Rincewinds Buch

der

Zaubersprüche

aus dem

Feemreich

(mit einigen Kochrezepten)

Band 1

über Stimmen, Wachhunde und

allerley Anderes

Ausgabe 1

Über dieses Büchlein

Oft habe ich in der Vergangenheit die Ohren gespitzt, den Worten der Alten Meister lauschend, wenn diese jungen Novizen die Zauberwelt des Feemreiches näher brachten.

In diesem Büchlein habe ich einige der Sprüche notiert. Es sind unglaublich einfache Sprüche hier ebenso versammelt wie sehr Mächtige und Schwere. Wirf einen Blick hinein und staune in Ehrfurcht vor der Macht der alten Meister, wie diese sich der allumfassenden Macht des Feemreiches bedienen.

Meist sind es nicht meine Sprüche! Aber da ein junger Novize nicht überall seine Ohren haben kann, scheint es mir nicht die schlechteste Idee zu sein, einige von diesen Sprüchen nieder zu schreiben.

So sei jedoch gewarnt, junger Novize, dieses Büchlein mag dir bei dem ein oder anderen Problem gute Hilfe leisten, aber keinesfalls vermag es das Studium der Grundlagen zu ersetzen. Unbedacht angewandt, wirst du ungebetene Geister rufen, die du so schnell nicht wieder los wirst. Nur durch das intensive Studium der übrigen Schriften und durch viel Übung wirst du einst selbst ein Meister werden. Vielleicht schlummert ja sogar ein Großmeister in dir, der durch die Kraft seiner Worte das Feemreich mit neuen Zaubersprüchen beschenken wird.

Findest du Fehler oder vermisst du den ein oder anderen Spruch, so lass es mich wissen. So wird es eine neue Auflage, oder gar einen weiteren Band mit neuen Sprüchen geben.

Wie die Sprüche anzuwenden sind:

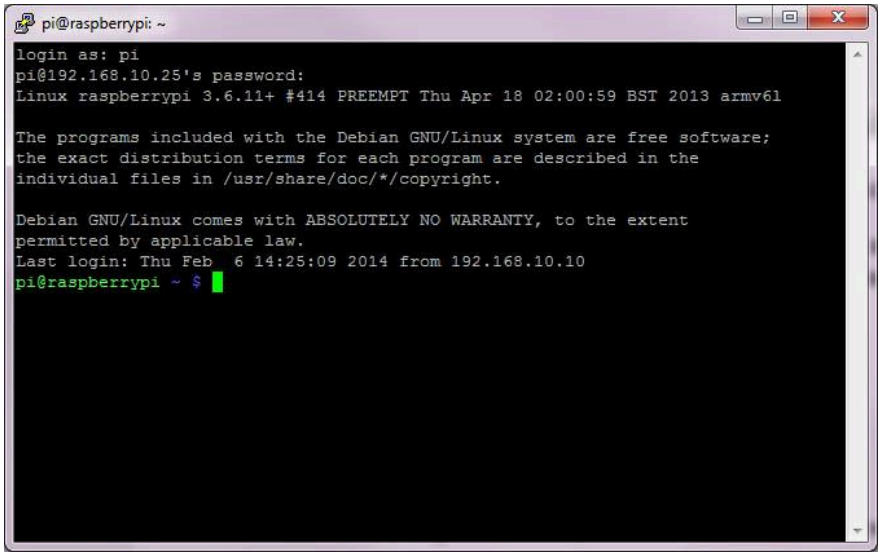
Wir müssen zwei Arten von Zaubersprüchen unterscheiden:

ssh

Einige dienen zur Kontrolle des Gerätes, welches fhem ausführt. Am einfachsten geht das mit ssh. Dann braucht man dann weder Monitor noch Tastatur dran. (Raspberry, FritzBox, BeagloBone…)

Im folgenden werde ich dies ssh Konsole nennen!

So könnte das aussehen:



Ich verwende Putty, das ist klein, schnell und kann ziemlich viel. Aber das ist nur meine Meinung.

Befehle die darin ausgeführt werden müssen, formatiere ich so:

```
echo Hallo mein Benutzer
```

fhem Befehlszeile

Der beste Weg die Zaubersprüche im Feemreich zu sprechen, ist dieser:

Schreibe die Sprüche in der Eingabeaufforderung / Befehlszeile von fhem

Auf diesem Bild sehen wir:

Die IP Adresse (192.168.10.25) unter der fhem bei mir zu finden ist, ist bei dir vermutlich eine andere Adresse.

Den Port den man angeben muss (:8083), ist der für das normale Webfrontend, für den Anfang solltest du den auch benutzen.

Und eben diese leere Box rechts neben dem Logo… herzlichen Glückwunsch, wir haben die Befehlszeile entdeckt :) Eingaben in dieser werden mit der <Return> Taste an fhem abgeschickt.

Wenn dir die getätigten Eingaben / Änderungen gefallen (und nur dann!), dann kannst du oben links auf “Save config” klicken.

Ich kennzeichne Eingaben in der fhem Befehlszeile mit dieser Formatierung:

get WEB pathlist

Dies für den Anfang. Im ersten Kapitel gibt es noch einige bebilderte Beispiele und die ein oder andere Farbmarkierung :)

Ganz wichtig: alles, was wir so fhem mitteilen, wird erst gespeichert wenn wir oben links auf "Save config" drücken.

Über die Beschwörung von Stimmen

Hier geht es um die Möglichkeit der Sprachausgabe von fhem. Für Novizen gut geeignet!

Zutaten:

Einen Rechner mit angeschlossenen Lautsprechern und Internet. Dieses Beispiel bezieht sich auf einen Raspberry Pi und per 3,5mm angeschlossenen Lautsprechern.

Du musst deinen Rechner vorbereiten mit folgenden Programmen:

```
sudo apt-get update
```

```
sudo apt-get install mplayer mp3wrap espeak
```

Vorbereitung:[1]

Nun müssen die Rechte angepasst werden. Per Default darf der fhem User nämlich nix mit Audio anstellen.

```
sudo visudo
```

Gewährt dir Zugriff auf die Datei, die einzelne Rechte erlaubt. Hier fügen wir folgendes hinzu, soweit es nicht schon da steht:

```
# Allow members of group sudo to execute any command
```

```
%sudo   ALL=(ALL:ALL) ALL
```

```
ALL     ALL = NOPASSWD: /usr/bin/mplayer
```

```
ALL     ALL = NOPASSWD: /usr/bin/espeak
```

```
ALL     ALL = NOPASSWD: /usr/bin/mp3wrap
```

Speichern…

Stelle den anlogen Audioausgang beim Raspberry ein.

