[x][7] === true) { // Das ist das zentrale Steuerthermostat

FinishRoom = true; // gehe zum naechsten Raum

// return; // das steuernde Thermostat wurde gefunden ensprechend prioritaet der ThermostatTypeTab

}

log("Routine GetDevices fuer HM Thermostate " + roomName +" - " + ThermostatTypeTab[x], "info");

}

}); // end Channel loop

if (FinishRoom) {

FinishRoom = false;

break;

} // gehe zum naechsten Raum

} // End ThermostatTypeTab

} // End rooms - Homematic Geraete bzw voll eingebundene Geraete

// Fuellen der Nicht-Homematic Geraete in die ControlTab

y = ControlTab.length;

var id;

for (var b in NoneHMTab) {

roomName = NoneHMTab[b][0];

if (roomName !== "initial" && RoomListUsage(roomName)) {

id = NoneHMTab[b][1] + '.' + NoneHMTab[b][2] + '.' + NoneHMTab[b][3];

var NoneHMObj = getObject(id);

if (NoneHMObj && NoneHMObj.common && NoneHMObj.common.name) {

fullname = NoneHMObj.common.name;

}

else {

fullname = "n/a";

}

devtype = "NoneHM";

if (debug ) {log("Routine getdevice fuer " + roomName, "info");}

CreateStates(roomName, devtype); // Lege die states an

roomNoSpace = roomName.replace(/\s/g, "\_");

if(NoneHMTab[b][5] === undefined) { // für den Fall, dass in der Tabelle kein Wert eingetragen wurde - wird nur zur kompatibilität zu den alten Einstellungen gebraucht

NoneHMTab[b][5] = 0;

}

// 0 = room 1 = id 2 = devtype 3 = FullName 4 ID Extract 5 = Type des Geraetes 6 = DP SollTemp 7 = Manu/Auto 8 = Steuerung zentral 9 = DP Ist-Temperatur 10 = DP MANU Check 11=Ventil Offen - 12= Delay nach Verschluss zu

ControlTab[y] = [roomNoSpace, id, devtype, fullname, NoneHMTab[b][2], 'HT', NoneHMTab[b][3], false, false, false, false, NoneHMTab[b][4], NoneHMTab[b][5] ];

y++;

// subscription für Aenderung auf Aenderung der Raumtemperatur des Thermostates zu reagieren

on({id: id, change: 'ne'}, function(obj) {

//if (obj.state.val !== obj.oldState.val) { // Nur bei Aenderungen

ThermostatChange(obj.id);

//} // End - react on change

}); // ende on id

log("Routine GetDevices fuer NoneHM Thermostate " + roomName +" - " + NoneHMTab[b], "info");

rooms[roomName] = true;

} // endif roomName war nicht initial

} // endfor Nicht Homematic Geraete

// jetzt die Control Tab Sortieren nach Raumnamen

ControlTab.sort(SortControlTab);

if (debug) {

log("Liste der Thermostate in der Control Tabelle", "info");

for (var c in ControlTab) {

log(ControlTab[c],"info");

}

}

// Verschluss sensoren einlesen und subscription buchen

// SensorTypeTab[5] = ['hm-rpc.1.', 'HMIP-SWDO' , 'Fenstersensor (HMIP )' , 'IPSE', '1.STATE' , 14, false, true];

// right string id.substr(-SensorTypeTab[x][5].length);

y = 0;

for (roomName in rooms) { // suche Sensoren in allen Räumen mit mind einem Kontroll-Gerät

roomNoSpace = roomName.replace(/\s/g, "\_");

for (x in SensorTypeTab){

hmrpc = SensorTypeTab[x][0];

sensortype = SensorTypeTab[x][1];

StateDP = SensorTypeTab[x][4];

$('channel[state.id=\*.' + SensorTypeTab[x][4] + '] (rooms=' + roomName + ') (functions=' + SensorGewerk + ') ').each(function (id, i) {

if (ExcludeHMSensors.indexOf(id) !== -1) return; // ID steht auf der Exclsude-Liste

idExtract = id.substr(0, id.length - StateDP.length - 1);

fullname = getObject(id).common.name;

devtype = getObject(idExtract).native.TYPE;

if (devtype === sensortype ) {

SensorList[y] = [roomNoSpace, id, devtype, fullname, idExtract, SensorTypeTab[x][3], SensorTypeTab[x][4], SensorTypeTab[x][5], SensorTypeTab[x][6], SensorTypeTab[x][7] ];

SensorList[y][7] = getState(SensorList[y][1]).val; // Status des Sensors

log("Routine GetDevices fuer HM Sensoren "+ roomName +" - " + SensorList[y], "info");

y++;

// SelectorVerschluss = $('channel[state.id='+id+'] (rooms='+roomName+') (functions='+SensorGewerk+') ');

SelectorVerschluss = $('channel[state.id='+id+'] ');

SelectorVerschluss.on(function(obj) { // bei Zustandaenderung

if (obj.state.ack && obj.state.val !== obj.oldState.val) {

SensorChange(id);

}

}); // endon

}

}); // end Channel loop

} // Endfor SensorTypeTab

} // Endfor rooms Verschlusssensoren

// Fuellen der Nicht-Homematic Sensoren in die SensorTab

// Map-Dokumentation:

// Sensortypetab = 0.RPC-Pfad 1.GeraeteType 2. Beschreibung, 3.Type 4.DP Status 5.Laenge ID 6. Verschlussstatus 7. direktverknuepft

// NonHM-SensorTab 0= Raum 1 = Datenpunkt vis vor Geraet 0.RPC-Pfad 2. Datenpunkt Geraet 3. Datenpunkt FensterstatusGeraeteType 4.Verschlussstatus bei geschlossen 5. TempAbsenkung automatisch,

//

// Sensorlist SensorList[y] = [roomNoSpace, id, devtype, fullname, idExtract, SensorTypeTab[x][3],SensorTypeTab[x][4],SensorTypeTab[x][5],SensorTypeTab[x][6],SensorTypeTab[x][7] ];

// Füllen der sensorlist raum=0 komplette id=1 NoneHM=2 Name aus getobject(id)=3 NoneHMSenorTab(1)=4 "NoneHM"=5 NoneHMSenorTab(2)=6 laenge ID=7 verschlussstatus=8 direktverknüpft=9 ,

y = SensorList.length; // letzter Eintrag der Sensorlist

for (x in NoneHMSenorTab) {

roomName = NoneHMSenorTab[x][0];

if(roomName !== "initial" && RoomListUsage(roomName)) { // wenn die Raumliste genutzt wird ist und der Raum in der Liste enthalten ist.

y++;

roomNoSpace = roomName.replace(/\s/g, "\_");

id = NoneHMSenorTab[x][1] + '.' + NoneHMSenorTab[x][2] + '.' + NoneHMSenorTab[x][3];

var NoneHMSensorObj = getObject(id);

if (NoneHMSensorObj && NoneHMSensorObj.common && NoneHMSensorObj.common.name) {

fullname = NoneHMSensorObj.common.name;

}

else {

fullname = "n/a";

}

devtype = "NoneHM";

if (debug ) {log("Routine getdevice fuer NoneHMSenorTab " + roomName,"info");}

//Füllen: raum=0 komplette id=1 NoneHM=2 Name aus getobject(id)=3 Datenpunkt bis vor Geraet=4 "NoneHM"=5 Beschreibung=6 frei=7 verschlussstatuss, direktverknuepft=9

SensorList[y] = [roomNoSpace, id, devtype, fullname, NoneHMSenorTab[x][1], NoneHMSenorTab[x][3], fullname, false, getState(id).val, NoneHMSenorTab[x][5] ];

log("Routine GetDevices NoneHM Sensoren: " + roomNoSpace + " - " + SensorList[y], "info");

on({id: SensorList[y][1], change: 'ne'}, function(obj) {

//if (obj.state.val !== obj.oldState.val) { // nur bei Aenderungen

SensorChange(obj.id);

//} // End - react on change

}); // ende on id

} // endif roomName war nicht initial

} // endfor Nicht Homematic Geraete

// Ausgabe der Devices - Thermostate und Sensoren

if (debug) {

for (var d in rooms){

log ("Liste der relevanten Räume " + d, "info");

}

} // ende debug

// Raumstatus ermitteln über alle Sensoren für jeden Raum

var Statuscheck = false;

for (roomName in rooms) { // loop ueber all Raeume

Statuscheck = false;

roomName = roomName.replace(/\s/g, "\_"); // Blanks durch unterstrich ersetzen

for (var x in SensorList ) { // loop über all Sensoren des Raumes

if (SensorList[x][0] === roomName) {

if(SensorStatCalc(SensorList[x][1], SensorList[x][2] ) ) {

Statuscheck = true; // wenn einer true dann alle true

break; // nächster Raum

} else {

Statuscheck = false;

} //SensorStatCalc - Ermitlung status des Sensors

} // endif nur sensoren des Raumes

} // endfor Sensorlist

if(Statuscheck === false) {

setOwnState(path + "." + roomName + ".RaumStatusVerschluss", false); // Raum ist geschlossen

log ("Raum "+ roomName + " Status geschlossen")

} else {

setOwnState(path + "." + roomName + ".RaumStatusVerschluss", true); // Raum ist geöffnet

log ("Raum "+ roomName + " Status geöffnet")

} // endif Statuscheck

} // endfor rooms

// jetzt noch evt fruehere manuelle Temps zurücksetzen

for (roomName in rooms) { // loop ueber all Raeume

roomName = roomName.replace(/\s/g, "\_");

if (getState(path + "." + roomName + ".SoftBoostActive").val ) { // SoftBoot Aktive

setOwnState(path + "." + roomName + ".SoftBoostActive", false); // Initilize

}

if ( getState(path + "." + roomName + ".SoftBoostValidity").val !== "init" ) {// SoftBoost Gueltigkeit bis - oder init

setOwnState(path + "." + roomName + ".SoftBoostValidity", "init"); // Initilize

}

InitilizeManChange(roomName); // Manuelle Aenderungen zuruecksetzen

}

log("Routine GetDevices Devices initialisiert","info");

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion LoopRooms - Abarbeiten der Raeume

//-----------------------------------------------------------------------------------------------------

function LoopRooms(room) {

var roomName;

var globalChange = false;

// Feiertagsflag synchronisieren

if (UseFeiertagskalender) { // Feiertagskalender Adapter aktiv ?

if (getState(StateFeiertagHeuteAdapter).val !== getState(StateFeiertagHeute).val ) { // Feiertagsflag aktualisieren

setState(StateFeiertagHeute, getState(StateFeiertagHeuteAdapter).val);

globalChange = true;

}

}

// Anwesenheitsflag synchronisieren

if (UseAnwesenheitserkennung) {

if (getState(StateAnwesenheitFunction).val !== getState(StateAnwesenheit).val ) { // Feiertagsflag setzen

setState(StateAnwesenheit, getState(StateAnwesenheitFunction).val);

globalChange = true;

}

}

Source\_last\_Program\_Run = formatDate(new Date(),"DD/MM/JJJJ SS:mm:ss");

setOwnState(Gparameterpath + ".Source\_last\_Program\_Run", Source\_last\_Program\_Run);

SetEventGlobalParameter(); // checken of ICAL events als global parameter vorliegen

for (roomName in rooms) { // loop ueber all Raeume

roomName = roomName.replace(/\s/g, "\_"); // Blanks durch unterstrich ersetzen

if (room && room !== roomName) continue; // Wenn für einen Raum getriggert müssen wir nur den Raum abarbeiten

if (getState(path + "." + roomName + ".AktivesRaumProfil").val > 0) {

if (debug) { log("Routine LoopRooms Starte Abarbeitung fuer Raum " + roomName, "info"); }

ClearSources();

// SetEventGlobalParameter(); // checken of ICAL events als global parameter vorliegen

LoopDevices(roomName);

setOwnState(path + "." + roomName + ".Source\_last\_Program\_Run", Source\_last\_Program\_Run);

if (debug) { log("Routine LoopRooms Ende Abarbeitung fuer Raum " + roomName, "info"); }

if (debug) { log("","info"); }

}

} // Endfor rooms

if (cron === 0) {

setTimeout(function() {

if (Object.keys(ownStateChanges).length > 0) {

if (debug) {

log("Restliche Triggered States resetten: " + JSON.stringify(ownStateChanges), "info");

}

}

ownStateChanges = {};

}, 10000);

}

} // endfunction

//-----------------------------------------------------------------------------------------------------

// Funktion LoopDevices - Abarbeiten der Thermostate

//-----------------------------------------------------------------------------------------------------

function LoopDevices(roomName) {

// var ActiveRoomProfile;

var ScheduledSollTemp;

var ManAdjDetected;

var ThermMode;

var ManAdj;

var ManAdjTimeStamp = getState(path + "." + roomName +".Source\_TimeStamp").val;

var ThermoType; // zum checken um welchen Thermostatypen es hier handelt aus der ControlTab(5)

var idExtract;

var deviceType;

var id;

var fullname;

var Sensor;

if(debug) { log("loop Devices gestarted fuer Raum "+roomName, "info"); }

Source\_ManTempSet = null; // Variable zurücksetzen - wird gebraucht fuer das synchen von Thermostaten

for (var x in ControlTab) {

if (ControlTab[x][0] !== roomName) continue;

Source\_CurrentSollTemp = 0;

Source\_NextSollTemp = 0;

Source\_SchedulePoint = "";

Source\_GlobalParameter = "";

idExtract = ControlTab[x][4];

deviceType = ControlTab[x][2];

id = ControlTab[x][1];

fullname = getObject(ControlTab[x][1]).common.name;

ThermoType = ControlTab[x][5];

// Setzen des Thermostates in den manuellen Modus - wenn moeglich / eingestellt

if (ControlTab[x][7] !== false && getState(path + "." + roomName + "." + "RaumParameter\_ManuellModeForce").val === true ) { // Geraet laesst sich auf MANU Schalten - im raumparameter steht auch, dass geschaltet werden soll

ThermMode = getState(idExtract + "." + ControlTab[x][10]).val; // Feststellen ob das Thermostat auf Auto oder MANU steht

// Check CC und DN Thermostate (nicht IP)

if ((ThermoType === "WT" || ThermoType === "HT") && (ThermMode === 0 || ThermMode === 2)) { // Pruefen ob der manuelle Modus oder Party Mode eingeschaltet ist fuer CC and DN Thermostate

log("Routine LoopDevices: Geraet " + idExtract +" Raum: " + roomName + " in den Manuellen Modus gesetzt ", "info");

setOwnState(idExtract + "." + ControlTab[x][7], getState(id).val); // setzen auf manuell - native boost und party werden nicht beachtet

writelog(roomName, id, "Thermostat in den manuellen Modus versetzt");

