Wie faul kann man sein? Die Ausgangssituation ist ein älterer Drucker, der sich softwareseitig immer wieder mal aufhängt, sprich: er ist dann zwar an, doch er ist per Netzwerk nicht erreichbar und macht auch sonst nichts mehr. Das einzige, was in dieser Situation hilft: erst ausschalten, dann anschalten.

Oder richtiger: erst hinlaufen, dann ausschalten, dann einschalten — Nervkram also. Die Idee lag daher in der Beschaffung einer IP-Steckdose, genauer einer EnerGenie EG-PMS2-LAN: einer 6fachen Überspannungsschutz-Steckdosenleiste mit LAN-Schnittstelle, die es ermöglicht, vier der Steckplätze aus der Ferne zu kontrollieren. Das ist erfreulich kostengünstig, konfiguriert ist sie auch schnell, und übers Webinterface lässt sie sich rasch bedienen — aber wer möchte das schon? Das muss doch auch eleganter gehen

Daher installierte ich auf meinem Monitoring-Host das Tool `egcfg`, das ich auf GitHub aufstöberte: es dient genau dazu, die vier Steckplätze anzusteuern.

```
$ git clone https://github.com/unterwulf/egctl.git
$ cd egctl
$ make && make install
```

Zum Funktionieren benötigt das Tool — es heißt übrigens `/usr/local/bin/egctl` — noch ein Konfigfile `egtab`, welches in `/etc` abgelegt werden will: hier sind IP-Adresse und Passwort der Steckdosenleiste zu hinterlegen.

```
# Name    Protocol IP              Port    Password
# ------- -------- --------------- ------- --------
egpms     pms20    a.b.c.d         5000    einpasswort
```

Über die Konsole kann das Gerät nun bereits angesteuert werden: soll also beispielsweise der vierte Steckplatz eingeschaltet werden und die übrigen drei unverändert bleiben, so wäre hierzu folgender Aufruf nötig:

```
$ /usr/local/bin/egctl egpms left left left on
```

Was also soll konkret passieren, wenn der Drucker mal wieder rumspinnt? Steckdose #4 abschalten… paar Sekunden warten… und Steckdose #4 dann wieder einschalten. Das passt alles in ein kleines Shell-Script namens `Reset_Printer.sh` — ich hab’s noch durch ein paar Debug-Meldungen aufgeblasen und dann in `/etc/icinga2/scripts` abgelegt.

```
#!/bin/bash

PROG="$( basename $0 )"
DEVICE="egpms"
SOCKET="4"
STATE="$1"

if [ $STATE -gt 0 ] ; then
  logger "$PROG: Resetting socket $SOCKET on poweroutlet $DEVICE:"
  /usr/local/bin/egctl $DEVICE left left left off
```

```
  if [ "$?" == "0" ] ; then
    logger "$PROG: Turned $SOCKET on $DEVICE OFF; waiting 5 seconds and
turning ON again."
    sleep 5
    /usr/local/bin/egctl $DEVICE left left left on
    if [ "$?" == "0" ] ; then
      logger "$PROG: Turned $SOCKET on $DEVICE ON again; all done."
    else
      logger "$PROG: Not successful. Please investigate."
    fi
  else
    logger "$PROG: Not successful. Please investigate."
  fi
else
  logger "$PROG: Host state is $STATE, nothing to do."
fi
```

Ist ein EventCommand hinterlegt, so wird es bei Statusänderung des Hosts ausgeführt — und zwar bei **jeder** Statusänderung. Deshalb fange ich seitens des Scripts ab, welchen Status mein Host hat, dieser wird nämlich als Argument mit übergeben; und das Script soll nur dann ausgeführt werden, wenn der Drucker ungleich `UP` (was einer `$host.state_id$` > 0 entspricht) gelistet wird. Ich habe auch kurzfristig mit `command = "test $host.state_id$ -eq 0 || /etc/icinga2/scripts/Reset_Printer.sh"` und ähnlichen Scherzen experimentiert, erhielt aber in den Logfiles die lapidare Meldung `no such file or directory`, und im Director lässt sich sowas überhaupt nicht anlegen, weil er ein Array draus macht — aber egal, kann sein, dass es irgendwie geht, ich hab mich nicht wirklich lange damit beschäftigt.

Brrr, okay, mein Script muss ich noch als EventCommand in Icinga 2 erfassen — ich hab das über den Director gemacht, aber geht natürlich auch so.

```
object EventCommand "event_reset_printer" {
    import "plugin-event-command"

    command = [ "/etc/icinga2/scripts/Reset_Printer.sh" ]
    arguments += {
        "-state" = {
            required = true
            skip_key = true
            value = "$host.state_id$"
        }
    }
}
```

Zuletzt muss man dem Host nun natürlich diesen Event-Handler verpassen: das wiederum geht über den Director nicht (dort kann man Services einen Handler zuweisen, aber eben nicht Hosts), weshalb ich den Drucker in Form einer Konfigdatei nach `/etc/icinga2/conf.d` geschmissen hab.

```
object Host "printer" {
  import "Old Printer"
```

```
  display_name = "printer"
  address = "w.x.y.z"
  enable_event_handler = true
  event_command = "event_reset_printer"
}
```

Grundsätzlich war's das auch schon: im Debug-Log lässt sich prima beobachten, ob und wann der Event Handler auf die Reise geschickt wurde, und die Debug-Meldungen des Scripts sorgen für Ausgabe in `/var/log/syslog`.

```
$ icinga2 feature enable debuglog
$ service icinga2 restart
$ multitail /var/log/icinga2/debug.log /var/log/syslog
[2018-03-23 09:34:11 +0100] notice/Checkable: State Change: Checkable
'printer' soft state change from UP to DOWN detected.
[2018-03-23 09:34:11 +0100] notice/Checkable: Executing event handler
'event_reset_printer' for service 'printer'
[2018-03-23 09:34:11 +0100] debug/DbEvents: add eventhandler history for
'printer'
---
Mar 23 09:34:11 monitor nagios: Reset_Printer.sh: Resetting socket 4 on
poweroutlet egpms:
Mar 23 09:34:12 monitor nagios: Reset_Printer.sh: Turned 4 on egpms OFF;
waiting 5 seconds and turning ON again.
Mar 23 09:34:17 monitor nagios: Reset_Printer.sh: Turned 4 on egpms ON again;
all done.
```

Mir sind die Syslog-Meldungen tatsächlich wichtig, da in Icinga Web 2 nicht ersichtlich ist, ob der Event Handler zuschlug und was er gegebenenfalls gemacht hat — an dieser Stelle ist man vollständig auf Logfiles angewiesen.
Kurzum: es funktioniert. Der als `DOWN` erkannte Drucker wird hart resettet und funktioniert anschließend wieder; da der nächste *state change* von `DOWN` nach `UP` geht, wird das Script dann nicht erneut ausgeführt — wer will sich schon in eine Endlosschleife katapultieren?

Jedenfalls ist das eine durchaus nicht uncoole Sache, und mir fallen da spontan noch ein paar Anwendungsbeispiele zu ein…