---

[1] Es gibt noch eine alternative Anleitung: https://sites.google.com/site/semilleroadt/home/raspberry-pi
Ob die besser ist als obige, kann ich nicht sagen. Vermutlich, aber wohl auch aufwendiger :)

```
sudo amixer cset numid=3 1
```

Noch etwas, um Fehlermeldungen von mplayer im Log zu vermeiden:

```
sudo nano /etc/mplayer/mplayer.conf
```

Dort ganz unten eine Zeile anfügen mit:

```
nolirc=yes
```

Speichern…

zur Sicherheit, damit darf fhem auch auf die Audiohardware zugreifen.

```
sudo gpasswd -a fhem audio
```

Neustart:

```
sudo shutdown -r now
```

Ist bequemer als den Stecker zu ziehen und verhindert einen möglichen Grund für eine defekte SD Karte. Aber keine Angst, man kann die auf viele Art und Weise zerstören. Eine andere Möglichkeit bestünde im Übertakten vom RasPi.

Nun ins Feemreich...

Definition

```
define MyTTS Text2Speech hw=0.0
```

MyTTS ist der Name, unter welchem du die Stimmen rufen wirst.

hw=0.0 ist das Device, welches der mplayer nutzen wird.

Teste:

```
set MyTTS tts Mein Herr und Meister, ich stehe zu deinen Diensten!
```

Alternativ

Du kannst mehrere Stimmen beschwören. Zum Beispiel um diese an verschiedenen Orten zu hören, oder aber weil dir Gedanke an einen schizophrenen Diener gefällt? Der Name MyTTS

muss dann natürlich ein Anderer sein.

Wenn du nicht willst, dass die Stimmen den Weg durch das große Netz gehen und dabei vielleicht Spuren hinterlassen, dann kannst du sie auch vor Ort erzeugen. Sie klingen halt nicht so gut:

define MyTTSlocal Text2Speech hw=0.0

attr MyTTSlocal TTS_resource=ESpeak

Wenn du verschiedene Lautsprecher am RasPi betreibst (z.B. mit USB Soundkarten), dann würde sich logischerweise die hw=0.0 ändern. Damit kannst du Stimmen verschiedenen Ausgabegeräten zuordnen.

Damit kannst du eine weitere Stimme rufen. Diese klingt schrecklich, aber sie hinterlässt keine Spuren im Netz.

Probiere es aus:

set MyTTSlocal tts Mein Herr und Meister, mein Klang schlägt deine Feinde in die Flucht und bringt deine Freunde zum lachen!

Was noch?

Du könntest natürlich deine Feinde mit Donnergrollen effektiv erschrecken.

Wir bleiben hierfür aber bei unserer MyTTS (mit espeak geht das nämlich nicht).

Du brauchst ein ordentliches Donnergrollen. Nennen wir es mal donnergrollen.mp3

Und vielleicht einen Sturmwind: sturmwind.mp3

Die packen wir in den Ordner cache/templates

Nun:

Attr MyTTS TTS_FileMapping donner:donnergrollen.mp3 sturm:sturmwind.mp3

Jetzt geht es los:

set MyTTS tts Mein Herr und Meister :donner: ich werde Eure Feinde zerbrechen wie der Sturmwind die Bäume zerbirst :sturm:

Kombinationen mit notify

Was die Stimme noch kann?

Sie kann dir bei jedem Start ihre Ehrerbietung erweisen:

Wenn du also immer von der Ehrerbietung begrüßt werden willst, so schreibe die Worte:

define Begruessung notify global:INITIALIZED set MyTTS tts Mein Herr und Meister, ich stehe zu deinen Diensten!

Was hast du getan?

Ein notify mit dem Namen "Begruessung" angelegt, welches dann ausgeführt wird, wenn fhem fertig geladen ist (global:INITIALIZED).

Sie kann dich vor Feinden warnen!

Szenario:

Bewegungsmelder (Homematic, Name: CUL_HM_HM_Sen_MDIR_O_1A8306) (wir benennen den irgendwann auch um)

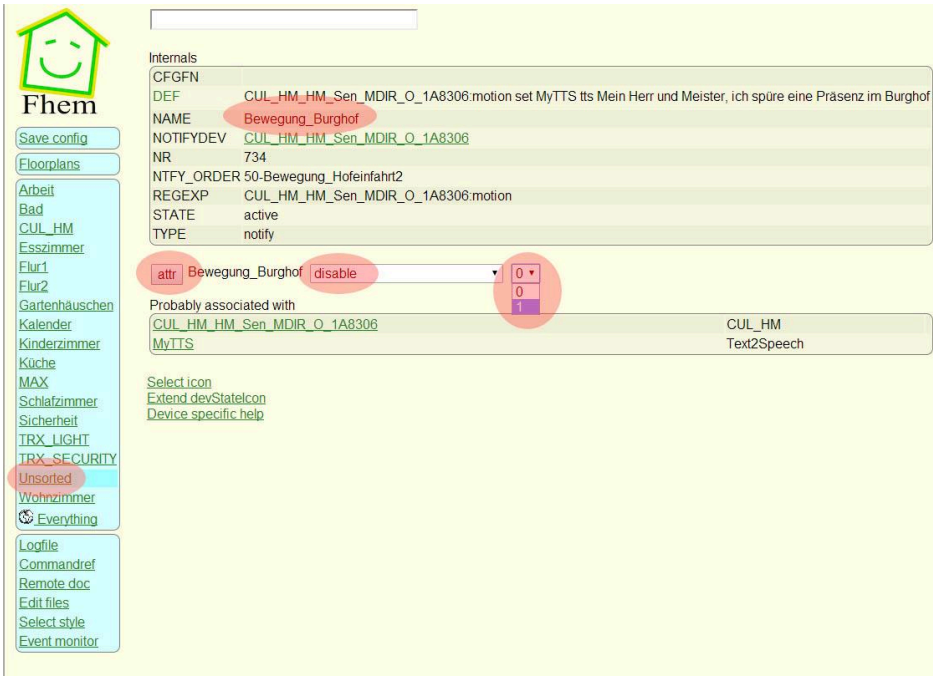
define Bewegung_Burghof notify CUL_HM_HM_Sen_MDIR_O_1A8306:motion set MyTTS tts Mein Herr und Meister, ich spüre eine Präsenz im Burghof

Wenn du hin und wieder eine andere Ansage bevorzugst, diese aber nicht löschen möchtest, so versuche es so:

Unter “Unsorted” (“Everything” geht auch) taucht rechts “notify” auf.

Darunter befindet sich Bewegung_Burghof => draufklicken

attr Bewegung_Burghof disable 1 => attr klicken



Damit unterbleibt diese Warnung.

Mit dem Spruch

define Bewegung_Hofeinfahrt notify CUL_HM_HM_Sen_MDIR_O_1A8306:motion set MyTTS tts Jemand ist in der Hofeinfahrt

warnt dich die gleiche Stimme, aber sie spricht eben etwas Anderes zu dir.