} // endeif automode war eingeschaltet

// check IP Thermostate

if ((ThermoType === "IPHT" || ThermoType === "IPWT") && (ThermMode === 0 || ThermMode === 2)) { // Pruefen ob der manuelle Modus oder Party Mode eingeschaltet ist fuer CC and DN Thermostate

log("Routine LoopDevices: Geraet " + idExtract +" Raum: " + roomName + " in den Manuellen Modus gesetzt ", "info");

log("Thermotype ist " + ThermoType + " ThermMode ist "+ ThermMode, "info");

setOwnState(idExtract + "." + ControlTab[x][7], 1); // setzen auf manuell - native boost und party werden nicht beachtet

writelog(roomName, id, "Thermostat in den manuellen Modus versetzt");

} // endeif automode war eingeschaltet

} // endif pruefe ob das Geraet in den manuellen Modus geschickt werden kann und soll

// Handling einer evt manuellen Temperatur im Regler oder im view eingestellt

ManAdj = ManAdjustments(roomName, id); // Finden ob es man adjustments gibt

if (ManAdj === false ) { // es gibt keine manuelle Anpassung

if(debug) { log( "Routine Loop Devices: es soll eine Temp-Findung durchgeführt werden ","info"); }

SaveStatus("Heizplan", roomName, false); // Status Meldung evt zurücksetzen

ExecuteTempDetermination(roomName, id);

writelog(roomName, id, "Temperatur wird nach Schedule eingestellt");

} // endif keine manuellen Adjustments

else { // es liegt eine manuelle Korrektur vor

if(debug) { log( "Routine Loop Devices: es soll keine Temp-Findung durchgeführt werden ","info"); }

SaveStatus("Manuell", roomName,false); // Meldung ausgeben dass eine manuelle Temp vorliegt

writelog(roomName, id, "");

if (Source\_ManTempSet === null) {

return; // es gibt eine manuelle Temperatur

}

if (Check\_ThermDV(roomName) ) { // sind die Thermostate direktverknuepft ? true heisst nein

if (debug) {log("Routine LoopDevices: Starte Sync fuer Manuelle Temperatur fuer " + id, "info"); }

SyncThermostat(roomName, "ManualTemp", Source\_ManTempSet, id); // jetzt manuelle Temp synchen

return;

}

} // endif Manuelle Anpassung erkannt

} // End for

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion ThermostatChange erkennt die Veraenderung der Solltemperatur durch subsription

//-----------------------------------------------------------------------------------------------------

function ThermostatChange(id) {

var ManAdj;

var room;

var ActTime = formatDate(new Date(),"YYYY/MM/DD SS:mm:ss"); // Formatierte Aktuelle Zeit

var ActTimeMilliSek = new Date(ActTime).getTime(); // Aktuelle Zeit in Millisekunden seit 1970

var LastRoomUpdate;

var source\_ManTempSet;

Source\_CurrentSollTemp = 0; // Ruecksetzen der current SollTemp fuer nachfolgende Programme

for (var x in ControlTab ) {

if (ControlTab[x][1] === id ) {

room = ControlTab[x][0];

LastRoomUpdate = ActTimeMilliSek - LastRoomUpdateTime(room, "find"); // Differenzzeit aus dieser Zeit und der letzten updatezeit des Raumes (Vermeidung von eigenen Triggern wenn die Solltemp geaendert wird)

if ( LastRoomUpdate < 1000) { // der Raum wurde vor weniger als 1 Sekunde bereits upgedated

if (debug) { log("Routine ThermostatChange: Der Raum wurde vor weniger als 1 Sekunde bereits upgedated Aenderung wird ignoriert " + LastRoomUpdate, "info"); }

return;

}

writelog(room, id, "Am Thermostat wurde eine neue Soll-Temperatur erkannt " + getState(id).val);

log("Routine ThermostatChange: " + id + " Raum " + room + " Solltemperatur-Aenderung erkannt auf " + getState(id).val , "info");

source\_ManTempSet = getState(id).val;

if (source\_ManTempSet !== VerschlussAbsenkungsGrenze) {

writelog(room, id, "Routine ThermostatChange: Thermostat " + id + " Raum " + room + " Thermostat Solltemperatur-Aenderung erkannt auf " + source\_ManTempSet) ;

LoopDevices(room);

} else {

writelog(room, id, "Routine ThermostatChange: Temperaturabsenkung erkannt" + id + " Raum " + room + " auf: " + source\_ManTempSet ) ;

}

if (Source\_ManTempSet === null) {

log("Routine ThermostatChange: ACHTUNG: ThermostatChange in Temperatur Source\_ManTempSet hat keinen Wert - kann nicht synchen")

return;

}

if (debug) { log("Routine ThermostatChange: Starte Sync fuer Manuelle Temperatur fuer " + id + " Temperatur = " + Source\_ManTempSet, "info"); }

SyncThermostat(room, "ManualTemp", Source\_ManTempSet, id); // jetzt manuelle Temp synchen

return;

} // ID in controll tab gefunden

} // endfor

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion SensorChange erkennt die Verschlussstellung eines Sensors und stellt die Temperatur entsprechend ein

//-----------------------------------------------------------------------------------------------------

function SensorChange(id) {

var tabNo;

var delay = 0;

var IDThermostat;

tabNo = SensorFind(id);

if (tabNo === 999) {

log("Routine SensorChange: Sensor " + id + " nicht in Sensorliste gefunden","info");

return; // Sensor nicht in Sensorlist gefunden

}

SensorList[tabNo][7] = SensorStatCalc(id, SensorList[tabNo][2]); // id des Sensors und device type - Der Status des Sensors wird ermittelt und in die sensorliste eingetragen

// Status des Verschlusses in Tabelle updaten

var room = SensorList[tabNo][0].toString();

// Wenn keine Heizperiode

if (getState(StateHeizperiode).val === false) {

return; // keine Heizperiode

}

if (getState(path + "." + room + ".Source\_Profil").val === 0 ) {

return; // es ist noch kein Profil dem Raum zugeordnet

}

if (debug) {

log("Routine SensorChange: Fenster " + id + " status geaendert fuer Raum " + room + " " + SensorList[tabNo][1] + " " + SensorList[tabNo][0] + " " + SensorList[tabNo][7], "info" );

log("Routine SensorChange: Sensor ist direktverknuepft ? " + SensorList[tabNo][9] + " fuer Raum "+ room,"info");

log("Routine SensorChange: Sensor status ist ? " + SensorList[tabNo][7] + " fuer Raum "+ room,"info");

}

// delay in Minuten ermitteln und Thermostat ID ermitteln

for (var x in ControlTab ) {

if (ControlTab[x][0] === room ) {

delay = ControlTab[x][12];

IDThermostat = ControlTab[x][1];

break;

}

}

// Fenster wurde geoeffnet

if (VerschlussRaumStatus(room) === true ) { // Mindestens ein Fenster ist geoeffnet

log("SensorChange Raum ist offen")

setOwnState(path + "." + room + ".RaumStatusVerschluss", true); // Raum ist geoeffnet

SaveStatus("Fenster", room,false); // "Verschluss offen TemperaturAbsenkung gesetzt";

if (SensorList[tabNo][9] === false) { // Sensor ist nicht direktverknuepft

if (debug) { log("Routine SensorChange : Raum " + room + " nicht direkt verknuepfter Sensor - Verschluss offen es Temp wird abgesenkt", "info"); }

SetTemp(room, VerschlussAbsenkungsGrenze, IDThermostat, false);

writelog(room, id, "Routine SensorChange: Sensor Direktverknuepft Sensorstatus geaendert - ID - Temp auf Absenkung" + id + " Raum " + room +" auf "+ SensorList[tabNo][7]) ;

} // endif Sensor ist nicht direktverknüpft

if (SensorList[tabNo][9] === true ) { // Sensor ist direktverknuepft

if (debug) { log("Routine SensorChange : Raum " + room + " direkt verknuepfter Sensor - Verschluss offen es Temp wird vom Thermostat abgesenkt", "info"); }

}

}

// Fenster wurde geschlossen

if (VerschlussRaumStatus(room) === false ) { // Raum ist geschlossen

setOwnState(path + "." + room + ".RaumStatusVerschluss", false); // Raum ist geschlossen

SaveStatus("Heizplan", room,false); // Meldung loeschen

if (SensorList[tabNo][9] === false) { // Sensor ist nicht direktverknuepft

if(debug) { log("Routine SensorChange: Delay Time nach Schlessen des nicht direkt verknüpften Sensors gesetzt","info"); }

} else {

if (debug) { log("Routine SensorChange : Raum " + room + " nicht verknuepfter Sensor - Verschluss geschlossen - Thermostat passt die Temp an", "info"); }

}

SetRoomClosed(room,delay); // zeitstempel setzen wenn Fenster geschlossen wurde - delay wenn erforderlich

}

LoopDevices(room); // jetzt wieder normale Temp ermitteln

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion SensorStatCalc Setzt den Sensorsatus um in true oder false fuer Geraete die mehr Status zur Verfuegung stellen

//-----------------------------------------------------------------------------------------------------

function SensorStatCalc(id, devtype) {

var SensorStatus = getState(id).val;

if (SensorStatus === true || SensorStatus === false ) {

if ( debug ) { log("Routine SensorStatCalc: Sensorstatus ist "+ SensorStatus + " fuer devtype = " + devtype + " und id "+ id); }

return SensorStatus;

}

// handelt es sich um einen HM Sensor mit anderen Status als true oder false ?

if (devtype !== "NoneHM") {

for (var x in SensorTypeTab ) {

if (devtype == SensorTypeTab[x][1]) {

if (SensorStatus === SensorTypeTab[x][6] ) {

if (debug ) {log("Routine SensorStatCalc: Sensorstatus ist geschlossen fuer devtype = " + devtype + " und id "+ id); }

return false;

}

else {

if (debug ) {log("Routine SensorStatCalc: Sensorstatus ist geoeffnet fuer devtype = " + devtype + " und id "+ id); }

return true;

}

}

}

} // HM check

// jetzt checken ob der NoneHM Sensor geschlossen oder geöffnet ist

if (devtype === "NoneHM") {

for (var x in NoneHMSenorTab ) {

if (id === NoneHMSenorTab[x][1] + "." + NoneHMSenorTab[x][2] + "." + NoneHMSenorTab[x][3]) {

if (SensorStatus == NoneHMSenorTab[x][4] ) {

if (debug ) { log("Routine SensorStatCalc: Sensorstatus ist geschlossen fuer devtype = " + devtype + " und id "+ id); }

return false;

}

else {

if (debug ) { log("Routine SensorStatCalc: Sensorstatus auf geoeffnet fuer devtype = " + devtype + " und id "+ id); }

return true;

}

}

}

} // Ende NoneHM check

log("Routine SensorStatCalc: Sensorstatus fuer " + id + " und devtype " + devtype + " Logik nicht implementiert - Status war " + SensorStatus , "info");

return false; // fall back wenn keine Bedingung zutrifft = nicht implementierte Logik

} // endfunction

//-----------------------------------------------------------------------------------------------------

//ab hier Hauptfunktionen

// 1. Manual Adjustments -- erkennt manuelle Anpassungen der Temperatur oder setzt diese zurueck

// 2. Overrule -- Behandelt globale Parameter und ggf davon abhaengige Temperaturanpassungen

// 3 ExecuteTempDetermination -- Hauptroutine zur Findung und speichern der Solltemp abhängig vom schedule

// 4. SelectSwitchTime -- Ermittlung die gültige Zeit des schedules (nicht Planungstag)

// 5. ActiveProfile -- Bestimmt das aktive Raumprofil

// 6. SetEventGlobalParameter -- ermittelt aufugrund von ICAL ggf gueltige Events

// 7. SetTemp -- Nur hier wird die Solltemperatur fuer die Thermostate gesetzt

// 8. DetermineSchedule -- ist input fuer ExecuteTempDetermination und ermittelt den richtigen Planungstag (1 bis 8 ) SelectSwitchTime ermittelt die Zeit

// 9. SyncThermostat -- Synchronisiert andere im Raum befindliche Thermostate wenn kein anderes zentrales Thermostat vorhanden ist

// 10.CreateStates -- legt alle relevanten States fuer Raeume und profile an

//-----------------------------------------------------------------------------------------------------

//-----------------------------------------------------------------------------------------------------

// Funktion ManAdjustments checkt ob eineThermostat/Raum Temperatur manuell angpasst wurde

//-----------------------------------------------------------------------------------------------------

function ManAdjustments(room, id) {

var ActiveRoomProfile = ActiveProfile(room); // Ermittlung des aktiven Raumprofilsfunction ManAdjustments(room, id) {

var ViewManDuration = getState(path + "." + room +".View\_Manual\_Temp\_Duration").val; // Dauer der manuellen Aenderung

var ViewManValidity = getState(path + "." + room +".View\_ManTemp\_Validity").val; // Errechnete Gueltigkeit als Ende Zeit

var ViewManValue = Calculate\_SollTemp(getState(path + "." + room +".View\_Manually\_Adjusted").val, "SetTemp"); // Check fuer eine manuelle Temp Aenderung aus dem view

var SourceManValue = Calculate\_SollTemp(getState(path + "." + room +".Source\_Manually\_Adjusted").val, "SetTemp"); // Check fuer eine manuelle Temp Aenderung aus dem view

var ManAdjTimeStamp = getState(path + "." + room +".Source\_TimeStamp").val; // Zeitstempel fall eine manuelle korrektur schon aktiv ist

var last\_Soll\_Temp = getState(path + "." + room +".Source\_Last\_Temp").val; // Letzte gespeicherte Soll Temperatur

var currentSollTemp = getState(id).val; // Ist die Geraete SollTemp ungleich der zuletzt gespeicherten Soll Temp ?

var NewCurrSollTemp = OverruleSollTemp (room,ActiveRoomProfile,SelectSwitchTime (room, ActiveRoomProfile, "CurrSollTemp"),id); // NEU -- ermittellt die aktuell zu erwartende Solltemperatur

var NewCurrTimeSlot = OverruleSollTemp (room,ActiveRoomProfile,SelectSwitchTime (room, ActiveRoomProfile, "CurrSlot"),id); // NEU -- // NEU -- ermittellt den aktuellen Timeslot für zu erwartende Solltemperatur

var NextTimeSlot = OverruleSollTemp (room,ActiveRoomProfile,SelectSwitchTime (room, ActiveRoomProfile, "NextSlot"),id); // NEU -- ermittellt den nächsten Timeslot für die nächste zu erwartende Solltemperatur

var NextSollTemp = OverruleSollTemp (room,ActiveRoomProfile,SelectSwitchTime (room, ActiveRoomProfile, "NextSollTemp"),id); // NEU -- ermittellt die nächste zu erwartende Solltemperatur

var ActTime = formatDate(new Date(),"YYYY/MM/DD SS:mm:ss"); // Formatierte Aktuelle Zeit

var ActTimeMilliSek = new Date(ActTime).getTime(); // Aktuelle Zeit in Millisekunden seit 1970

var bisTime = getState(path + "." + room + ".View\_ManTemp\_Validity").val; // gespeicherte Bistime

var bisSetTimeMilliSek;

var lastSetTimeMilliSek;

var tillSetTimeMilliSek; // Restdauer der manuellen Temperatur

var ChckAbsenkung = false;

var SoftBoostSchalter = getState(path + "." + room + ".SoftBoostActive").val; // SoftBoost Switch

var SoftBoostDauer = getState(path + "." + room + ".SoftBoostDuration").val; // SoftBoost Dauer in Minuten

var SoftBoostGueltigkeit = getState(path + "." + room + ".SoftBoostValidity").val; // SoftBoost Gueltigkeit bis - oder init

var SoftBoostTime = getState(path + "." + room + ".SoftBoostValidity").val; // Zeitstempel bei SoftBoost Aktiv

var ManDebug = false;

