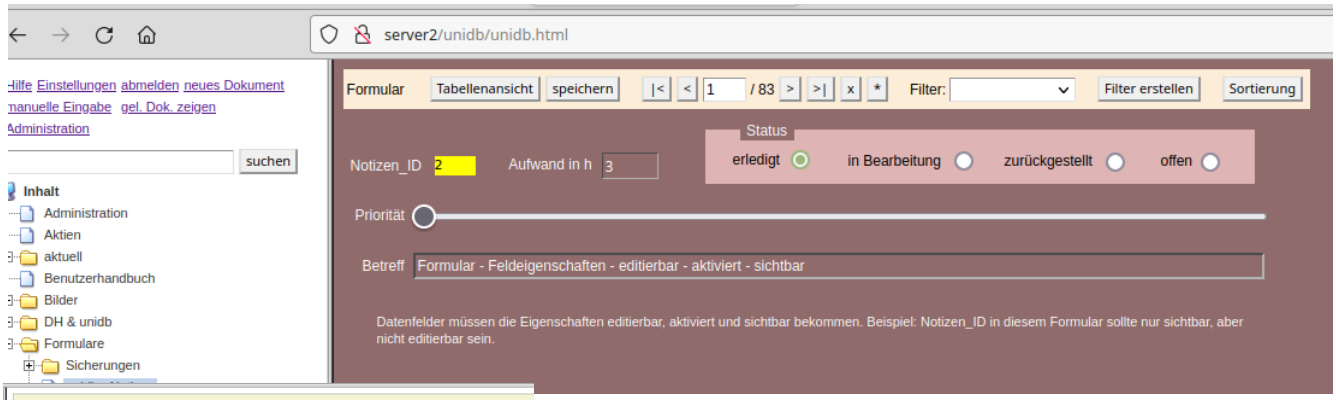
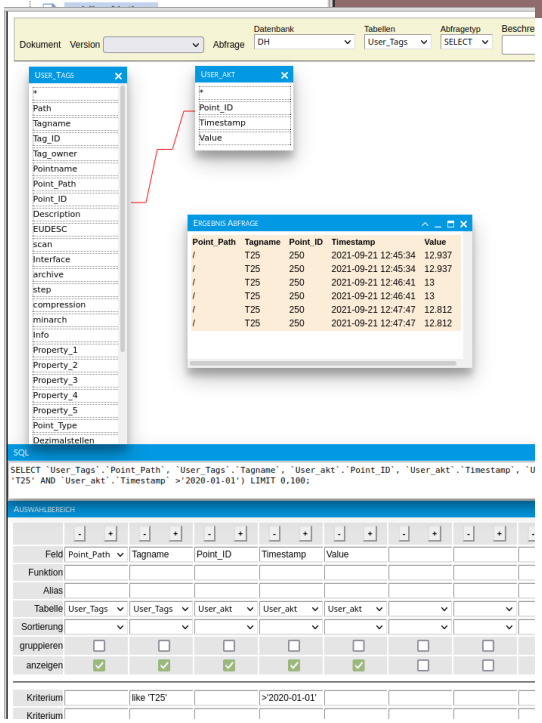
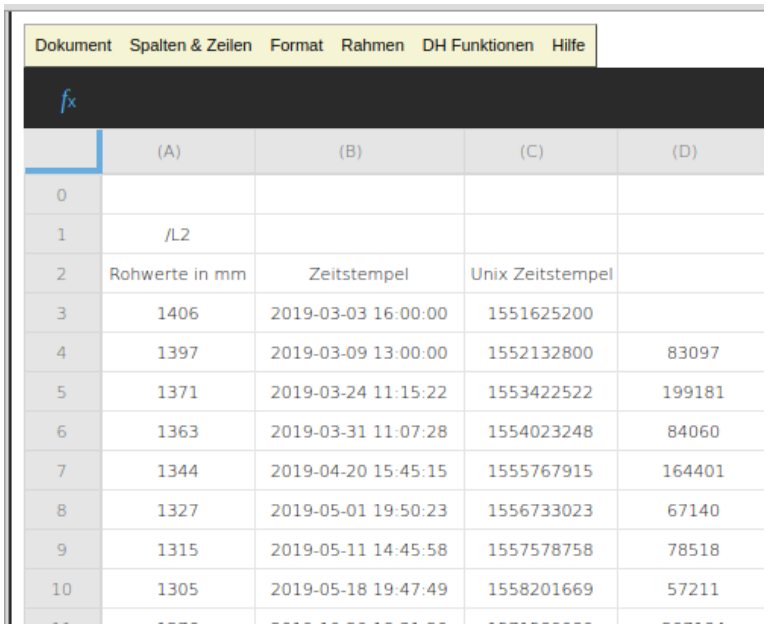
DH Schnittstellen Programmierung