Wie kann man das nun prüfen? Natürlich könntest du jetzt in deinen Burghof sausen und das Event auslösen. Leider musst du dann ziemlich schnell dorthin flitzen, wo deine Stimme ertönt. Wer macht das schon gerne?

Mach es dir einfacher: Du kannst mit einem einfach Spruch dein Notify testen!

In meinem Fall müsste ich dazu sagen:

trigger CUL_HM_HM_Sen_MDIR_O_1A8306 motion

(beachte, da ist der : von oben nicht dabei!)

Damit würde das zugehörige Notify ausgelöst! Bei dir wird der Befehl anders aussehen, CUL_HM_HM_Sen_MDIR_O_1A8306 ist schließlich nur der Name meines Bewegungsmelders, nicht der deines. Durch das "trigger" sagen wir fhem, er soll so tun als ob der Bewegungsmelder ausgelöst hätte.

Kombinationen mit at

Nun lassen wir uns jeden Tag an ein wichtiges Ereignis erinnern:

Hierfür gibt es den Befehl “at”

define Festmahl at 19:58:00 set MyTTS tts Gleich beginnt das Festmahl

Dieser Befehl würde genau einmalig ausgeführt! (und dann gelöscht…)

Wollen wir das täglich haben, so füge ein * dazu!

define Festmahl at *19:58:00 set MyTTS tts Gleich beginnt das tägliche Festmahl

Wenn du merkst, dass du regelmäßig zu spät kommst und früher erinnert werden möchtest:

modify Festmahl *19:55:00

Willst du in 10 Minuten an etwas erinnert werden?

define Erinnerung at +00:10:00 set MyTTS tts Mein großer Zaubermeister, ich sollte dich an etwas erinnern.

Kombinationen mit Variablen

Die Stimme informiert dich z.B. jeden Tag um 6.45 Uhr wie warm es ist:

Dazu brauchen wir nun natürlich einen Temperatursensor.

Meiner nennt sich:

GH_gh_TF_TEMPERATUR (ist ein Homematic Außentemperatursensor)

Das zugehörige Reading heißt *temperature*

Hierfür gibt es jetzt deutlich mehr und sich unterscheidende Lösungsmöglichkeiten!

Vorüberlegung (was immer du tust, tue es weise und bedenke das Ende)

Du kennst mittlerweile “notify” und “at”. Welchen nehmen? Zunächst klingt “at” ja nicht dumm.

define TemperaturAnsage at *06:45:00 ….

Was bedeutet das aber? Letztlich, dass immer um 6.45 Uhr TemperaturAnsage losläuft. Wollen

wir das wirklich? Was würde denn passieren, wenn wir um 16.00 Uhr wissen wollen, wie kalt oder warm es denn ist?

Dann hätten wir ein Problem! Wieso? Weil wir zwar ein Notify mit Trigger auslösen können, aber leider kein “at”. (hüte dich davor, auch wenn du die Macht über die Systemzeit besitzt, einfach mit dieser zu spielen!)

Wir benötigen also, wenn es denn flexibel werden soll, ein kleine Hilfskonstruktion.

Die Lösung nennt sich “Dummy”. Das ist ein Gerät ohne Hardware. Quasi die fhem Version einer systemweiten Variablen. Dem können wir nun jeden Wert zuweisen, den wir wollen. Und dann triggern wir das Notify einfach mit dem Dummy :)

Variablen, ja, da war noch etwas: natürlich gibt es in fhem auch Variablen. Die gelten aber nur innerhalb eines Moduls. Bzw. innerhalb einer Routine. Sowas brauchen wir auch noch, damit wir schön einen Zugriff auf das Temperatur-Reading bekommen.

Warum nicht direkt das Reading benutzen? Nun, zum einen ist es etwas kompliziert, zum anderen ist der Wert, zumindest bei mir, z.B. 8.8 statt 8,8 => d.h. der Punkt sollte für die Sprachausgabe auch in ein Komma umgewandelt werden. Damit sind wir wieder bei der Variablen...

Der Reihe nach:

Vorbereitung:

1. Dummy definieren (true/false, on/off, rede/schweig; prinzipiell ist es egal, aber wenn du in 1 Jahr mal wieder nachsiehst, sollte es dir wieder einfallen)

2. Notify auf Dummy schreiben

Inhalt des Notify:

3. Variable mit dem Temperatur-Reading *temperature* füllen

4. Variable umformen, Punkt zu Komma

5. Sprechen

6. Dummy wieder zurück setzen (Eigentlich unnötig, das Notify unten wird nur 1x auslösen, auch wenn der Wert bei “rede” verbleibt; aber irgend etwas sinnvolles wird uns schon noch dafür einfallen. Ist ja quasi erst Kapitel 1 vom 1. Band...)

Auslösen des Ganzen:

Den Dummy triggern

(das geht jetzt z.B. mit “at *06:45:00”, aber eben beliebig oft und nicht nur mit “at”)

Soweit die Überlegung. Jetzt zur Umsetzung:

1. define TemperaturAnsageDummy dummy

set TemperaturAnsageDummy schweig

(damit haben wir einen Dummy erschaffen, und ihm den Wert “schweig” zugewiesen)

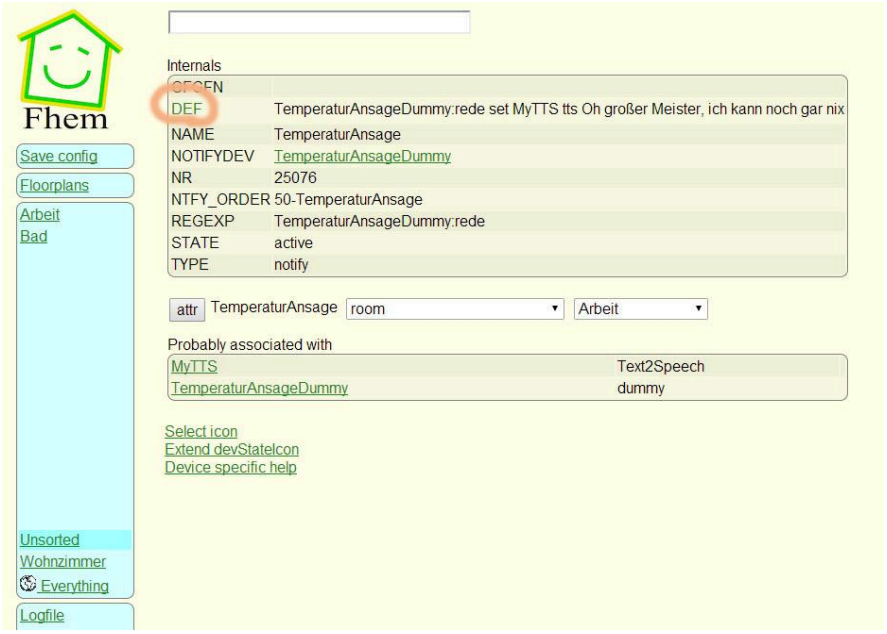
attr TemperaturAnsageDummy setList rede schweig

(in dem Fall eher kosmetischer Natur, wir bekommen damit eine DropDown Liste mit den Items rede und schweig. Brauchen wir hier nicht, aber wenn du in ferner Zukunft nicht mehr weißt, welche Werte du für deinen Dummy gewählt hattest, kannst in der Liste nachsehen.)