// Zeitberechnungen fuer manuelle Gültigkeiten

lastSetTimeMilliSek = ActTimeMilliSek;

bisSetTimeMilliSek = lastSetTimeMilliSek + ViewManDuration \* 60 \* 1000;

if (ManDebug) {

if( room === "Wohnzimmer") {

log("Routine Manadj: errechnete SollTemp ist NewCurrSollTemp " + NewCurrSollTemp, "info");

log("Routine Manadj: errechneter TimeSlot ist NewCurrTimeSlot " + NewCurrTimeSlot, "info");

log("Routine Manadj: momentane SollTemp ist currentSollTemp " + currentSollTemp + " fuer ID " + id, "info");

log("Routine Manadj: errechnete Next SollTemp ist NextSollTemp " + NextSollTemp, "info");

log("Routine Manadj: errechneter Next Timeslot ist NextTimeSlot " + NextTimeSlot, "info");

log("Routine Manadj: ViewManValue ist " + ViewManValue, "info");

log("Routine Manadj: ViewManValidity ist " + ViewManValidity, "info");

log("Routine Manadj: last\_Soll\_Temp ist " + last\_Soll\_Temp, "info");

log("Routine Manadj: last\_Soll\_Temp ist " + last\_Soll\_Temp, "info");

log("Routine Manadj: bisTime ist " + bisTime, "info");

}

}

LastRoomUpdateTime(room, "push"); // die aktuelle Zeit wird in das Array eingetragen um doppelte Ausführungen zu vermeiden, falls die SollTemp geaendert wird.

Source\_NextSollTemp = NextSollTemp;

// Erster Abschnitt der manuellen Tempermittlung ermittelt Bedingungen, die nicht zu einer manuellen Temperatur führen

// Pruefen ob ein FEnster geoeffnet ist

if (VerschlussRaumStatus(room) === true ) { // Mindestens ein Fenster ist geoeffnet

log("Raum geoeffnet "+ room)

setOwnState(path + "." + room + ".RaumStatusVerschluss", true); // Raum ist geoeffnet

SaveStatus("Fenster", room,false); // "Verschluss offen TemperaturAbsenkung gesetzt";

if (debug) { log("Routine ManAdjustments : Raum " + room + " ist geoeffnet ", "info"); }

SetTemp(room, VerschlussAbsenkungsGrenze, id, false);

writelog(room, id, "Routine ManualAdjustments: Raum Geoeffnet - Temp wird auf Absenkung gesetzt" + id + " Raum " + room) ;

if (debug) { log("Routine SensorChange : Raum " + room + " Verschluss offen es Temp wird vom Thermostat abgesenkt", "info"); }

return true;

}

// entspricht die aktuelle Temp des Thermostates der Absenktemperatur bei Fenster offen ?

if (currentSollTemp === VerschlussAbsenkungsGrenze) {

if (ManDebug) { log("Tempabsenkung erkannt fuer Raum "+ room,"info"); }

ChckAbsenkung = true;

}

// Allgemeine Bedingungen bei denen zunächst nur prinzipiell ermittelt wird ob evt eine manuelle Temp vorliegt

var manuellRelevant = false;

if (ManAdjTimeStamp !== "init") manuellRelevant = true;

if (last\_Soll\_Temp !== currentSollTemp) manuellRelevant = true;

if (ViewManValue > 0) {

if (last\_Soll\_Temp !== ViewManValue) manuellRelevant = true;

if (ViewManValue !== currentSollTemp) manuellRelevant = true;

}

if (debug) { log("Routine ManAdjustments: Manuell Relevant?: " + manuellRelevant, "info"); }

if (ManAdjTimeStamp !== "init") manuellRelevant = true;

if (last\_Soll\_Temp !== currentSollTemp) manuellRelevant = true;

if (ViewManValue > 0) {

if (last\_Soll\_Temp !== ViewManValue) manuellRelevant = true;

if (ViewManValue !== currentSollTemp) manuellRelevant = true;

}

if (debug) { log("Routine ManAdjustments: Manuell Relevant?: " + manuellRelevant, "info"); }

if (ManTimeouts[room]) {

clearTimeout(ManTimeouts[room]);

ManTimeouts[room] = null;

}

// Delaybearbeitung - Nach schliessen eines Fensters wird abgewartet bis der Delay beendet ist (Einstellung in der Thermostattabelle)

var DelayTime = CheckDelay(room); // gibt es einen delay ? 0 = kein Delay - Delays werden in der ThermostatTypeTab eingetragen und bei schliessen eines Raumes aktiviert

if (DelayTime > 0 ) { // es gibt einen aktiven Delay des Raumes

var RestDelay = DelayTime - ActTimeMilliSek;

ManTimeouts[room] = setTimeout(TriggerUpdate, RestDelay, room, true); // Wiederherstellen des Delays mit der Restzeit - geht irgendwie verloren

if (ManDebug) { log("Routine ManAdjustments: Warten bis Ablauf DelayTime - geplant fuer Raum "+ room + " um " + formatDate(DelayTime,"YYYY/MM/DD SS:mm:ss"),"info"); }

return true; // mache nichts weiter und warte auf Ablauf des Delays

}

//....Ende Delaybearbeitung

// SoftBoost Bearbeitung

//SoftBoost läuft aber Raum wurde geoeffnet

if ( ChckAbsenkung === true ) {

if (SoftBoostSchalter === true || SoftBoostGueltigkeit !== "init" ) { // SoftBoost läuft

if (ManDebug) { log ("Boost läuft und Absenkung erkannt ","info"); }

// jetzt wieder den timeout fuer den Softboost setzen, der wurde nämlich durch die gesetzte Absenkung gelöscht

tillSetTimeMilliSek = new Date(SoftBoostTime).getTime() - new Date().getTime();

if (tillSetTimeMilliSek > 0) { // SoftBoost Time noch gueltig

if (ManDebug) { log("Routine ManAdjustments: Restlaufzeit der SoftBoost Zeit "+ tillSetTimeMilliSek,"info"); }

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek, room, true);

} else {

setOwnState(path + "." + room + ".SoftBoostValidity", "init"); // Initilize

setOwnState(path + "." + room + ".SoftBoostActive", false); // Initilize

}

return true; // zurück und keine neue Tempfindung

}

} // Ende Raum geoeffnet

// SoftBoost neu gestartet

if (SoftBoostGueltigkeit === "init" && SoftBoostSchalter) { // SoftBoost wurde gerade gestartet

if(!ChckAbsenkung) { // nur wenn keine TempAbsenkung

bisSetTimeMilliSek = ActTimeMilliSek + SoftBoostDauer \* 60 \* 1000; // Bis Zeit der Gueltigkeit der manuellen Aenderung in Millisekunden

setOwnState(path + "." + room + ".SoftBoostValidity", formatDate(bisSetTimeMilliSek, "YYYY/MM/DD SS:mm:ss")); // Zeitstempel mit der aktuellen Zeit versehen

SetTemp(room, SoftBoostTemp, id, false); // setze Temp auf BoostTemp

ManTimeouts[room] = setTimeout(TriggerUpdate, SoftBoostDauer \* 60 \* 1000, room, true);

if (debug) { log("Routine ManAdjustments: SoftBoost neu gesetzt für " + room + " ist " + formatDate(bisSetTimeMilliSek+1000, "YYYY/MM/DD SS:mm:ss")); }

return true; // zurück und keine neue Tempfindung

}

}

// SoftBoost abgelaufen ?

lastSetTimeMilliSek = new Date(SoftBoostGueltigkeit).getTime(); // gespeicherte Von-Zeit wenn SoftBoost gestartet wurde in Millisekunden seit 1970

if (SoftBoostSchalter && SoftBoostGueltigkeit !== "init") { // SoftBoost läuft

if (lastSetTimeMilliSek <= ActTimeMilliSek) { // die SoftBoost Zeit ist abgelaufen

if (debug) { log("Routine ManAdjustments: SoftBoost abgelaufen","info"); }

setOwnState(path + "." + room + ".SoftBoostValidity", "init"); // Initilize

setOwnState(path + "." + room + ".SoftBoostActive", false); // Initilize

SoftBoostSchalter = false;

if (ManDebug) { log("SoftBoost Bedingung für reaktiverung einer manuellen Temp erfüllt","info"); }

if (ManTempReset(room,bisTime,ViewManValue,id) === true ) { // reaktivierung manuelle aenderung und Zeittrigger setzen

return true; // zurück und keine neue Tempfindung

} else {

return false; // zurück und neue Tempfindung

}

} // endif lastsettimemillisek

} // endif softboostschalter

// SoftBoost manuell beendet

if (SoftBoostSchalter === false && SoftBoostGueltigkeit !== "init") { // SoftBoost läuft wurde aber beendet

setOwnState(path + "." + room + ".SoftBoostValidity", "init"); // Initilize

if (debug) { log("Routine ManAdjustments: SoftBoost manuell beendet","info"); }

if (ManTempReset(room,bisTime,ViewManValue,id) === true ) { // reaktivierung manuelle aenderung und Zeittrigger setzen

return true; // zurück und keine neue Tempfindung

} else {

return false; // zurück und neue Tempfindung

}

}

if (SoftBoostSchalter) { // SoftBoost läuft noch - abwarten

log("Softboost fuer Raum " + room + " läuft noch - abwarten", "info");

if ( ViewManValue > 0 && ManAdjTimeStamp === "init") { // neue manuelle Temp während SoftBoost

if (ManDebug) { log("Softboost fuer Raum " + room + " läuft noch - Berechnung der Gültigkeit der neuen manuellen Temp","info"); }

setOwnState(path + "." + room + ".Source\_TimeStamp", ActTime); // Zeitstempel mit der aktuellen Zeit versehen

setOwnState(path + "." + room + ".View\_ManTemp\_Validity", formatDate(bisSetTimeMilliSek, "YYYY/MM/DD SS:mm:ss")); // Zeitstempel fuer die manuelle Gueltigkeit

// jetzt wieder den timeout fuer den Softboost setzen, der wurde nämlich durch die gesetzte manuelle Temp gelöscht

tillSetTimeMilliSek = new Date(SoftBoostTime).getTime() - new Date().getTime();

if (tillSetTimeMilliSek > 0) { // SoftBoost Time noch gueltig

if (ManDebug) { log("Routine ManAdjustments: Restlaufzeit der SoftBoost Zeit "+ tillSetTimeMilliSek,"info"); }

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek, room, true);

} else {

setOwnState(path + "." + room + ".SoftBoostValidity", "init"); // Initilize

setOwnState(path + "." + room + ".SoftBoostActive", false); // Initilize

}

}

return true; // zurück und keine neue Tempfindung

}

// Ende SoftBoost

if (ManDebug) { log("checke Gueltigkeit View kleiner Null","info"); }

// Wenn im View die Gültigkeit kleiner Null ist wird jegliche manuelle Temp zurückgesetzt

if (ViewManDuration < 0 ) { // die Dauer der manuellen Aenderung ist auf kleiner null gesetzt . das heisst, dass keine manuellen Aenderungen zugelassen sind

Source\_ManualAdjustment = "Manuelle Temperatur Erkennung ist ausgeschaltet";

if (ManAdjTimeStamp !== "init" || ViewManValidity !== "init" || ViewManValue > 0) {

InitilizeManChange(room); // Manuelle Aenderungen zuruecksetzen

Source\_ManualAdjustment = "Manuelle Temperatur Erkennung initialisiert und ausgeschaltet";

}

if (debug) { log("Routine ManAdjustments: " + Source\_ManualAdjustment, "info"); }

return false;

}

if (ManDebug) { log("checke manuelle Temp = aktueller geplanter Temp ?","info"); }

// jetzt prüfen ob evt eine manuelle Temp gesetzt werden soll die der neuen SollTemp entspricht - Wenn dies der Fall ist, dann manuelle Temp ignorieren

if (ViewManValue === NewCurrSollTemp) { // Die aktuelle Solltemperatur des Thermostates ist gleich der Solltemperatur, die gleich gesetzt werden soll oder die view temp ist auf null gesetzt

InitilizeManChange(room); // Manuelle Aenderungen zuruecksetzen

if (debug) {log("Manuelle Temperatur entspricht der errechneten SollTemp - Manuelle Temp wird zurückgesetzt ", "info"); }

writelog(room, id, "Manuelle Temperatur entspricht der errechneten SollTemp - Manuelle Temp wird zurückgesetzt ");

return false; // gehe weiter mit der normalen Temp Determination

} // Ende manuelle Temp = errechnete SollTemp

lastSetTimeMilliSek = ActTimeMilliSek;

bisSetTimeMilliSek = lastSetTimeMilliSek + ViewManDuration \* 60 \* 1000;

// Wenn es bereits eine Manuelle Temp gibt, dann wird jetzt Die Biszeit in Millisek berechnet

if (ManAdjTimeStamp !== "init") { // eine manuelle Temperatur wurde bereits gespeichert

lastSetTimeMilliSek = new Date(ManAdjTimeStamp).getTime(); // gespeicherte Von-Zeit der manuellen Aenderung in Millisekunden seit 1970

bisSetTimeMilliSek = lastSetTimeMilliSek + ViewManDuration \* 60 \* 1000; // Bis Zeit der Gueltigkeit der manuellen Aenderung in Millisekunen

}

// wenn die manuelle Aenderung bei einem Slotwechsel zurückgesetzt werden soll wird das Ende des Slotwechsels berechnet

if (ViewManDuration == 0 && ManAdjTimeStamp === "init" && manuellRelevant) { // "==="" ergibt Probleme bei ViewManDuration Aenderungen im view - nummber- string problem

bisSetTimeMilliSek = SelectSwitchTime (room, ActiveRoomProfile, "CurrSlotEnde");

bisSetTimeMilliSek = bisSetTimeMilliSek ; // bis time entspricht genau dem slotwechsel

if (debug) { log("Routine ManAdjustments: Die manuelle gesetzte Temperatur wird zurückgesetzt um " + formatDate(bisSetTimeMilliSek, "YYYY/MM/DD SS:mm:ss", "info")); }