2. define TemperaturAnsage notify TemperaturAnsageDummy:rede set MyTTS tts Oh großer Meister, ich kann noch gar nix

Warum das?

Nun, wir müssen erst mal das Notify mit einem gültigen Befehl anlegen, sonst meckert fhem. Der Obige ist ein gültiger Befehl, deshalb können wir nun unter Unsorted, notify unser Notify (TemperaturAnsage) aufrufen, und durch klicken auf DEF (bei Internals) die Befehle bearbeiten.

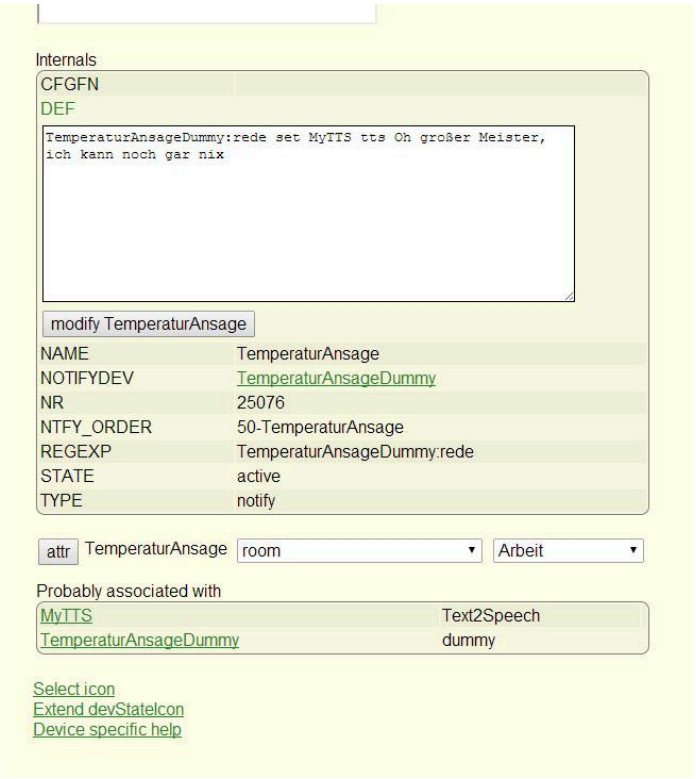


So sollte das dann aussehen…

Wir sehen also das Gerät (unseren Dummy),

welchen Zustand er haben muss (rede)

und was er tut (set MyTTS…)



Jetzt geht es weiter mit unserer Variablen…

Um mehr Befehle übersichtlich in ein Notify zu packen, ist eine { eine gute Lösung.

Wichtig: Immer wenn du { machst, bedarf es irgendwann einer }

Genau genommen schreiben wir in {} Perl Code. Ich glaube, ich habe das schon erwähnt. Werde ich noch öfter machen :)

Zur Erinnerung:

Mein Temperatursensor nennt sich: *GH_gh_TF_TEMPERATUR*

3. Wir schreiben nun in dem Fenster (also den Text drin verändern!):

```
TemperaturAnsageDummy:rede {

my $temperatur=(ReadingsVal("GH_gh_TF_TEMPERATUR","temperature",99));

fhem ("set MyTTS tts Oh grosser Meister, die Temperatur beträgt $temperatur Grad");

}
```

Was passiert hier?

Wir weisen der Variablen $temperatur einen Wert aus den Readings zu. Die 99 sind der Wert, der angenommen werden soll, wenn das aus irgend einem Grund in die Hose geht. Erzählt dir fhem also, dass es kuschelige 99 Grad draußen hat, brennt vermutlich nicht gerade dein Garten ab, eher hast du einen Fehler in dem Temperaturreading.

Weiterhin beenden wir jede Zeile mit ;

Noch etwas anderes passiert:

Wir schreiben das ganze etwas umständlicher als sonst. Warum, etwas kompliziert zu erklären. Sagen wir der Einfachheit im Moment: Das ist nötig, da wir gerade Perl sprechen, jedoch fhem Befehle, die Perl nicht kennt, .benutzen wollen.

4. Bis jetzt klingt die Ansage noch etwas, hm, merkwürdig. Formen wir sie um :)

```
TemperaturAnsageDummy:rede {

my $temperatur=(ReadingsVal("GH_gh_TF_TEMPERATUR","temperature",99));
```

```
$temperatur =~ s/\./ komma /;

fhem ("set MyTTS tts Oh grosser Meister, die Temperatur beträgt $temperatur Grad");

}
```

Was haben wir jetzt gemacht?

=~ besagt, dass wir den Inhalt von $temperatur verwenden wollen

s steht für substitute, wir wollen was ersetzen

/ / behinhaltet die zu ersetzende Zeichenfolge. Eigentlich ja . (also den Punkt). In diesem Fall funktioniert es aber nicht so einfach, der . ist ein spezielles Zeichen, welches maskiert werden muss. Dazu der \ vor dem .

komma / ist das, womit wir unser . Zeichen ersetzen

5. Für die Sprachausgabe basteln wir uns dies nun in einen Satz.

Steht ja oben schon drin :)

6. Dummy zurück setzen

```
TemperaturAnsageDummy:rede {

my $temperatur=(ReadingsVal("GH_gh_TF_TEMPERATUR","temperature",99));

$temperatur =~ s/\./ komma /;

fhem ("set MyTTS tts Oh grosser Meister, die Temperatur beträgt $temperatur Grad");

fhem ("set TemperaturAnsageDummy schweig");

}
```

Wie bei der Sprachausgabe auch, müssen wir nun Perl sagen (weil das sprechen wir gerade), das nun ein fhem Befehl kommt.

Auslösen:

```
trigger TemperaturAnsageDummy rede
```

oder aber::

set TemperaturAnsageDummy rede

In diesem Beispiel würde beides gehen.

Für später:

Der Unterschied zwischen Trigger und Set ist im wesentlichen folgender:

Trigger tut nur so, als ob. Mit Set weisen wir tatsächlich in fhem einen Wert zu.

Um also um 06.45 Uhr die Temperatur erzählt zu bekommen, schreiben wir:

define TemperaturAnsageEinmal at 06:45:00 trigger TemperaturAnsageDummy rede

Und täglich:

define TemperaturAnsageMorgen at *06:45:00 trigger TemperaturAnsageDummy rede

Schick, oder?

Anmerkung:

Prinzipiell kann man sich das ganze $Zeug auch sparen. Wenn man Wert auf kurzen Programmcode legt, sollte man das auch machen. Andererseits ist es auf die gezeigte Art und Weise zwar umständlicher, aber dafür sieht man auch als Perl Neuling schön, was passiert.

Jetzt können wir das ausbauen.

Warum sollte uns unsere Stimme nur doof die Temperatur vorlesen?

Wir könnten uns ja gleich noch eine einfache Kleidungsempfehlung mit angeben lassen.

Kombination mit if[2]

IN BEARBEITUNG, KLAPPT NOCH NICHT

Keiner friert gerne. Damit wir also gleich die richtige Robe auswählen, werten wir doch die Temperatur gleich noch mit aus.

if $temperatur < -5 $Empfehlung = "Du solltest dir echt eine Winterrobe anziehen heute.";
if $temperatur >=-5 && <10 $Empfehlung = "Vielleicht könnte man etwas wärmeres tragen.";
if $temperatur >=10 $Empfehlung = "Ich würde heute eine leichte Sommerrobe empfehlen.";

Achtung:
Das müssen wir jetzt aber dringen VOR unserer Komma-Zauberei einbauen, weil anschließend die Zahl ja nicht mehr existiert. Aus "8.8" haben wir ja "8 komma 8" gemacht!

Also:

```
TemperaturAnsageDummy:rede {
my $temperatur=(ReadingsVal("GH_gh_TF_TEMPERATUR","temperature",99));
my $Empfehlung eq " ";
if ($temperatur < "-5") { $Empfehlung eq "Du solltest dir echt eine Winterrobe anziehen heute."
};
if ($temperatur >= "-5") && $temperatur <"10") { $Empfehlung eq "Vielleicht könnte man etwas
wärmeres tragen." };
if ($temperatur  >= "10") { $Empfehlung eq "Ich würde heute eine leichte Sommerrobe
empfehlen." };
$temperatur =~ s/\./ komma /;
fhem ("set MyTTS tts Oh grosser Meister, die Temperatur beträgt $temperatur Grad.
$Empfehlung");
fhem ("set TemperaturAnsageDummy schweig");
}
```

[2] Vorsicht: es gibt seit neuestem zwei verschiedene if in fhem: ein mal if (kleingeschrieben), das ist das Perl eigene if. Dann ist, ganz neu, auch ein IF (großgeschrieben) möglich. Letzteres ist quasi ein fhem eigenes IF, welches nicht im Perl Code auftaucht, sondern in fhem Code. Darauf möchte ich jetzt nicht eingehen, könnte aber was für Band 2 der Zaubersprüche sein.

Lichtzauber

Sind wir uns einig. Niemand schreitet gerne in Dunkelheit und Finsternis durch die Welt. Überall lauern Gefahren. Man kann auf spitze Gegenstände treten[3], gegen Dinge laufen[4], ja sogar angefallen und gebissen werden[5]. Andererseits, immer überall Licht um sich zu haben ist auch keine gute Idee[6].
Beginnen wir mit einer einfachen Übung, wir schalten mit fhem ein Licht ein. Das ist nicht schwer.
Meine Lichter bezeichne ich im wesentlichen so[7]:
Stockwerk_Raum_Gerät_Bezeichnung
Also z.B. so: OG_ar_WS_LICHT => Obergeschoss Arbeitszimmer Wandschalter Licht
oder OG_wz_DI_LICHT => Obergeschoss Wohnzimmer Dimmer Licht

Fangen wir also einfach an:

set OG_ar_WS_LICHT on
set OG_ar_WS_LICHT off
set OG_ar_WS_LICHT toggle
War nicht schwer. Zuerst haben wir das Licht eingeschaltet, dann aus, und jetzt sollte es grade brennen. Das macht toggle. Ist es an, geht es aus, ist es aus, geht es an.

Für eine bestimmte Zeit einschalten

Nehmen wir an, es gibt ein Licht, welches aus irgend einem Grund immer vergessen wird, ausgeschaltet zu werden.
Mein Nachwuchs und das Badezimmerlicht sind so eine Konstellation.

Idee 1, Timer

set OG_bd_WS_LICHT on-for-timer 600

Was passiert da?

---

[3] Reißzwecken im Fuß sehen zwar witzig aus, vor allem die Bunten, sind aber ansonsten eher schmerzhaft
[4] Kinder haben ein Gespür dafür, sperrige Gegenstände in Gänge zu stellen (jedenfalls meine)
[5] Meine Katze kennt keinen Pardon, wenn man ihr auf den Schwanz tritt.
[6] Kühlschränke sind mir in dieser Hinsicht etwas suspekt. Weißt du sicher, dass das Licht wirklich aus ist, wenn du die Türe geschlossen hast?
[7] Nicht meine Idee, geborgt von http://www.fischer-net.de/hausautomation/fhem/22-fhem-devicenamen.html

Das Licht für 600 Sekunden, gemein hin also 10 Minuten, eingeschaltet. Dann geht es wieder aus.
Ok, nur wird vermutlich niemand erst den PC anwerfen, die fhem Seite aufrufen und den Befehl eintippen, nur damit er vergessen kann das Licht aus zu schalten.

Idee 2, Wachhunde gesucht (Watchdog, Zeitschaltuhr)

Wir brauchen einen Wachhund. Dieser soll nun nicht bellen, sondern das Licht ausschalten.

Vorüberlegung:

Wir kennen schon ein notify. Warum nicht das nehmen?
define LichtVonSelbstAus notify OG_bd_WS_LICHT:on set OG_bd_WS_LICHT on-for-timer 600
Du kannst es ja ausprobieren. Ich würde davon abraten. Wenn du es probieren willst, mach vorher einen Event Monitor auf, das sieht eindrucksvoller aus.
Was passiert?
Nun, der Status vom Licht ist on. Also versucht nun unser Notify das Licht für 600 Sekunden einzuschalten. Soweit, so gut. Welchen Status hat unser Licht? Für ganz kurze Zeit hat es den Status on-for-timer. Aber eben nur für kurze Zeit. Dann bekommt es wieder den Status on. Ergo läuft unser Notify wieder von vorne los, on-for-timer 600. Ist quasi eine Endlosschleife.
Die wird irgendwann abbrechen, weil die Funklast zu groß wird.

Das ist also nix.