}

// Ermittlung der restlichen Zeit bis zum Ablauf der manuellen Temperatur

tillSetTimeMilliSek = bisSetTimeMilliSek - new Date().getTime(); //tilSetTimeMilliSek ist die Restdauer der manuellen Temperatur

// 0. Fall Eine manuelle Tempertur war eingestellt und muss wieder aktiviert werden+

if (ManDebug) { log("Ueberpruefung Fall 0 ManAdjTimeStamp "+ ManAdjTimeStamp +" NewCurrSollTemp "+ NewCurrSollTemp + " ViewManValue "+ ViewManValue + " Raum " + room,"info"); }

if (ManAdjTimeStamp !== "init" && NewCurrSollTemp !== ViewManValue && ViewManValue > 0 && ChckAbsenkung !== true && currentSollTemp !== SoftBoostTemp ) { // Manuelle Temp ist eingestellt . Die gegenwärtige Temp ist aber anders

if (ManDebug) { log("Bedingung fuer Reaktivierung der manuellen Temp erfüllt","info"); }

if( currentSollTemp !== ViewManValue ) { // es scheint, dass die Temperatur am Thermostat sich geändert hat

setOwnState(path + "." + room + ".View\_Manually\_Adjusted", Calculate\_SelectValueWert(currentSollTemp, "SetTemp"));

if (ManTempReset(room,bisTime,currentSollTemp,id) === true ) { // reaktivierung manuelle aenderung und Zeittrigger setzen

return true; // zurück und keine neue Tempfindung

} else {

return false; // zurück und neue Tempfindung

}

}

if (ManTempReset(room,bisTime,ViewManValue,id) === true ) { // reaktivierung manuelle aenderung und Zeittrigger setzen

return true; // zurück und keine neue Tempfindung

} else {

return false; // zurück und neue Tempfindung

}

}

// Es liegt eine manuelle Temperatur vor - entweder ist sie gerade erst festgestellt oder sie gab es schon und deren weiteren Behandlung wird überprüft und ausgeführt

if (ManDebug) { log("checke Fall 1", "info"); }

// 1. Fall eine manuelle Korrektur ist erkannt - Sie wurde gerade erst eingestellt

if (currentSollTemp !== last\_Soll\_Temp && ManAdjTimeStamp === "init" && ChckAbsenkung !== true) { // Temperturdifferenz erkannt und Zeitstempel noch nicht gesetzt

Source\_ManualAdjustment = "Manuelle Temperatur Verstellung am Thermostat auf " + currentSollTemp + " " + "erkannt";

SetTemp(room, currentSollTemp, id, false);

Source\_ManTempSet = currentSollTemp;

setOwnState(path + "." + room + ".Source\_TimeStamp", ActTime); // Zeitstempel mit der aktuellen Zeit versehen

setOwnState(path + "." + room + ".View\_ManTemp\_Validity", formatDate(bisSetTimeMilliSek, "YYYY/MM/DD SS:mm:ss")); // Zeitstempel mit der aktuellen Zeit versehen

setOwnState(path + "." + room + ".Source\_Manually\_Adjusted", Calculate\_SelectValueWert(currentSollTemp, "SetTemp"));

setOwnState(path + "." + room + ".View\_Manually\_Adjusted", Calculate\_SelectValueWert(currentSollTemp, "SetTemp"));

if (debug) { log("1.Fall " + Source\_ManualAdjustment, "info"); }

if (tillSetTimeMilliSek > 0) {

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek + 1000, room, true);

if (debug) {log("Routine ManAdjustments: Timeout für Delaytime gesetzt für " + room + " ist " + tillSetTimeMilliSek);}

}

writelog(room, id, "1. Fall - manuelle Temp erkannt auf " + currentSollTemp);

return true;

}

if (ManDebug) { log("checke Fall 2", "info"); }

// 2. Fall eine Temperatur wurde im View eingegeben - eine vorherige Manuelle Temp gibt es nicht

if (ManAdjTimeStamp === "init" && ViewManValue > 0) { // Im View wurde eine manuelle Temperatur eingetragen

Source\_ManualAdjustment = "Manuelle Temperatur Verstellung im View auf " + ViewManValue + " " + "erkannt";

SetTemp(room, ViewManValue, id, false);

Source\_ManTempSet = ViewManValue;

setOwnState(path + "." + room + ".Source\_TimeStamp", ActTime); // Zeitstempel mit der aktuellen Zeit versehen

setOwnState(path + "." + room + ".View\_ManTemp\_Validity", formatDate(bisSetTimeMilliSek, "YYYY/MM/DD SS:mm:ss")); // Zeitstempel mit der aktuellen Zeit versehen

setOwnState(path + "." + room + ".Source\_Manually\_Adjusted", Calculate\_SelectValueWert(ViewManValue, "SetTemp"));

if (debug) { log("2.Fall " + Source\_ManualAdjustment, "info"); }

if (tillSetTimeMilliSek > 0) {

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek + 1000, room, true);

if (debug) {log("Routine ManAdjustments: Timeout für Delaytime gesetzt für " + room + " ist " + formatDate(tillSetTimeMilliSek, "YYYY/MM/DD SS:mm:ss"));}

}

writelog(room, id, "2. Fall eine Temperatur wurde im View eingegeben " + ViewManValue);

return true;

}

if (ManDebug) { log("checke Fall 3", "info"); }

// 3. Fall im View wurde eine Temperatur gesetzt die von der vorher gesetzten Temperatur abweicht - overrule durch view

if (ManAdjTimeStamp !== "init" && SourceManValue !== ViewManValue && ViewManValue > 0) { // Im View wurde eine manuelle Temperatur eingetragen

Source\_ManualAdjustment = "Manuelle Temperatur wurde im View auf neu gesetzt ";

SetTemp(room, ViewManValue, id, false);

Source\_ManTempSet = ViewManValue;

setOwnState(path + "." + room + ".Source\_Manually\_Adjusted", Calculate\_SelectValueWert(ViewManValue, "SetTemp")) ;

setOwnState(path + "." + room + ".View\_Manually\_Adjusted", Calculate\_SelectValueWert(ViewManValue,"SetTemp")) ;

if (debug) { log("3.Fall " + Source\_ManualAdjustment, "info"); }

if (tillSetTimeMilliSek > 0) {

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek + 1000, room, true);

if (debug) {log("Routine ManAdjustments: Timeout für Delaytime gesetzt für " + room + " ist " + tillSetTimeMilliSek);}

}

writelog(room, id, "3. Fall im View wurde eine Temperatur gesetzt die von der vorher gesetzten Temperatur abweicht " + ViewManValue);

return true;

}

if (ManDebug) { log("checke Fall 4", "info"); }

// 4. Fall im View wurde die Temperatur auf 0 gesetzt - das fuehrt zum reset der manuellen Temperatur

if (ManAdjTimeStamp !== "init" && ViewManValue === 0 && SourceManValue > 0) { // Im View wurde eine manuelle Temperatur eingetragen

Source\_ManualAdjustment = "4. Fall - Manuelle Temperatur wurde im View auf Null gesetzt - Loeschen der manuellen Temperatur";

InitilizeManChange(room); // Manuelle Aenderungen zuruecksetzen

if (debug) { log("4.Fall " + Source\_ManualAdjustment, "info"); }

writelog(room, id, "4. Fall im View wurde die Temperatur auf 0 gesetzt ");

return false;

}

if (ManDebug) { log("checke Fall 5", "info"); }

// 5. Fall Die Manuelle Temperatur wurde am Thermostat veraendert

if (currentSollTemp !== SourceManValue && ManAdjTimeStamp !== "init" && ChckAbsenkung !== true) {

if( currentSollTemp === NewCurrSollTemp ) { // Die aktuelle Solltemperatur des Thermostates ist gleich der Solltemperatur, die gleich gesetzt werden soll oder die view temp ist auf null gesetzt

if (ManAdjTimeStamp !== "init" || ViewManValidity !== "init" || ViewManValue > 0 || ViewManValue > 0) {

InitilizeManChange(room); // Manuelle Aenderungen zuruecksetzen

Source\_ManualAdjustment = "5. Fall - Manuelle Temperatur zurückgesetzt";

if (debug) { log("Routine ManAdjustments: " + Source\_ManualAdjustment, "info"); }

}

if (debug) { log("keine ManAdjustments festgestellt fuer Raum "+room, "info"); }

writelog(room, id, "5. Fall Die Manuelle Temperatur wurde am Thermostat veraendert " + ViewManValue);

return true; // zurück

}

Source\_ManualAdjustment = "Manuelle Temperatur Verstellung im Termostat auf " + currentSollTemp + " " + "erkannt";

setOwnState(path + "." + room + ".Source\_Manually\_Adjusted", Calculate\_SelectValueWert(currentSollTemp, "SetTemp"));

setOwnState(path + "." + room + ".View\_Manually\_Adjusted", Calculate\_SelectValueWert(currentSollTemp, "SetTemp"));

Source\_ManTempSet = currentSollTemp;

if (debug) { log(Source\_ManualAdjustment, "info"); }

if (tillSetTimeMilliSek > 0) {

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek + 1000, room, true);

if (debug) {log("Routine ManAdjustments: Timeout für Delaytime gesetzt für " + room + " ist " + tillSetTimeMilliSek);}

}

return true;

}

if (ManDebug) { log("checke Fall 6", "info"); }

// 6. Fall - die Manuell gesetzte Temperatur wird auf abgelaufen ueberprueft

if (currentSollTemp !== last\_Soll\_Temp && ManAdjTimeStamp !== "init" ) { // Die manuelle Temperatur ist - pruefen ob abgelauen

// wenn abgelaufen, dann die manuelle Temperatur loeschen

if (bisSetTimeMilliSek <= ActTimeMilliSek) { // die manuelle Zeit ist abgelaufen

InitilizeManChange(room); // Manuelle Aenderungen zuruecksetzen

Source\_ManualAdjustment = "6a. Fall - Manuelle Temperatur abgelaufen um: " + ViewManValidity + " - zurueck zum Schedule";

if (debug) { log(Source\_ManualAdjustment, "info"); }

writelog(room, id, "6a Fall. Manuelle Temperatur abgelaufen um " + ViewManValidity);

return false;

}

else { // Temperatur noch nicht abgelaufen

if (ChckAbsenkung !== true ) { // wenn keine Temperaturabsenkung vorliegt

Source\_ManualAdjustment = "6b. Fall - Manuelle Temperatur noch aktuell - warten bis " + ViewManValidity + " - Temperatur ist " + currentSollTemp;

Source\_ManTempSet = currentSollTemp;

if (debug) { log(Source\_ManualAdjustment,"info"); }

if (tillSetTimeMilliSek > 0) {

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek + 1000, room, true);

if (debug) {log("Routine ManAdjustments: Timeout für Delaytime gesetzt für " + room + " ist " + tillSetTimeMilliSek);}

}

writelog(room, id, "6b Fall. Manuelle Temperatur noch aktuell bis " + ViewManValidity);

}

else {

SetTemp(room, SourceManValue, id, false); // alte manuelle Temperatur wieder einstellen

Source\_ManTempSet = SourceManValue;

if (tillSetTimeMilliSek > 0) {

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek + 1000, room, true);

if (debug) {log("Routine ManAdjustments: Timeout für Delaytime gesetzt für " + room + " ist " + tillSetTimeMilliSek);}

}

writelog(room, id, "6c Fall. Manuelle Temperatur wieder eingestellt " + SourceManValue);

return true;

}

} // endif Die manuelle Zeit ist abgelaufen

} // check ob die manuelle Zeit abgelaufen ist

if (ManDebug) { log("checke Fall 1", "info"); }

// 7. Fall - die Manuell gesetzte Temperatur ist abgelaufen- der schedule hat gewechselt und entspricht der vorherigen manuellen Temperatur (Ausnahmesituation)

if (ManAdjTimeStamp !== "init" ) { // Die manuelle Temperatur ist - pruefen ob abgelauen

if (bisSetTimeMilliSek + cron \* 60 \* 1000 < ActTimeMilliSek) {

InitilizeManChange(room); // Manuelle Aenderungen zuruecksetzen

Source\_ManualAdjustment = "7. Fall - Manuelle Temperatur abgelaufen um: " + ViewManValidity + " - zurueck zum Schedule";

if (debug) { log(Source\_ManualAdjustment, "info"); }

writelog(room, id, "7. Fall - die Manuell gesetzte Temperatur ist abgelaufen");

return false;

}

}

if (ManDebug) { log("nichts in den manuellen checks traf zu. zurück und neue Tempfindung", "info"); }

return false;

} // Endfunction

//-----------------------------------------------------------------------------------------------------

// Funktion ManTempReset - Reset Manuelle Aenderung und starte timer neu

//-----------------------------------------------------------------------------------------------------

function ManTempReset (room,bisTime,ViewManValue,id ) {

var bisSetTimeMilliSek = new Date(bisTime).getTime();

var tillSetTimeMilliSek = bisSetTimeMilliSek - new Date().getTime();

if (tillSetTimeMilliSek > 0) { // manuelle Temp gueltig

if (debug) { log("Routine ManTempReset: Restlaufzeit einer ggf man Aenderung "+ tillSetTimeMilliSek,"info"); }

SetTemp(room, ViewManValue, id, false);

Source\_ManTempSet = ViewManValue;

ManTimeouts[room] = setTimeout(TriggerUpdate, tillSetTimeMilliSek, room, true);

return true // manuelle Aenderung eingestellt - keine neue Tempfindung

} else { // manuelle Temp abgelaufen

if (debug) { log("Routine ManTempReset: Manuelle Temp abgelaufen zurück zum Schedule","info"); }

InitilizeManChange(room); // Manuelle Aenderungen zuruecksetzen

Source\_ManualAdjustment = "Manuelle Temperatur abgelaufen fuer Raum " + room + " - zurueck zum Schedule";

if (debug) { log(Source\_ManualAdjustment, "info"); }

writelog(room, id, "Manuelle Temperatur abgelaufen -Raum " + room );

return false; // neue Tempfindung

}

} // Endfunction

//-----------------------------------------------------------------------------------------------------

// Funktion OverruleSollTemp arbeitet nach der vorgegebenen Reihenfolge die globalen und Profilparameter ab

//-----------------------------------------------------------------------------------------------------

function OverruleSollTemp (room,Profil,SollTempSched,id) {

if (getState(StateHeizperiode).val === false ) { // keine Heizperiode

SaveStatus("HeizperiodeAus", room,false);

return SollTempSched; // also keine Veraenderung der SollTemp

}

for (var x in OverruleTab) {

if (OverruleTab[x][0] === "UrlaubAnwesend") { // Wenn Urlaub dann check wie ein Feiertag eingestellt ist (Urlaub wie Feiertag)

if (getState(StateUrlaubAnwesend).val) {

if (getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_UrlaubWieFeiertag).val) {

SaveStatus("UrlaubAnwesend", room,false);

} // endif ist der Urlaub wie ein Feiertag ?