Genau dafür gibt es nun einen Wachhund. Unser fhem Wachhund tut im wesentlichen das, was ein Echter auch täte. Dasitzen, warten, und dann losschlagen.

Jetzt nicht erschrecken

define LichtVonSelbstAus watchdog OG_bd_WS_LICHT:on 00:10:00 OG_bd_WS_LICHT:off set OG_bd_WS_LICHT off

Ganz langsam:

define LichtVonSelbstAus watchdog	=> wie bei at oder notify auch, nur eben ein Watchdog
OG_bd_WS_LICHT:on	=> das auslösende Ereignis, das Licht ist an
00:10:00	=> das Ereignis soll 10 Minuten dauern, dann wird der Watchdog aktiv
OG_bd_WS_LICHT:off	=> bei dem Ereignis soll der Watchdog nicht aktiv sein!
set OG_bd_WS_LICHT off	=> dann soll das Licht ausgeschaltet werden

Das funktioniert soweit schon ganz gut. Probier es aus. Nun rufe deine Frau und führe ihr dein neues Kunststück vor.

Hihi. Verschaukelt. Auch wenn es gerade wunderbar funktioniert hat, wird das Licht nun an bleiben.
Wieso denn das? Nun, vereinfacht, ausgedrückt, wir haben einen Hund von der Kette losgelassen. Dann hat er zugeschnappt, jetzt ist er weg.
Wir müssen also noch dafür sorgen, dass unser Wachhund wieder an der Kette hängt, damit wir ihn wieder los lassen können.
Glücklicherweise ist das ganz einfach.

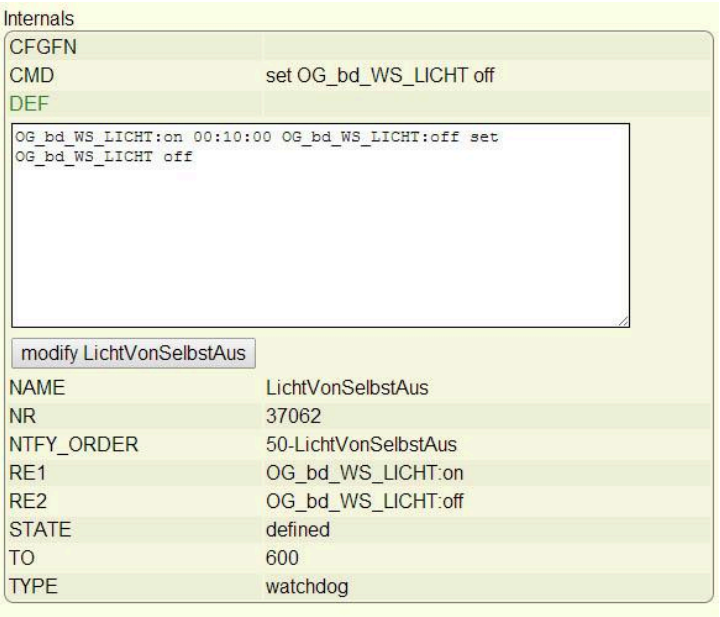
Den Wachhund wieder auf die Lauer legen!

Wir machen es genau wie vorhin auch, nur angereichert um ;; und einen weiteren Befehl
define LichtVonSelbstAus watchdog OG_bd_WS_LICHT:on 00:10:00 OG_bd_WS_LICHT:off set OG_bd_WS_LICHT off;; trigger LichtVonSelbstAus .

;;	=> ist eigentlich ein Zeilenumbruch, es kommt also noch ein Befehl dazu
trigger LichtVonSelbstAus .	=> Unser Wachhund soll sich anschließend wieder auf die Lauer legen. Bitte unbedingt auch den Punkt nach dem Leerzeichen berücksichtigen! Sonst klappt das nicht.

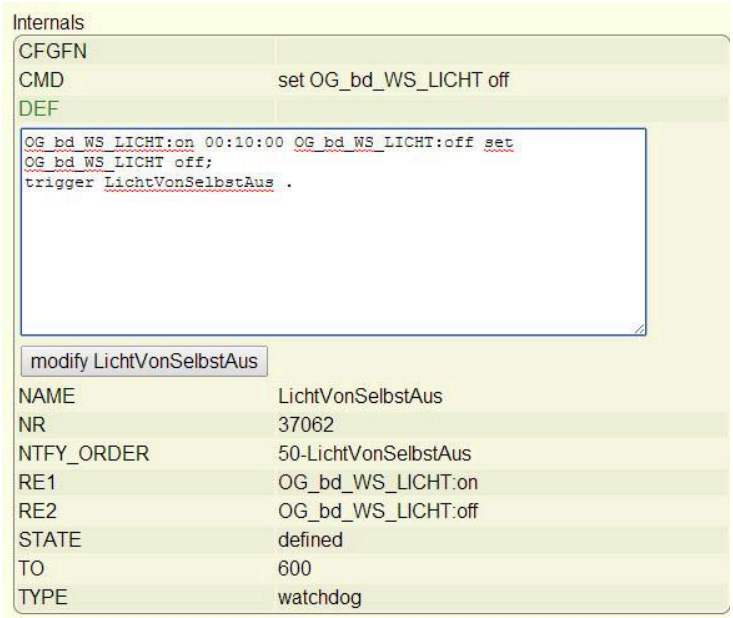
Das gibt jetzt vermutlich eine Fehlermeldung, die in etwa lautet: LichtVonSelbstAus allready defined. Delete it first.
???
Na ja, wir hatten den Namen ja schon vergeben.
Also entweder löschen wir unseren Befehl, oder wir modifizieren unseren Obigen.
Löschen geht ganz einfach: delete LichtVonSelbstAus
Dann können wir die erweiterte Zeile von oben nutzen.

Modifizieren ist auch einfach. Suchen wir unseren Wachhund unter Everything => watchdog.
Jetzt ohne Scheu auf DEF klicken



Das modifizieren wir jetzt:

;
trigger LichtVonSelbstAus .



Mausklick auf modify LichtVonSelbstAus
Fertig :)

Verwirrt?
Oben in dem gelben "Einzeiler" müssen wir ;; schreiben. Wenn wir über DEF den Watchdog bearbeiten, reicht ; aus.

Warum dürfen wir nun hier eigentlich normale fhem Befehle schreiben, und nicht das etwas kompliziertere fhem ("...")? Nun, das liegt daran, dass wir keine {} angegeben haben. Wir wollten ja keinen Perl Code schreiben. Obwohl wir das natürlich auch könnten. Mir fällt nur im Moment kein brauchbarer Anwendungsfall ein.

Noch etwas verfeinert:

Das funktioniert soweit ganz gut. Was aber tun, wenn wir wissen, dass wir das Licht etwas länger benötigen? Es soll im Bad ja manchmal passieren, dass 10 Minuten nicht ausreichen. Beim Reinigen des Magierkörpers z.B. :)

Dann könnten wir uns einem weiteren magischen Spruch zu Nutze machen!

Wir legen unseren Wachhund auf einen Wink hin schlafen:

sequence

Die Idee dahinter ist einfach. Wir bringen fhem bei, auf ein geheimes Zeichen zu achten.
define Bad_Licht_Bleibt_An_Seq sequence OG_bd_WS_LICHT:on 5.0 OG_bd_WS_LICHT:on
Bis jetzt passiert noch nicht viel.
Die Sequence wird prinzipiell dann ausgelöst, wenn man den Schalter im Abstand von 5,0 Sekunden 2 mal auf "on" schaltet. Quasi ein Doppelklick.[8]

Jetzt muss auf unser geheimes Zeichen hin etwas passieren!
Wir brauchen noch ein Notify, welches für uns Befehle ausführt.
Testweise geben wir mal folgendes ein:
define Schlaf_Bad_Wachhund notify Bad_Licht_Bleibt_An_Seq:trigger set MyTTS tts Mein Meister, fast schon hast du deinen Wachhund zum einschlafen gebracht

Nur, welche Befehle eigentlich?
Nun, es wäre schlau, wenn er unser LichtVonSelbstAus Wachhund schlafen geht. Wir bearbeiten dazu wieder mit DEF unser Notify (Schlaf_Bad_Wachhund).

attr LichtVonSelbstAus disable 1
(Die set MyTTS können wir löschen und mit obigem einfach ersetzen, oder wir hängen das attr mit ; an die Zeile dran, dann haben wir auch gleich eine akkustische Bestätigung, wenn unser Wachhund zugeschlagen hat)

Dieses ist es. Genau wir bei den Notifys aus Kapitel 1. Da, wo wir einen Wächter für den Burghof beschworen hatten…

---

[8] Du könntest natürlich auch mehrere Knöpfe mit verschiedenen Zeiten aneinanderreihen. Beispielsweise einen am Dachboden, einen im Keller, die müssen 3x hintereinander gedrückt werden, damit die Alarmsirene wieder aufhört. Vielleicht gut zum Abnehmen geeignet?

Wir setzen des Watchdog auf disabled...

Was noch fehlt? Nun, wir haben nun mit einem Doppelten Tastendruck den Wachhund schlafen gelegt. Wir müssen ihn wieder aufwecken. Zum Beispiel wenn man das Licht nun von Hand ausschaltet.

define WachAuf_Bad_Wachhund notify OG_bd_WS_LICHT:off attr LichtVonSelbstAus disable 0

Das passiert nun immer, wenn man den Lichtschalter ausschaltet. Sollte LichtVonSelbstAus schon auf disable 0 stehen, passiert aber auch nichts schlimmes. Das passt soweit.

Mögliche Probleme[9]

Es kann sein, dass der Trigger für sequence schon scheinbar auf den ersten Tastendruck auslöst. Hat mir selber auch die ein oder andere Stunde der Fehlersuche beschert.
Wenn das passiert, rate ich dazu, mal den Eventmonitor im Auge zu behalten.
Meine HM-LC-Sw1PBU-FM zeigen da etwas merkwürdige Verhalten, dass die on-Befehle 3x hintereinander gesendet werden, obwohl das Pairing völlig korrekt ist.
Dazu sind 2 Lösungen denkbar:
1. Wenn er unbedingt pro Druck 3x senden will, definiert man eben die sequence mit 6 on-Befehlen
2. Einfacher, man setzt bei dem betreffenden Schalter ein Attribut wie:
attr OG_bd_WS_LICHT event-min-Intervall state:2
Was bewirkt das?
Nun, wenn der Lichtschalter innerhalb von 2 Sekunden mehrmals geschaltet wird (also auch unsere Funkwiederholung), so werden diese Schaltbefehle ignoriert. Dann kommen sie auch nicht bei der Sequence an.

Beides hilft gegen die Symptome (die Sequence arbeitet damit korrekt), beseitigt aber die Ursache für den Fehler nicht!

[9] Was auch wichtig ist: So ein Funkschalter ist kein PC-Eingabegerät! Auch wenn ein Doppelklick 200msec schnell ist, wird man das mit einem Funklichtschalter nicht hinbekommen. Ich würde eher 3-5 Sekunden einplanen, und auch das braucht etwas Übung… Eine Stimme ertönen zu lassen, wenn der Watchdog deaktiviert worden ist, erscheint mir bei weitem nicht die dümmste Idee zu sein, die man haben kann.

Idee 2 Zusammengefasst

Hier nochmal kurz die einzelnen Schritte als Übersicht:

Watchdog definieren, dafür sorgen, dass er erneut gestartet wird:
define LichtVonSelbstAus watchdog (wenn Licht an, nach best. Zeit wieder aus)

Sequence definieren, um eine Tastenfolge zu definieren:
define Bad_Licht_Bleibt_An_Seq sequence (best. Tastenfolge triggert ein Notify)

Die Sequence triggert ein Notify, welches den Watchdog deaktiviert
define Schlaf_Bad_Wachhund notify (besagtes Notify, deaktiviert Watchdog)

Ein normaler Aus-Tastendruck soll den Watchdog wieder starten
define WachAuf_Bad_Wachhund notify (Watchdog wird prinzipiell aktiviert)

Das schwierige daran ist eigentlich nur, im Kopf zu behalten wie die einzelnen Notify etc. ineinander greifen.

Nochmal zusammengefasst, aber diesmal farbig markiert:

define LichtVonSelbstAus watchdog OG_bd_WS_LICHT:on 00:10:00 OG_bd_WS_LICHT:off set OG_bd_WS_LICHT off;; trigger LichtVonSelbstAus .

define Bad_Licht_Bleibt_An_Seq sequence OG_bd_WS_LICHT:on 5.0 OG_bd_WS_LICHT:on

define Schlaf_Bad_Wachhund notify Bad_Licht_Bleibt_An_Seq:trigger set MyTTS tts Mein Meister, der Wachhund schläft jetzt;;attr LichtVonSelbstAus disable 1

define WachAuf_Bad_Wachhund notify OG_bd_WS_LICHT:off attr LichtVonSelbstAus disable 0

Ich empfehle dringend, die Namen der einzelnen Notify, Sequence etc. Definitionen auf einem Stück Papier zu notieren, und selbiges in einem Ordner gut zu verwahren.
Zumindest mein Kopf pflegt mich diesbezüglich gerne im Stich zu lassen :)

Licht soll mich immer umgeben

Jetzt haben wir gesehen, wie man Licht Einschalten kann und dafür sorgt, dass es wieder aus geht.
Aber auch auf unseren Reisen, wenn wir an vielen Orten nur kurz verweilen, wäre Licht schon eine feine Sache.
Sorgen wir also dafür, dass das Licht immer da ist, wo wir gerade sind.
Wir benötigen dazu einen Bewegungsmelder.