} // endif Urlaubanwesend ist true

}

if (OverruleTab[x][0] === "UrlaubAbwesend" && getState(StateUrlaubAbwesenheit).val) { // Wenn Urlaub dann Absenkung bis Mindestemperatur

if (Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_UrlaubAbsenkung).val, "CorrectTemp") !== 0) { // Absenkung geplant ?

SollTempSched = SollTempSched - Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_UrlaubAbsenkung).val, "CorrectTemp");

if (SollTempSched < Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_MinimaleTemperatur).val,"SetTemp")) { // Minimaltemp zieht

SollTempSched = Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_MinimaleTemperatur).val,"SetTemp");

SaveStatus("UrlaubAbwesend", room, true); // mit Mindesttemperatur

return SollTempSched;

}

SaveStatus("UrlaubAbwesend", room,false);

return SollTempSched;

}

}

if (OverruleTab[x][0] === "Gaeste") { // Wenn Gaeste dann Anhebung um Profilparameter

if (getState(StateGaesteDa).val && Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_GaesteAnhebung).val, "CorrectTemp") !== 0) { // Gaeste Anhebung muess ungleich 0 sein // nur wenn einen Anhebung geplant ist

SollTempSched = SollTempSched + Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_GaesteAnhebung).val, "CorrectTemp");

if (SollTempSched < Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_MinimaleTemperatur).val, "SetTemp")) {

SollTempSched = Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_MinimaleTemperatur).val, "SetTemp");

SaveStatus("Gaeste", room, true); // mit Mindesttemperatur

return SollTempSched;

}

SaveStatus("Gaeste", room,false);

return SollTempSched;

}

}

if (OverruleTab[x][0] === "Abwesenheit") { // Wenn Abwesenheit dann Absenkung um Profilparameter

if (getState(StateAnwesenheit).val === false && Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_AbwesenheitAbsenkung).val, "CorrectTemp") !== 0) { // Gaeste Anhebung muess ungleich 0 sein // nur wenn einen Anhebung geplant ist

log("Routine Overrule Profil ist " + Profil + " Absenkung ist " + Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_AbwesenheitAbsenkung).val, "CorrectTemp") + " Raum ist " + room)

SollTempSched = SollTempSched - Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_AbwesenheitAbsenkung).val, "CorrectTemp");

if(SollTempSched < Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_MinimaleTemperatur).val, "SetTemp")) {

SollTempSched = Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_MinimaleTemperatur).val, "SetTemp");

SaveStatus("Abwesend", room, true); // mit Mindesttemperatur

return SollTempSched;

}

SaveStatus("Abwesend", room,false);

return SollTempSched;

}

}

if (OverruleTab[x][0] === "Party") { // Wenn Party dann Absenkung um Profilparameter

if (getState(StatePartyjetzt).val && Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_PartyAbsenkung).val, "CorrectTemp") !== 0) { // Gaeste Anhebung muess ungleich 0 sein // nur wenn einen Anhebung geplant ist

SollTempSched = SollTempSched - Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_PartyAbsenkung).val, "CorrectTemp");

if (SollTempSched < Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_MinimaleTemperatur).val, "SetTemp")) {

SollTempSched = Calculate\_SollTemp(getState(path + "." + room + "." + "Profil-" + Profil + "." + StatePP\_MinimaleTemperatur).val, "SetTemp");

SaveStatus("Party", room, true); // mit Mindesttemperatur

return SollTempSched;

}

SaveStatus("Party", room,false);

return SollTempSched;

}

}

}

return SollTempSched;

} // Ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion ExecuteTempDetermination checkt ob eineThermostat/Raum Temperatur manuell angpasst wurde

//-----------------------------------------------------------------------------------------------------

function ExecuteTempDetermination(roomName, id) {

var ActiveRoomProfile;

var ScheduledSollTemp;

// Findung des aktuellen RaumProfiles

ActiveRoomProfile = ActiveProfile(roomName); // Ermittlung des aktiven Raumprofils

//die geplante Soll Temperatur aus dem Raumschedule aus dem aktuellen Profil ermitteln

ScheduledSollTemp = SelectSwitchTime(roomName, ActiveRoomProfile, "CurrSollTemp",id); // Ermittlung der geplanten Solltemperatur

if (debug) {

log("Routine ExecuteTempDetermination: raum " + roomName + "Solltemp nach Switchtime: " + ScheduledSollTemp + " Findung " + Source\_SchedulePoint, "info");

}

// Schauen ob die ermittelte Temperatur angepasst werden muss z.B. party Gaeste etc

ScheduledSollTemp = OverruleSollTemp(roomName, ActiveRoomProfile, ScheduledSollTemp, id); //Global und Profilparameter koennen den schedule uebersteuern

if (debug) {

Source\_GlobalParameter = getState(path + "." + roomName + ".Source\_Global\_Parameter").val;

log("Routine ExecuteTempDetermination: raum " + roomName + "Solltemp nach overrule: " + ScheduledSollTemp+ " Findung " + Source\_GlobalParameter, "info");

}

// jetzt die Temperatur dem Thermostat uebermitteln

SetTemp(roomName, ScheduledSollTemp, id, true); // jetzt die Temperatur schalten

if (NextSchedules[roomName]) {

if (debug) { log("Schedule gelöscht für " + roomName, "info"); }

clearSchedule(NextSchedules[roomName]);

NextSchedules[roomName] = null;

}

// jetzt Delay-Sekunden ermitteln, um die Schedules nicht gleichzeitig auszuführen

var delaysek=0;

var z = 1;

for (var roomCheck in rooms) { // loop ueber all Raeume

roomCheck = roomCheck.replace(/\s/g, "\_"); // Blanks durch unterstrich ersetzen

if ( roomCheck === roomName) {

delaysek = z \* 2; // es werden alle 2 Sekunden ein schedule geplant also 2,4,6....

if (delaysek > 58) { // jetzt sind 58 Sekunden erreicht

delaysek = (delaysek - 59) \* 2 - 1 ; // also mit 1,3,5... Sekunden weiter planen

}

if ( z > 59) { // mehr als 59 Räume mit Thermostaten? wohl kaum

delaysek = 0;

}

break;

}

z = z + 1

}//endfor roomcheck

// jetzt die Cron Pattern bestimmen und einplanen

var schedArr = Source\_SchedulePoint.split("\_"); // [0]=Mo, [1]=00:00:00

schedArr[1] = schedArr[1].substr(0,6) + delaysek;

var nextSchedule = parseInt(schedArr[1].substr(6, 2), 10) + " " + parseInt(schedArr[1].substr(3, 2), 10) + " " + parseInt(schedArr[1].substr(0, 2), 10) + " \* \* \*";

if (debug) { log("Setze Schedule für nächste Planzeit " + nextSchedule + " für Raum " + roomName, "info"); }

NextSchedules[roomName] = schedule(nextSchedule, function() {

if (debug) { log("Schedule Triggered für nächste Planzeit für Raum " + roomName, "info"); }

TriggerUpdate(roomName);

});

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion SelectSwitchTime Finde Schaltzeit im schedule und geplante Solltemperatur

// Find nächst Schaltzeit

// Finde nächste Solltemperatur

//-----------------------------------------------------------------------------------------------------

// Funktion =

// NextSlot = finde nächste Schaltzeit

// CurrSlot = finde aktuellen ZeitSlot

// CurrSollTemp = finde aktuelle Solltemperatur des Slots

// NextSollTemp = finde Solltemperatur des nächsten Slots

// CurrSlotEnde = finde aktuellen Slot und sende die Zeit in Millisekunden für das Slotende zurück

//

function SelectSwitchTime(room, RaumProfil, Funktion,id) {

if (Funktion !== "CurrSlot" && Funktion !== "NextSlot" && Funktion !== "CurrSollTemp" && Funktion !== "NextSollTemp" && Funktion !== "CurrSlotEnde") {

log("Funktion unbekannt in fuction SelectSwitchTime Fumktion war " + Funktion, "info");

return 0 ; // Funktion unbekannt

}

// zuerst wird geprueft ob die Heizuperiode ausgeschaltet ist, In diesem Fall braucht kein Schedule ermittelt werden und die Sommer SollTemp wird zurueckgegeben

if (getState(StateHeizperiode).val === false ) { // keine Heizperiode

if ( Funktion === "CurrSollTemp" || Funktion === "NextSollTemp" ) { // nur fuer Temperaturermittlung. Der Slot muss trotzdem gefunden werden

for(var x in ControlTab) {

if ( id === ControlTab[x][1]) { // ID in der ControlTab gefunden

return ControlTab[x][11]; // Die Solltemp für die Sommerperiode wird zurückgegeben (kommt aus der ThermostatTab)

}

} // endfor controltab

} // endif Funktion

} // endif keine Heizperiode

// Folgende Verarbeitung findet den Scheduled Slot

room = room.replace(/\s/g, "\_"); // Alle Leerzeichen aus der Raumbezeichnung entfernen

var currTime = formatDate(new Date(),"SS:mm:ss");

var TemperatureScheduledTemp = 0;

var TemperatureScheduledSlot;

var NextTemperatureScheduledTemp = 0;

var NextTemperatureScheduledSlot;

var TimeBisScheduled = " ";

var TimeFrom = "00:00:00";

var y = 1;

var NextSchedulePointer = 0;

//var d = new Date();

var heute = currentDate();

var MilliSekHeute = 0;

var MilliSekSlot;

var weekday = new Date().getDay();

weekday = DetermineSchedule(room, weekday, RaumProfil); // tatsächlicher weekday wird uebersteuert, wenn ein "WieVortag" vorkommt

for (var i = 1; i <= 6; i++) { // es gibt 6 schedules pro Tag

TimeBisScheduled = getState(path + "." + room + ".Profil-" + RaumProfil + "." + Wochentag(weekday) + "\_" + i + "\_" + "bis").val;

if (isTimeInRange(TimeFrom, TimeBisScheduled)) {

Source\_SchedulePoint = Wochentag(weekday) + "\_" + TimeBisScheduled;

if (Funktion === "CurrSollTemp" ) {

TemperatureScheduledTemp = getState(path + "." + room + ".Profil-" + RaumProfil + "." + Wochentag(weekday) + "\_" + i + "\_" + "Temp").val;

TemperatureScheduledTemp = Calculate\_SollTemp(TemperatureScheduledTemp, "SetTemp");

if(debug) { log("Routine SelectSwitchTime - Aktuelle Solltemperatur ist " + TemperatureScheduledTemp + " fuer Raum " + room + " Raumprofil ist " + RaumProfil, "info"); }

return TemperatureScheduledTemp; //exakte Schaltzeit gefunden - also raus jetzt

} // endif currsolltemp

if (Funktion === "CurrSlot" || Funktion === "CurrSlotEnde") {

TemperatureScheduledSlot = Wochentag(weekday) + "\_" + TimeBisScheduled;

if( Funktion === "CurrSlot") { // errechne curr slot ende in Zeit WW\_hh:mm:ss

if(debug) { log("Routine SelectSwitchTime - Aktueller Zeitslot ist " + TemperatureScheduledSlot + " fuer Raum" + room + " Raumprofil ist " + RaumProfil, "info"); }

return TemperatureScheduledSlot; // exakte Schaltzeit gefunden - also raus jetzt

}

if (Funktion === "CurrSlotEnde") { // errechne currslot ende in millisekunde

var schedArr = TemperatureScheduledSlot.split("\_");

MilliSekSlot = addTime(schedArr[1]).getTime();

if (schedArr[1] === "00:00:00") {

MilliSekSlot += 24 \* 60 \* 60 \* 1000;

}

//MilliSekSlot = parseInt(TemperatureScheduledSlot.substr(3, 2), 10) \* 1000 \* 60 \* 60; // Stunden

//MilliSekSlot = MilliSekSlot + parseInt(TemperatureScheduledSlot.substr(6, 2), 10) \* 1000 \* 60; // Minuten addieren

//MilliSekSlot = MilliSekSlot + parseInt(TemperatureScheduledSlot.substr(9, 2), 10) \* 1000; // Sekunden addieren

//MilliSekSlot = MilliSekSlot + heute.getTime();

return MilliSekSlot;

}

break;

} // endif Currslot

if (Funktion === "NextSollTemp" || Funktion === "NextSlot" ) {

NextSchedulePointer = i; // merken der slotnummer

break; // schleife verlassen

}

} // endif timeInRange

} // endfor

if (NextSchedulePointer < 6 ) { // Wenn noch mindestens ein Slot am Selben Tag zur Verfügung steht

NextSchedulePointer = NextSchedulePointer + 1;

}

if (TimeBisScheduled === "00:00:00") { // Ende der Planungszeit des Tages erreicht Tag muss gewechselt werden

if (weekday < 6) { // Nicht Samstag und Nicht Feiertag

weekday = weekday + 1;

NextSchedulePointer = 1;

}

if (weekday === 6) { // 6 = Samstag

weekday = 0; // 0 = Sonntag

NextSchedulePointer = 1;

}

if (weekday === 7) { // 7 = Feiertag

if (new Date().getDay() === 6) { // 1 = Morgen ist Sonntag

weekday = 0; // also auf Sonntag gehen

NextSchedulePointer = 1;

}

else {

weekday = new Date().getDay() + 1;

NextSchedulePointer = 1;

}

}

}

if (NextSchedulePointer >= 6) {

weekday = weekday + 1; // der letzte Slot war erreicht also Tageswechsel

NextSchedulePointer = 1; // next slot ist der erste

} // endif weekday

if (NextSchedulePointer === 1 ) { // Es liegt ein Tageswechsel vor

weekday = DetermineSchedule(room, weekday, RaumProfil,true); // weekday wird uebersteuert, wenn ein "WieVortag" vorkommt

}

else {

weekday = DetermineSchedule(room, weekday, RaumProfil,false); // weekday wird uebersteuert, wenn ein "WieVortag" vorkommt

}

TimeBisScheduled = getState(path + "." + room + ".Profil-" + RaumProfil + "." + Wochentag(weekday) + "\_" + NextSchedulePointer + "\_" + "bis").val; // die BIS-Zeit lesen

if (Funktion === "NextSollTemp") {

NextTemperatureScheduledTemp = getState(path + "." + room + ".Profil-" + RaumProfil + "." + Wochentag(weekday) + "\_" + NextSchedulePointer + "\_" + "Temp").val;

NextTemperatureScheduledTemp = Calculate\_SollTemp(NextTemperatureScheduledTemp, "SetTemp");

if(debug) { log("Routine SelectSwitchTime - Nächste Solltemperatur ist " + NextTemperatureScheduledTemp + " fuer Raum" + room + " Raumprofil ist "+ RaumProfil, "info"); }

return NextTemperatureScheduledTemp; //Next Schaltzeit gefunden - also raus jetzt

} // endif next Solltemp

if (Funktion === "NextSlot") {

NextTemperatureScheduledSlot = Wochentag(weekday) + "\_" + TimeBisScheduled;

if(debug) { log("Routine SelectSwitchTime - Nächster Zeitslot ist " + NextTemperatureScheduledSlot + " fuer Raum" + room + " Raumprofil ist "+ RaumProfil , "info"); }

return NextTemperatureScheduledSlot; //Next Schaltzeit gefunden - also raus jetzt

} // endif next slot

return 0; // da ist was schiefgelaufen. es haette einen schedule geben sollen

} // ende der Function

//-----------------------------------------------------------------------------------------------------

// Funktion um den richtigen Planungstag zu finden bei Verwendung von WieVortag im schedule

//-----------------------------------------------------------------------------------------------------

function DetermineSchedule(room,weekday,Profil,FeiertagMorgenChck) {

if (FeiertagMorgenChck === undefined) {

FeiertagMorgenChck = false;

}

var currentday = weekday;

var prePath = path + "." + room + ".Profil-" + Profil + ".";

var StateUrlaubWieFeiertag = prePath + "ProfilParameter\_UrlaubWieFeiertag"; // Profilparameter Urlaub = Feiertag gesetzt ?

if (getState(StateUrlaubAnwesend).val && getState(StateUrlaubWieFeiertag).val ) { // Heute ist ein Urlaubstag und soll wie ein Feiertag behandelt werden

weekday = 7; // Urlaub ist wie Feiertag also den Feiertagsschedule setzen

}

if(UseFeiertagskalender) {

if (!FeiertagMorgenChck& getState(StateFeiertagHeute).val ) { // Heute ist Feiertag

weekday = 7; // Urlaub ist wie Feiertag also den Feiertagsschedule setzen

}

if (FeiertagMorgenChck & getState(StateFeiertagMorgenAdapter).val ) { // Heute ist Feiertag

weekday = 7; // Urlaub ist wie Feiertag also den Feiertagsschedule setzen

}

}

if (weekday === 1) { // der schedule fuer Montag hat nie einen "wie Vortag"

return weekday;

}

var TempWieVortag = getState(prePath + Wochentag(weekday) + "\_" + "wieVortag").val; // Wenn es keinen "WieVortag gib, dann kann der Tag genommen werden"

if (TempWieVortag === false) {

return weekday;

}

if (weekday === 0 ) { // Sonntag

TempWieVortag = getState(prePath + Wochentag(0) + "\_" + "wieVortag").val;

if (TempWieVortag === false) {

if (debug) {log("Routine DetermineSchedule: Zu planedner Tag ist = " + Wochentag(weekday) + " Tag fuer den Schedule ist = " + Wochentag(0),"info"); }

return(0); // Sonntagschedule hat keinen "WieVortag"-tick

}

else {

weekday = 6; // Sonntagschedule hat einen "WieVortag"-tick also ist geht es jetzt mit Samstag weiter

}

}

// Erstmal bei Feiertag schauen ob der Sonntag-Schedule der richtige ist

if (weekday === 7 ) { // Feiertag

TempWieVortag = getState(prePath + Wochentag(0)+ "\_" + "wieVortag").val; // Hat der folgende Sonntag einen WieVortag Tick ?

if (TempWieVortag === false) { // hater er nicht

if (debug) { log("Routine DetermineSchedule: zu planender Tag ist = " + Wochentag(weekday) + " Tag fuer den Schedule ist = " + Wochentag(0),"info"); }

return(0); // Sonntagschedule hat keinen "WieVortag"-tick

}

else {

weekday = 6; // Sonntagschedule hat einen "WieVortag"-tick also ist geht es jetzt mit Samstag weiter

}

}

// da Feiertag und Sonntag schon klar sind koennen die Tage von Samstag bis Montag (6-1) abgearbeitet werden

for (var i = weekday; i > 0; i--) {

TempWieVortag = getState(prePath + Wochentag(i)+ "\_" + "wieVortag").val;

if (TempWieVortag === false) {

if (debug) { log("Routine DetermineSchedule: zu planender Tag ist = " + Wochentag(currentday) + " Tag fuer den Schedule ist = " + Wochentag(i),"info"); }

return i; // Sonntagschedule ist richtig

}

}

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

//Globale Parameter aus ICAL Feiertagskalender und Anwesenheitserkennung setzen

// Globale Parameter aus ICAL: Urlaub Anwesend, Urlaub Abwesend, Party, Gaeste, Feiertage

// Wenn events genutzt werden, dann koennen globale Parameter nicht mehr manuell eingestellt werden

//-----------------------------------------------------------------------------------------------------

function SetEventGlobalParameter() {

if (getState(Gparameterpath + ".ICAL-Events\_Aktiv").val === false) {

return; // global parameter events sind ausgeschaltet

}

if (getState(ICALPath + "."+ EventG\_Party).val) {

setState(StatePartyjetzt, true);

Source\_ICALEvent = EventG\_Party;

return;

}

else {

setState(StatePartyjetzt, false);

}

if (getState(ICALPath + "."+ EventG\_UrlaubAbwesend).val) {

setState(StateUrlaubAbwesenheit, true);

Source\_ICALEvent = EventG\_UrlaubAbwesend;

return;

}

else {

setState(StateUrlaubAbwesenheit, false);

}

if (getState(ICALPath + "."+ EventG\_UrlaubAnwesend).val) {

setState(StateUrlaubAnwesend, true);

Source\_ICALEvent = EventG\_UrlaubAnwesend;

} else {

setState(StateUrlaubAnwesend, false);

}

if (getState(ICALPath + "."+ EventG\_Gaeste).val) {

setState(StateGaesteDa, true);

Source\_ICALEvent = EventG\_Gaeste;

return;

} else {

setState(StateGaesteDa, false);

}

if (UseFeiertagskalender === false) { // Wenn der Feiertagsadapter genutzt wird werden Events fuer Feiertage nicht genutzt

if (getState(ICALPath + "."+ EventG\_Feiertag).val) {

setState(StateFeiertagHeute, true);

Source\_ICALEvent = EventG\_Feiertag;

return;

}

else {

setState(StateFeiertagHeute, false);

} // endif check ICAL

} // endif feiertagskalender nicht aktiv

} // Ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion Finde Aktives Raumprofil

//-----------------------------------------------------------------------------------------------------

function ActiveProfile (room,Profil) {

if(Profil === undefined) {

Profil = 1;

}

var pathprofile = path + "." + room;

var ProfilName;

if (MaxProfile === 1) {

Profil = 1;

Source\_Profil = 1;

return Profil;

}

// jetzt ICAL Profil auswählen wenn ICAL für Raumprofile aktiviert ist

var ICALProfilSelect = getState(Gparameterpath+".ICAL-Events\_Aktiv").val

if (ICALProfilSelect === true ) { // Events aktiv ?

// Erst Raumprofil checken - prio1

for (var i = MaxProfile; i > 0; i--) {

ProfilName = UseEventR\_Profil;

ProfilName = UseEventR\_Profil.replace("<Raumname>", room);

ProfilName = ProfilName.replace("<ProfilNummer>", i);

const PN = ICALPath + "." + ProfilName;

if(getObject(PN)) { // Existiert der DP ? - Lange Liste Warnung im log vermeiden, falls nicht vorhanden

if (getState(ICALPath + "." + ProfilName).val) {

setOwnState(pathprofile + ".AktivesEventProfil", i); // Eventprofil update

setOwnState(path + "." + room + ".AktivesRaumProfil", i); // Aktives Profil Update

Source\_Profil = i;

Source\_ICALEvent = ProfilName;

return i;

}

}

} // ende for i

// Globales Profil ist prio2

for (var i = MaxProfile; i > 0; i--) {

ProfilName = UseEventG\_Profil;

ProfilName = UseEventG\_Profil.replace("<ProfilNummer>", i);

const PN = ICALPath + "." + ProfilName;

if(getObject(PN)) { // Existiert der DP ? - Lange Liste Warnung im log vermeiden, falls nicht vorhanden

if (getState(ICALPath + "." + ProfilName).val) {

setOwnState(pathprofile+".AktivesEventProfil", i); // Eventprofil update

setOwnState(path + "." + room + ".AktivesRaumProfil", i); // Aktives Profil Update

Source\_Profil = i;

Source\_ICALEvent = ProfilName;

return i;

}

}

} // ende for i

} // ende if ICAL ProfilSelect

if (getState(pathprofile + ".AktivesEventProfil").val !== 0 && ICALProfilSelect === true) { // falls vorher ein Eventprofil aktiv war jetzt deaktivieren

setOwnState(pathprofile + ".AktivesEventProfil", 0);

setOwnState(path + "." + room + ".AktivesRaumProfil", 1); // Aktives Profil Update

return 1;

}

// Wenn kein anderes Profil vorliegt dann gilt das manuell eingstellte Profil

Profil = getState(pathprofile + ".AktivesRaumProfil").val;

if(Profil > MaxProfile || Profil === 0 ) {

Profil = 1;

setOwnState(path + "." + room + ".AktivesRaumProfil", 1); // Aktives Profil Update

}

return Profil;

}

//-----------------------------------------------------------------------------------------------------

// Funktion SetTemp zum setzen einer neuen Solltemperatur

//-----------------------------------------------------------------------------------------------------

function SetTemp(room, SollTemp, id, AdjustLastTemp) {

SetDetData(room); // Speichere Findungsdaten

if (getState(id).val !== SollTemp) { // ist die SET-Temperature unterschiedlich zur errechneten Solltemperatur ?

setOwnState(id, SollTemp);

if (debug) { log ("Routine SetTemp: ID updated " + id+" Raum " + room + " SollTemp = " + SollTemp, "info"); }

Source\_CurrentSollTemp = SollTemp;

}

if (AdjustLastTemp && getState(path + "." + room + ".Source\_Last\_Temp").val !== SollTemp) { // LastSollTemp synhronisieren

setOwnState(path + "." + room + ".Source\_Last\_Temp", SollTemp);

if (debug) {log(' Setze ' + room + ".Source\_Last\_Temp zu " + SollTemp);}

} // endif

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion SyncThermostat - bei manuellen Aenderungen werden alle Thermostate des Raumes gesynched

//-----------------------------------------------------------------------------------------------------

function SyncThermostat(room, Action, SollTemp, Source\_id) {

var id;

for (var x in ControlTab) {

id = ControlTab[x][1];

if (ControlTab[x][0] === room && Action === "ExecuteTemp" && Source\_id !== id) { // Raum selektieren und alle abhaengingen Thermostate (mit 1 gekennzeichnet)

if (debug) {log("Routine SyncThermostat: Temperatur wird nach schedule synchronisiert fuer id " + id ); }

ExecuteTempDetermination(room,id);

} // endif setzte Temp entsprechend schedule

if (ControlTab[x][0] === room && Action === "ManualTemp" && Source\_id !== id ) { // Raum selektieren und alle abhaengingen Thermostate (mit 1 gekennzeichnet)

if (debug) {log("Routine SyncThermostat: Temperatur " + SollTemp + " wird synchronisiert fuer id " + id ); }

SetTemp(room, SollTemp, id, false);

} // endif setze Manuelle Temperatur

} // endfor

} // Ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion CreateStates zum Anlegen der Datenpunkte

//-----------------------------------------------------------------------------------------------------

function CreateStates(room, DeviceType) {

room = room.replace(/\s/g, "\_"); // Alle Leerzeichen aus der Raumbezeichnung entfernen

// Globale Parameter

createState(StateAnwesenheit, false, {read: true, write: true, type: 'boolean', name: 'Anwesenheitsflag', desc: 'Anwesenheitsflag'});

createState(StateFeiertagHeute, false, {read: true, write: true, type: 'boolean', name: 'Feiertag Heute', desc: 'Feiertag Heute- zur Temperaturanpassung (wie Sonntag)'});

createState(StatePartyjetzt, false, {read: true, write: true, type: 'boolean', name: 'Party Jetzt', desc: 'Party Jetzt- zur Temperaturanpassung'});

createState(StateGaesteDa, false, {read: true, write: true, type: 'boolean', name: 'Gaeste da', desc: 'Gaeste da - zur Temperaturanpassung'});

createState(StateUrlaubAnwesend, false, {read: true, write: true, type: 'boolean', name: 'Urlaub Anwesend', desc: 'Urlaub Heute- zur Temperaturanpassung (wie Sonntag)'});

createState(StateUrlaubAbwesenheit, false, {read: true, write: true, type: 'boolean', name: 'Urlaub Abwesend', desc: 'Urlaub und nicht zu Hause)'});

createState(StateHeizperiode, true, {read: true, write: true, type: 'boolean', name: 'Wenn Heizperiode dann Aktivierung der Heizplaene', desc: 'Ausserhalb der Heizperiode werden Ventile geschlossen'});

State = Gparameterpath + ".Source\_last\_Program\_Run";

createState(State, "init", {read: true, write: false, type: 'string', name: 'Datum/Zeit des letzten Programmlaufes' , desc: 'Datum/Zeit des letzten Programmlaufes'});

State = Gparameterpath + ".ICAL-Events\_Aktiv";

createState(State, false, {read: true, write: false, type: 'boolean', name: 'Global ICAL Event aktiv' , desc: 'Uebersteuert Skripteinstellung'});

// Anlegen der raumbezogenen/profilbezogenen Datenpunkte

var State;

var RoomPath = path + "." + room + ".";

var Profilpath;

var Vorgabewert1 = Calculate\_SelectValueWert(1, "CorrectTemp");

var Vorgabewert2 = Calculate\_SelectValueWert(2, "CorrectTemp");

var Vorgabewert17 = Calculate\_SelectValueWert(17, "SetTemp");

var bisTime;

var SollTemp;

var StateBis;

var StateTemp;

var StateWieVortag;

State = RoomPath + "Source\_last\_Program\_Run";

createState(State, "init", {read: true, write: false, type: 'string', name: 'Datum/Zeit des letzten Programmlaufes für diesen Raum' , desc: 'Datum/Zeit des letzten Programmlaufes für diesen Raum'});

for (var y = 1; y <= MaxProfile; y++) { // es werden alle Daten je profil angelegt

Profilpath = "Profil-" + y +".";

for (var i = 1; i <= 6; i++) { // es werden 6 BIS Zeiten angelegt

for (var x = 0; x <= 7; x++) { // es wird ein Plan je Wochentag und ein Feiertag angelegt

if (i <= 6) { // die 6. Zeit hat keine BIS Zeit - nur Temperatur erforderlich

StateBis = RoomPath + Profilpath + Wochentag(x) + "\_" + i +"\_" + "bis";