Licht und Bewegungsmelder

Mein Bewegungsmelder nennt sich: OG_fl2_BM
Mein Lichtschalter nennt sich OG_fl2_DIM_LICHT (ist ein Dimmer)
Es sind HomeMatic Geräte. Einige Befehle und Eigenschaften dieser Geräte werden bei anderen Systemen eventuell anders sein!

Vorbemerkungen zu HomeMatic Bewegungsmeldern:

Wir haben im ersten Kapitel ja schon mit Bewegungsmeldern experimentiert. Vielleicht ist dir aufgefallen, dass die Meldung immer erst nach 240 Sekunden ausgelöst wird.
Das ist die Voreinstellung der Bewegungsmelder. Man kann dies verändern[10]. Gilt für Innensensoren, wie für Außensensoren. Meine Bewegungsmelder im Haus habe ich alle auf 30 Sekunden eingestellt. Wenn also eine andauernde Bewegung ist, wird das alle 30 Sekunden gemeldet.
Noch etwas wichtiges: der state von diesen Geräten ist immer motion! Eine Prüfung auf motion ja oder nein, macht daher so keinen Sinn!
Weiterhin messen diese Bewegungsmelder auch die Helligkeit mit. Das ist eine Zahl von 0 (ziemlich finster) bis 200 (ziemlich hell).

Der Zauberspruch, einfachste Form

define Flur2_LichtAn notify OG_fl2_BM:motion set OG_fl2_DIM_LICHT on
Was tut er?
Es ist ein Notify. Die kennen wir schon. Ich habe es Flur2_LichtAn genannt. Eine reine Willkürentscheidung…
Das notify wird ausgelöst wenn OG_fl2_BM:motion gesendet wird.
Und es macht ein einfaches set … on

Wie wird das funktionieren?

---

[10] Hier der Link zur Anleitung

http://www.fhemwiki.de/wiki/HM-Sec-MDIR_Funk-Bewegungsmelder_innen#Editieren_von_Registerwerten

Nun, im wesentlichen wird nun für alle Ewigkeit[11] ein Licht brennen, sobald ein mal jemand oder etwas vom Bewegungsmelder erfasst worden ist.
Insbesondere wird dies Tag und Nacht sein.
Blöd?
Nun, nicht besonders schlau.

Der Zauberspruch, etwas verbessert, zeitliche Beschränkung

```
define Flur2_LichtAn notify OG_fl2_BM:motion set OG_fl2_DIM_LICHT on-for-timer 45
```

Was ist neu?
Das on wurde ersetzt durch on-for-timer 45

Wie wird das funktionieren?
Bei Bewegung wird das Licht für 45 Sekunden an gehen, anschließend geht es von alleine wieder aus.
Sind wir damit fertig?
Leider noch nicht, es spielt immer noch keine Rolle, ob es Tag oder Nacht ist.

Der Zauberspruch weiter verbessert, Helligkeit und if

Dafür gibt es mehrere Möglichkeiten. Man kann bei einigen Bewegungsmeldern Helligkeitsschwellen im Gerät festlegen. Halte ich für unpraktisch. Geht auch direkt mit fhem. Hat den Vorteil, dass der weg so prinzipiell mit jedem Gerät klappt, wenn man irgendwo eine Helligkeit her bekommt.

Wir sollten jetzt direkt wieder die Internals unseres Notify mit DEF bearbeiten. Zur Übung.

```
OG_fl2_BM:motion {
my $Helligkeit=(ReadingsVal("OG_fl2_BM","brightness",99));
fhem ("set OG_fl2_DIM_LICHT on-for-timer 45") if ($Helligkeit <= 90);
}
```

Hoppla.
Jetzt ist es wieder etwas mehr geworden:
Zum einen verwenden wir nun wieder {}. Als erfahrener Leser weißt du mittlerweile, dass wir nun wieder was mit Perl zaubern.
Wir definieren eine Variable, my $Helligkeit, und füllen diese mit dem, was das Gerät OG_fl2_BM unter dem Reading brightness versteht.
Sollte das nicht klappen, nehmen wir an, der Wert wäre 99.

Warum das geht? Weil die HM Bewegungsmelder netterweise auch gleich die Helligkeit messen. Wir sind hier völlig frei, irgend ein Gerät zu nehmen. Wir müssen wissen wie es sich nennt, und wir müssen das entsprechende Reading kennen, hinter dem sich die Helligkeit versteckt.

---

[11] Man könnte natürlich den Lichtschalter drücken, dann ist es aus...

Beispiel, anderes Gerät zur Helligkeit:

```
OG_fl2_BM:motion {
my $Helligkeit=(ReadingsVal("CUL_HM_HM_Sen_MDIR_O_1A8306","brightness",99));
fhem ("set OG_fl2_DIM_LICHT on-for-timer 45") if ($Helligkeit <= 110);
}
```

Hier würde ich nun den Helligkeitswert des Bewegungsmelders im Burghof verwenden. Das mache ich übrigens wirklich so!

Weiter im Text:
if
Mit der Zeile `fhem ("set OG_fl2_DIM_LICHT on-for-timer 45") if ($Helligkeit <= 110);` sorgen wir nun dafür, dass das Licht nur geschaltet, wird, wenn unsere $Helligkeit kleiner oder gleich 110 ist.
Ach ja, eben für 45 Sekunden.

Wieso 110?
Äh, tja, ich habe mich abends in den Flur gesetzt und geschaut, ab welcher Helligkeit ich gerne ein Licht hätte. Dann habe ich das aktuelle Reading abgelesen und diese Zahl genommen.

Der Zauberspruch nochmal verbessert, wenn man einen Dimmer hat

Wenn du einen Dimmer hast, kannst du es noch etwas verbessern. Es ist ja schön, wenn Nachts das Licht angeht. Aber wenn der kleine Magier eigentlich nur kurz für den Mitternachtssnack in die Küche will, oder aber mal für kleine Magier muss, dann ist dieses blendende Licht doch irgendwie störend.
Mit einem Dimmer würde der Befehl so aussehen:

```
OG_fl2_BM:motion {
my $Helligkeit=(ReadingsVal("CUL_HM_HM_Sen_MDIR_O_1A8306","brightness",99));
fhem ("set OG_fl2_DIM_LICHT pct 30 45") if ($Helligkeit <= 110);
}
```

Statt on-for-timer 45 ersetze es durch pct 30 45.
pct fährt direkt die Helligkeit des Dimmers an. Also hier 30%.
Die 45 dahinter sind die Sekunden, die es brennen soll.