}

StateTemp = RoomPath + Profilpath + Wochentag(x) + "\_"+ i +"\_" + "Temp";

if (i===1) { bisTime = "06:00:00"; SollTemp = Calculate\_SelectValueWert(17,"SetTemp");}

if (i===2) { bisTime = "08:00:00"; SollTemp = Calculate\_SelectValueWert(21,"SetTemp");}

if (i===3) { bisTime = "16:00:00"; SollTemp = Calculate\_SelectValueWert(18,"SetTemp");}

if (i===4) { bisTime = "21:00:00"; SollTemp = Calculate\_SelectValueWert(21,"SetTemp");}

if (i===5) { bisTime = "23:30:00"; SollTemp = Calculate\_SelectValueWert(19,"SetTemp");}

if (i===6) { bisTime = "00:00:00"; SollTemp = Calculate\_SelectValueWert(17,"SetTemp");}

createState(StateBis ,bisTime , {read: true, write: true, type: 'string', name: 'Zeit ' + i + ' von' , desc: 'Zeit von'});

createState(StateTemp ,SollTemp , {read: true, write: true, type: 'number', name: 'Solltemperatur '+ i , desc: 'Solltemperatur'});

StateWieVortag = RoomPath + Profilpath + Wochentag(x)+ "\_" + "wieVortag";

createState(StateWieVortag, false, {read: true, write: true, type: 'boolean', name: 'Wie Vortag ' , desc: 'Wie Vortag'});

} // endfor y 6 Zeiten bzw Temperatur je schedule

} // endfor i 6 BIS Zeiten

State = RoomPath + Profilpath + StatePP\_PartyAbsenkung;

createState(State, Vorgabewert2 , {read: true, write: true, type: 'number', name: 'Absenkung bei Party in Grad Celsius' , desc: 'Absenkung bei Party - negativer Wert = Anhebung'});

State = RoomPath + Profilpath + StatePP\_GaesteAnhebung;

createState(State, Vorgabewert1 , {read: true, write: true, type: 'number', name: 'Anhebung, wenn Gaeste anwesend' , desc: 'Anhebung bei Gaesten - negativer Wert = Absenkung'});

State = RoomPath + Profilpath + StatePP\_AbwesenheitAbsenkung;

createState(State, Vorgabewert1 , {read: true, write: true, type: 'number', name: 'Absenkung bei Abwesenheit in Grad Celsius' , desc: 'Absenkung bei Abwesenheit - negativer Wert = Anhebung'});

State = RoomPath + Profilpath + StatePP\_UrlaubAbsenkung;

createState(State, Vorgabewert2 , {read: true, write: true, type: 'number', name: 'Absenkung bei Urlaubs-Abwesenheit in Grad Celsius' , desc: 'Absenkung bei Urlaubs-Abwesenheit - negativer Wert = Anhebung'});

State = RoomPath + Profilpath + StatePP\_UrlaubWieFeiertag;

createState(State, true, {read: true, write: true, type: 'boolean', name: 'bei Anwesenheit wg Urlaub Temperaturen laut Feiertagsplan' , desc: 'Temperaturen laut Feiertagsplan'});

State = RoomPath + Profilpath + "ProfilParameter\_MinimaleTemperatur";

createState(State, Vorgabewert17 , {read: true, write: true, type: 'number', name: 'Minimale Temperatur fuer Absenkung' , desc: 'Minimale Temperatur fuer Absenkung'});

} // endfor y // Daten je Profil

// RaumParameter

State = RoomPath + "AktivesRaumProfil";

createState(State, 1, {read: true, write: true, type: 'number', name: 'Aktives EventProfil 1-9' , desc: 'Fuer jeden Raum koennen max 9 Profile verwendet werden'});

State = RoomPath + "AktivesEventProfil";

createState(State, 0, {read: true, write: true, type: 'number', name: 'Aktives EventProfil 1-9' , desc: 'Fuer jeden Raum koennen max 9 Profile verwendet werden'});

State = RoomPath + "Source\_Profil";

createState(State, 0, {read: true, write: true, type: 'number', name: 'ermitteltes Profil fuer die letzte RaumTemperatur' , desc: 'Fuer jeden Raum koennen max 9 Profile verwendet werden'});

State = RoomPath + "Source\_ICALEvent";

createState(State, "init", {read: true, write: true, type: 'string', name: 'ermitteltes ICAL Event fuer die Profilfindung / Parameterermittlung' , desc: 'Profil oder Parameteranpassung'});

State = RoomPath + "Source\_Schedule";

createState(State, "init", {read: true, write: true, type: 'string', name: 'der Schedulepunkt fuer die Solltemperatur' , desc: 'Der Schedule ist die taegliche Temperaturplanung'});

State = RoomPath + "Source\_NextTemp";

createState(State, 0, {read: true, write: false, type: 'number', name: 'nächste geplante Temperatur' , desc: 'nächste geplante Temperatur'});

State = RoomPath + "Source\_Global\_Parameter";

createState(State, "init", {read: true, write: true, type: 'string', name: 'Globaler Parameter, der die Temperaturfindung beeinflusst' , desc: 'Global Parameter wie Party, Urlaub, Gaeste etc'});

State = RoomPath + "Source\_Manually\_Adjusted";

createState(State, 0, {read: true, write: true, type: 'number', name: 'Manuell eingestellte Temperatur' , desc: 'Solltemperatur der manuellen Verstellung'});

State = RoomPath + "Source\_Last\_Temp";

createState(State, 0, {read: true, write: true, type: 'number', name: 'letzte eingestellte Temperatur' , desc: 'Letzte Solltemperatur'});

State = RoomPath + "Source\_TimeStamp";

createState(State, "init", {read: true, write: true, type: 'string', name: 'Datum und Zeit der letzten Tempanpassung' , desc: 'Zeitstempel'});

State = RoomPath + "View\_Manually\_Adjusted";

createState(State, 0, {read: true, write: true, type: 'number', name: 'Im View manuell eingestellte Temperatur - 0=reset' , desc: 'Im View manuell eingestellte Temperatur - 0=reset'});

State = RoomPath + "View\_Manual\_Temp\_Duration";

createState(State, 120, {read: true, write: true, type: 'number', name: 'Gueltigkeit in Minuten fuer manuelle Temperatur Aenderung' , desc: 'Zeitgueltigkeit in Minuten'});

State = RoomPath + "View\_ManTemp\_Validity";

createState(State, "init", {read: true, write: true, type: 'string', name: 'Datum und Zeit der Gueltigkeit (bis) zur Rueckkehr zum Plan' , desc: 'Zeitgueltigkeit bis (Datum / Zeit'});

State = RoomPath + "RaumParameter\_ManuellModeForce";

createState(State, true, {read: true, write: true, type: 'boolean', name: 'Immer Umschaltung auf manuell (nur neue Thermsotate)' , desc: 'Immer Umschaltung auf manuell (nur neue Thermsotate'});

State = RoomPath + "RaumStatusVerschluss";

createState(State, true, {read: true, write: true, type: 'boolean', name: 'Verschlussssstatus des Raumes)' , desc: 'Verschlussstatus true=geoeffnet - false = geschlossen'});

State = RoomPath + "ICALProfilSelect";

createState(State, true, {read: true, write: true, type: 'boolean', name: 'ICAL Auto-Select aktivieren/deaktivieren' , desc: 'True = ICAL selektiert Profile / false es kann manuell ein Profil gewaehlt werden'});

State = RoomPath + "SoftBoostActive";

createState(State, false, {read: true, write: true, type: 'boolean', name: 'Soft-Boost aktiviertn' , desc: 'True = Soft-Boost Aktiv / false Soft-Boost deaktiviert'});

State = RoomPath + "SoftBoostDuration";

createState(State, 60, {read: true, write: true, type: 'number', name: 'Dauer in Minuten fuer wenn Soft-Boost aktiviert wird' , desc: 'Im View manuell eingestellte Dauer von SoftBoost'});

State = RoomPath + "SoftBoostValidity";

createState(State, "init", {read: true, write: true, type: 'string', name: 'Datum und Zeit der Gueltigkeit (bis) Soft-Boost beendet wird imd Rueckkehr zum Plan' , desc: 'Zeitgueltigkeit bis (Datum / Zeit'});

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

//ab hier Nebenfunktionen

//-----------------------------------------------------------------------------------------------------

//-----------------------------------------------------------------------------------------------------

// Funktion SetRoomClosed setzt einen Zeitstempel, wenn der letzte Sensor geschlossen meldet

//-----------------------------------------------------------------------------------------------------

function SetRoomClosed (room, delay) {

if (delay === 0 ) { // delay in Minuten aus ThermTabType Tabelle

return;

}

log("SetRoomClosed gestartet")

var DelayDuration = delay\*60000;

var DelayTime = new Date().getTime()+DelayDuration; // DelayTime = jetzt plus Anzahl Millisekunden (umgerechnet von Minuten) aus der TermostatTypeTab Einstellungen

for (var x in DelayAfterClose) {

if (DelayAfterClose[x][0] === room) {

if(DelayAfterClose[x][1] < DelayTime) {

DelayAfterClose[x][1] = new Date().getTime() + DelayDuration; // Zeitstempel setzen und die gewünschte anzahl Minuten hinzurechnen aus ThermTabType Tabelle

ManTimeouts[room] = setTimeout(TriggerUpdate, DelayDuration, room, true);

log("Routine SetRoomClosed: Zeitstempel war gesetzt für geschlossenen Raum " + room + "neuer Zeitstempel = " + formatDate(DelayAfterClose[x][1], "YYYY/MM/DD SS:mm:ss"),"info");

return;

} else {

return; // wert ist noch gültig

} // endif DelayAfterClose

} // endif room gefunden

} // endfor

// Eintrag gibt es noch nicht also hinzufügen

DelayAfterClose.push([room, DelayTime]);

ManTimeouts[room] = setTimeout(TriggerUpdate, DelayDuration, room, false);

if (debug) {

log("Routine SetRoomClosed: Zeitstempel gesetzt für geschlossenen Raum " + room + " Zeitstempel = " + formatDate(DelayTime, "YYYY/MM/DD SS:mm:ss"));

}

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion CheckDelay checked ob ein Delay erforderlich ist für eine evt Anpassung der SollTemp

//-----------------------------------------------------------------------------------------------------

function CheckDelay (room) {

var ActTime = formatDate(new Date(),"YYYY/MM/DD SS:mm:ss"); // Formatierte Aktuelle Zeit

var ActTimeMilliSek = new Date(ActTime).getTime(); // Aktuelle Zeit in Millisekunden seit 1970

for (var x in DelayAfterClose ) {

if (DelayAfterClose[x][0] === room) {

if(ActTimeMilliSek > DelayAfterClose[x][1] && DelayAfterClose[x][1] > 0) { // DelayTime abgelaufen

log("Routine CheckDelay: delay für Raum " + room + " abgelaufen", "info" );

DelayAfterClose[x][1] = 0;

return 0;

}

if (debug) { log("Routine CheckDelay: delay für Raum " + room + " ist vorhanden bis " + DelayAfterClose[x][1] + " millisekunden ", "info" ); }

return DelayAfterClose[x][1]; // melde die Ablaufzeit

} // endif room gefunden

} // endfor

return 0; // kein Delay

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion Check\_SensorDV checkt ob eine Direktvernuepfung von Sensor zu Thermostat vorliegt im Raum

//-----------------------------------------------------------------------------------------------------

// function Check\_SensorDV (room) {

// for (var i in SensorList ) {

// if (SensorList[i][0] === room && SensorList[i][9] === true) { // Sensor ist direktverknuepft

// return true;

// }

// }

// return false;

// } // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion Check\_ThermDV checkt ob eine Direktvernuepfung von Thermostat zu Thermostat (Gruppen) vorliegt im Raum

//-----------------------------------------------------------------------------------------------------

function Check\_ThermDV(room) {

for (var i in ControlTab ) {

if (ControlTab[i][0] === room && ControlTab[i][7] === true) { // Thermostat ist direktverknuepft

return true;

}

}

return false;

} //ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion InitilizeManChange - Ruecksetzen der manuellen Daten - Manuelle Temperatur

//-----------------------------------------------------------------------------------------------------

function InitilizeManChange(room) {

setOwnState(path + "." + room + ".Source\_TimeStamp", "init");

setOwnState(path + "." + room + ".View\_ManTemp\_Validity", "init");

setOwnState(path + "." + room + ".Source\_Manually\_Adjusted", 0) ;

setOwnState(path + "." + room + ".View\_Manually\_Adjusted", 0) ;

} // endfunction

//-----------------------------------------------------------------------------------------------------

// Funktion SortControlTab - Sortieren der ControlTab nach Raumnamen

//-----------------------------------------------------------------------------------------------------

function SortControlTab(a,b) {

a = a[0];

b = b[0];

return a == b ? 0 : (a < b ? -1 : 1);

} // endfunction

//-----------------------------------------------------------------------------------------------------

// Funktion check Room in roomlist -

//-----------------------------------------------------------------------------------------------------

function ChckRoom(room) {

var z;

for (z in ControlTab) {

if (ControlTab[z][0] === room) {

if (debug) {log("Routine ChckRoom - Dem Raum " + room + " ist ein Gerät zugeordnet ", "info"); }

return true;

}

}

for (z in NoneHMTab) { // auch nicht HM checken

if (NoneHMTab[z][0] === room) {

if (debug) {log("Routine ChckRoom - Dem Raum " + room + " ist ein Gerät zugeordnet ", "info"); }

return true;

}

}

for (z in NoneHMSenorTab) { // auch nicht HM checken

if (NoneHMSenorTab[z][0] === room) {

if (debug) {log("Routine ChckRoom - Dem Raum " + room + " ist ein Gerät zugeordnet ", "info"); }

return true;

}

}

return false;

} // ende funktion

//-----------------------------------------------------------------------------------------------------

// Funktion SensorFind Sucht den Sensor aus der SensorTabelle

//-----------------------------------------------------------------------------------------------------

function SensorFind(id) {

for (var i in SensorList) {

if (debug) { log("Routine SensorFind ID = " + SensorList[i][1] + " Raum = " + SensorList[i][0], "info"); }

if (SensorList[i][1] === id) {

return i;

}

}

log("Routine SensorFind: sensor nicht gefunden " + id);

return 999;

}

//-----------------------------------------------------------------------------------------------------

//Function VerschlussRaumStatus

// ermittelt ob irgend ein Verschluss auf offen steht

//-----------------------------------------------------------------------------------------------------

function VerschlussRaumStatus(room) {

var count = 0;

for (var x in SensorList) {

if (SensorList[x][0].toString() === room) {

count = count + 1;

if (SensorList[x][7] === true) {

return true; // mindestens ein Sensor steht auf offen

}

}

}

return false; // alle Sensoren auf geschlossen

} // Ende Funktion

//-----------------------------------------------------------------------------------------------------

//-----------------------------------------------------------------------------------------------------

// Funktion SaveStatus - Speichert den Raumstatus

// Type = Type der Meldung: Heizplan, Fenster, HeizperiodeAus, Abwesend, Party, Gaeste, Urlaub Anwesend, Urlaub Abwesend,

// room = auf den Raum bezogen

// MindestTempflag = Mindesttemperatur gesetzt - true/false

//-----------------------------------------------------------------------------------------------------

function SaveStatus(Type,room, MindestTemp) {

if(Type === "Heizplan") {

Source\_GlobalParameter = ""; // es wird nach Heizplan geheizt - keine Meldung

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Ende Heizplan

if(Type === "Fenster") {

Source\_GlobalParameter = "Absenkung - Verschluss geoeffnet";

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Endif Fenster

if(Type === "HeizperiodeAus") {

Source\_GlobalParameter = "keine Heizperiode - Heizplan wird nicht ausgeführt";

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Ende Heizperiode

if(Type === "Abwesend") {

if(MindestTemp === true) {

Source\_GlobalParameter = "Absenkung Abwesenheit auf MindestTemperatur";

} else {

Source\_GlobalParameter = "Absenkung Abwesenheit";

}

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Ende Abwesend

if(Type === "Party") {

if(MindestTemp === true) {

Source\_GlobalParameter = "Temperaturanpassung auf MindestTemperatur- Partyflag gesetzt";

} else {

Source\_GlobalParameter = "Temperaturanpassung - Partyflag gesetzt";

}

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Ende Party

if(Type === "Gaeste") {

if(MindestTemp === true) {

Source\_GlobalParameter = "Temperaturanpassung auf MindestTemperatur- Gaeste-Flag gesetzt";

} else {

Source\_GlobalParameter = "Temperaturanpassung - Gaeste-Flag gesetzt";

}

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Ende Gaeste

if(Type === "UrlaubAnwesend") {

Source\_GlobalParameter = "Urlaub anwesend - Plannung ist Feiertag";

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Ende Urlaub anwesend

if(Type === "UrlaubAbwesend") {

if(MindestTemp === true) {

Source\_GlobalParameter = "Temperaturanpassung auf MindestTemperatur- Urlaub Abwesend";

} else {

Source\_GlobalParameter = "Temperaturanpassung - Urlaub Abwesend";

}

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Ende Urlaub Abesend

if(Type === "Manuell") {

Source\_GlobalParameter = "Manuelle Temperaturanpassung";

if (getState(path + "." + room + ".Source\_Global\_Parameter").val !== Source\_GlobalParameter) {

setOwnState(path + "." + room + ".Source\_Global\_Parameter", Source\_GlobalParameter);

if (debug) {log(' Setze ' + room + ".Source\_Global\_Parameter zu " + Source\_GlobalParameter);}

} // Endif update notwendig

} // Ende Manuell

} // Ende Function SaveStatus

//-----------------------------------------------------------------------------------------------------

// Funktion SetDetData - Schreibt die Findungsdaten in die source states

//-----------------------------------------------------------------------------------------------------

function SetDetData (room) {

if (typeof(Source\_ICALEvent) === "undefined" ) {

Source\_ICALEvent = "";

}

if (typeof(Source\_Profil) === "undefined" ) {

Source\_Profil = 99;

}

if (typeof(Source\_SchedulePoint) === "undefined" ) {

Source\_SchedulePoint = "";

}

if (getState(path + "." + room + ".Source\_Profil").val !== Number(Source\_Profil)) {

setOwnState(path + "." + room + ".Source\_Profil", Source\_Profil);

if (debug) {log(' Setze ' + room + ".Source\_Profil zu " + Source\_Profil);}

}

if (getState(path + "." + room + ".Source\_ICALEvent").val !== Source\_ICALEvent) {

setOwnState(path + "." + room + ".Source\_ICALEvent", Source\_ICALEvent);

if (debug) {log(' Setze ' + room + ".Source\_ICALEvent zu " + Source\_ICALEvent);}

}

//if (getState(path + "." + room + ".Source\_ICALEvent").val !== Source\_ICALEvent) {

// setOwnState(path + "." + room + ".Source\_ICALEvent", Source\_ICALEvent);

//}

if (getState(path + "." + room + ".Source\_Schedule").val !== Source\_SchedulePoint) {

setOwnState(path + "." + room + ".Source\_Schedule", Source\_SchedulePoint);

if (debug) {log(' Setze ' + room + ".Source\_Schedule zu " + Source\_SchedulePoint);}

}

if (getState(path + "." + room + ".Source\_NextTemp").val !== Source\_NextSollTemp) {

setOwnState(path + "." + room + ".Source\_NextTemp", Source\_NextSollTemp);

if (debug) {log(' Setze ' + room + ".Source\_NextTemp zu " + Source\_NextSollTemp);}

}

} // Ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion Calculate SollTemperatur Urechnung des Select Value Wertes in Gradzahlen

//-----------------------------------------------------------------------------------------------------

function Calculate\_SollTemp(SollTemp, Calc\_type) {

var MinVal;

var StepVal;

var MaxVal ;

if (VerwendungSelectValue !== true) {

return SollTemp;

}

if (SollTemp === 0) {

return SollTemp;

}

if (Calc\_type === "SetTemp") {

MinVal = 12; // Liste faengt mit 12 an

StepVal = 0.5; // Schrittwert ist 0.5

SollTemp = MinVal + ( SollTemp \*StepVal); // Errechnung der SollTemp

}

if (Calc\_type === "CorrectTemp") {

MinVal = 0; // Liste faengt mit 0 an

MaxVal = 5; // Der Maxvalue ist Positiv und Negativ

StepVal = 0.5;

SollTemp = SollTemp \* StepVal; // Errechnung der SollTemp fuer positive Werte

if (SollTemp > (MaxVal - MinVal)) {

SollTemp = (SollTemp \* -1 +StepVal) + MaxVal; // fuer Negative Werte

}

}

//log("aus Calc\_SollTemp Ausstieg "+ SollTemp)

return SollTemp;

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Function LastRoomUpdateTime

// Findet die letzte Updatezeit eines Raumes oder speichert die letzte update Zeit

// Funktion find und push

//-----------------------------------------------------------------------------------------------------

function LastRoomUpdateTime (room, Funktion) {

// Funktion = "find" gibt die letzte updatezeit in Millisekunden zurück

// Funktion = "push" fügt den Raum und aktuelle Zeit hinzu

var ActTime = formatDate(new Date(),"YYYY/MM/DD SS:mm:ss"); // Formatierte Aktuelle Zeit

var ActTimeMilliSek = new Date(ActTime).getTime(); // Aktuelle Zeit in Millisekunden seit 1970

var y = SubscribeThermBlock.length;

for (var x in SubscribeThermBlock) {

if (SubscribeThermBlock[x][0] === room && Funktion == "find") {

return SubscribeThermBlock [x][1];

}

if (SubscribeThermBlock[x][0] === room && Funktion == "push") {

SubscribeThermBlock[x][1] = ActTimeMilliSek;

return ActTimeMilliSek;

}

}

// wenn es den Raum noch nicht gab

if ( Funktion == "find") {

return 0;

}

if ( Funktion == "push") {

SubscribeThermBlock[y] = [room, ActTimeMilliSek];

return ActTimeMilliSek;

}

} // ende der Function LastRoomUpdateTime

//-----------------------------------------------------------------------------------------------------

// Funktion Wochentag zur ermittlung des Tages

//-----------------------------------------------------------------------------------------------------

function Wochentag(tag) {

if (tag === 0) { tag = "So"; return tag; }

if (tag === 1) { tag = "Mo"; return tag; }

if (tag === 2) { tag = "Di"; return tag; }

if (tag === 3) { tag = "Mi"; return tag; }

if (tag === 4) { tag = "Do"; return tag; }

if (tag === 5) { tag = "Fr"; return tag; }

if (tag === 6) { tag = "Sa"; return tag; }

if (tag === 7) { tag = "Feiertag"; return tag; }

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion Calculate SelectValueList Wert Urechnung des Sollwertes in SelectValue List Wert

//-----------------------------------------------------------------------------------------------------

function Calculate\_SelectValueWert(SelValWert, Calc\_type) {

var MinVal;

var StepVal;

var MaxVal;

if (VerwendungSelectValue !== true) {

return SelValWert;

}

if (Calc\_type === "SetTemp") {

MinVal = 12; // Liste faengt mit 12 an

StepVal = 0.5; // Schrittwert ist 0.5

SelValWert = (SelValWert- MinVal ) / StepVal; // Errechnung des Select Value Wertes

}

if (Calc\_type === "CorrectTemp") {

MinVal = 0; // Liste faengt mit 0 an

StepVal = 0.5; // Schrittwert ist 0.5

MaxVal = 5; // Der Maxvalue ist Positiv und Negativ

if(SelValWert > (MaxVal/StepVal) ) {

SelValWert = (SelValWert \* (1 - StepVal) \* -1) + 10; // Negative Werte

}

else {

SelValWert = (SelValWert / StepVal); // Errechnung des Select Value Wertes positive Werte

}

}

// log ("SelValWert = " + SelValWert,"info");

return SelValWert;

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion Setzt den Findungsnachweis zurueck

//-----------------------------------------------------------------------------------------------------

function ClearSources () {

Source\_Profil = 99;

Source\_ICALEvent = "";

Source\_ManualAdjustment = "";

Source\_LastTemp = "";

Source\_Timestamp = "";

Source\_SchedulePoint = "";

} // ende Funktion

//-----------------------------------------------------------------------------------------------------

// Function UserExitPrep - hat nur die Aufgabe die Routine fuer den Userexit zu ermitteln.

//-----------------------------------------------------------------------------------------------------

function UserExitPrep (id,value) {

if (debug) {log("Routine UserExitPrep aufgerufen " + id + " " + value, "info"); }

for (var x in UserExitTab) {

if (UserExitTab[x][0] === id ) {

UserExit(id, value, UserExitTab[x][1]); // uebergabe id, wert und Routine zum eigentlichen UserExit

return; // eine ID darf nur einmal aufgerufen werden

} // endif

} // endfor

} // Endfunction

//-----------------------------------------------------------------------------------------------------

// Funktion listed die sources im log

//-----------------------------------------------------------------------------------------------------

function LogSources() {

if (debug !== true) {

return;

}

log("Source\_Profil " + Source\_Profil, "info");

log("Source\_ICALEvent " + Source\_ICALEvent, "info");

log("Source\_ManualAdjustment " + Source\_ManualAdjustment, "info");

log("Source\_LastTemp " + Source\_LastTemp, "info");

log("Source\_Timestamp " + Source\_Timestamp, "info");

log("Source\_SchedulePoint " + Source\_SchedulePoint, "info");

log("Source\_last\_Program\_Run " + Source\_last\_Program\_Run, "info");

} // Ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion schreibt einen Logeintrag in das Filesystem

//-----------------------------------------------------------------------------------------------------

function writelog(room,id, Text) {

if (typeof(Source\_CurrentSollTemp) === "undefined") {

Source\_CurrentSollTemp = 0;

}

if (typeof(Source\_ManualAdjustment) === "undefined") {

Source\_ManualAdjustment = 0;

}

if (OnlyChanges && Source\_CurrentSollTemp === 0) {

return;

}

// jetzt evt Kommawerte fuer Excel aufbereiten - Excel mit Komma =, und iobroker = .

var Form\_CurrentSollTemp = Source\_CurrentSollTemp.toString();

Form\_CurrentSollTemp = Form\_CurrentSollTemp.replace(".", ",");

var Form\_Source\_ManualAdjustment = Source\_ManualAdjustment.toString();

Form\_Source\_ManualAdjustment = Form\_Source\_ManualAdjustment.replace(".", ",");

if (LogFlag === true) {

Source\_GlobalParameter = getState(path + "." + room + ".Source\_Global\_Parameter").val;

var logdate = formatDate(new Date(), "TT.MM.JJJJ");

var logtime = formatDate(new Date(), "SS:mm:ss");

if (!fs.existsSync(LogPath)) {

log("Routine writelog: Logfile nicht gefunden - wird angelegt", "info");

var headerLine= "Datum;Uhrzeit;Raum;Geraete-ID;SollTemp gesetzt;Profil;Global-Parameter;Event;Manuelle Temp;Schedule-Point;Bemerkung";

fs.appendFileSync(LogPath, headerLine + "\n"); // Fuege Satz in Datei ein

}

fs.appendFileSync(LogPath, logdate + ";" + logtime + ";" + room + ";" + id + ";" + Form\_CurrentSollTemp + ";" + Source\_Profil + ";" + Source\_GlobalParameter + ";" + Source\_ICALEvent + ";" + Form\_Source\_ManualAdjustment + ";" + Source\_SchedulePoint + ";" + Text + "\n"); // Fuege Satz in Datei ein

} // Ende check on logflag

} // Ende Funktion

//-----------------------------------------------------------------------------------------------------

// Funktion RoomListUsage - Ist der UseRoomList eingeschaltet und befindet sich der Raum in der Liste

//-----------------------------------------------------------------------------------------------------

function RoomListUsage (Room) {

if (UseRoomList === false) {

return true;

}

for (var i in RoomList ) {

if (RoomList[i][0] === Room) {

return true;

}

} // endfor

return false;

} // End Function

//-----------------------------------------------------------------------------------------------------

// 3 Funktionen zum Zeitrange check zur Pruefung ob die Schaltungszeiten erreicht sind

// Autor ist Beatz - uebernommen aus:

// viewtopic.php?f=21&t=1072&p=11167&hilit=isTimeInRange&sid=4dca8ea2c7f9337cdc73a1a9e4824a40#p11167

//-----------------------------------------------------------------------------------------------------

function isTimeInRange(strLower, strUpper) {

if (strLower === null || strUpper === null) {

return;

}

//

var now = new Date();

var lower = addTime(strLower);

var upper = addTime(strUpper);

var inRange = false;

if (upper > lower) {

// opens and closes in same day

inRange = (now >= lower && now <= upper) ? true : false;

} else {

// closes in the following day

inRange = (now >= upper && now <= lower) ? false : true;

}

return inRange;

}

function currentDate() {

//var d = new Date();

//return new Date(d.getFullYear(), d.getMonth(), d.getDate());

var heute = new Date();

heute.setHours(0);

heute.setMinutes(0);

heute.setSeconds(0);

heute.setMilliseconds(0);

return heute;

}

function addTime(strTime) {

var time = strTime.split(':');

var d = currentDate();

d.setHours(parseInt(time[0], 10));

d.setMinutes(parseInt(time[1], 10));

d.setSeconds(parseInt(time[2], 10));

return d;

} // Ende Funktion