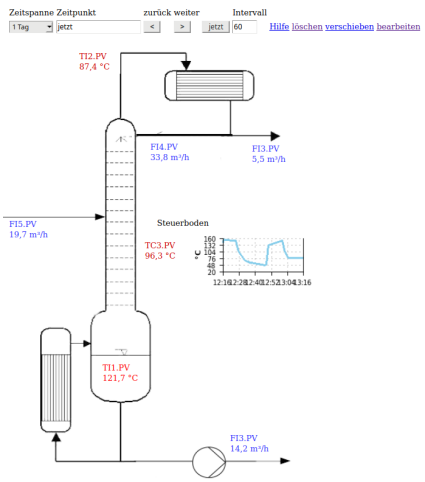
Version 1.0





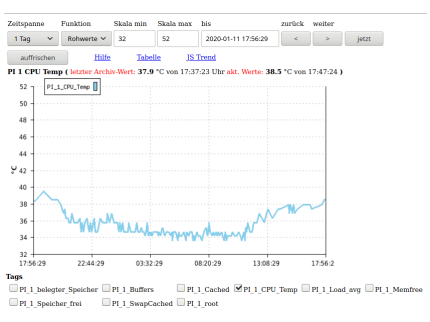


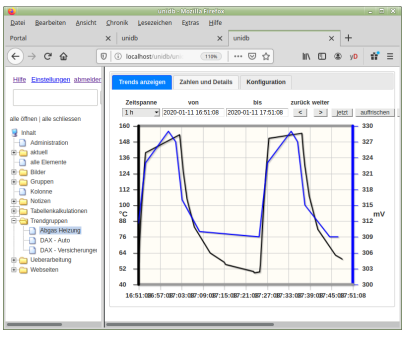
	(A)	(B)	(C)	(D)
0				
1	/L2			
2	Rohwerte in mm	Zeitstempel	Unix Zeitstempel	
3	1406	2019-03-03 16:00:00	1551625200	
4	1397	2019-03-09 13:00:00	1552132800	83097
5	1371	2019-03-24 11:15:22	1553422522	199181
6	1363	2019-03-31 11:07:28	1554023248	84060
7	1344	2019-04-20 15:45:15	1555767915	164401
8	1327	2019-05-01 19:50:23	1556733023	67140
9	1315	2019-05-11 14:45:58	1557578758	78518
10	1305	2019-05-18 19:47:49	1558201669	57211



Hilfe Gruppe bearbeiten löschen verschieben

PI_1_belegter_Speicher	benötigter RAM	2020-01-11 17:47:24	344.3	MByte
PI_1_Buffers	Puffer	2020-01-11 17:47:24	161692	KByte
PI_1_Cached	Cached	2020-01-11 17:47:24	394728	KByte
PI_1_CPU_Temp	PI 1 CPU Temp	2020-01-11 17:47:24	38.5	°C
PI_1_Load_avg	PI 1 Auslastung	2020-01-11 17:47:24	9	%
PI_1_Memfree	freier Speicher	2020-01-11 17:47:24	39324	KByte
PI_1_Speicher_frei	freier Speicher	2020-01-11 17:47:24	62.8	%
PI_1_SwapCached	Auslagerungsspeicher	2020-01-11 17:47:24	72	KByte
PI_1_root	Server Root Partition	2020-01-11 17:47:24	16.855	GByte





Inhaltsverzeichnis

Begriffe und Abkürzungen

DataHistorian	Datenbankanwendung, welche fortlaufend Daten im Format Datenpunkt – Zeitstempel – Wert sammelt und diese in aufbereiteter Form wieder zur Verfügung stellt.
DH	Abkürzung für DataHistorian
DB	Abkürzung für Datenbank.
Tag	Datenpunkt in einem DataHistorian
Point	Andere Bezeichnung für einen Tag / Datenpunkt.
Point_ID	Eindeutige Nummer eines Tags, vergleichbar einer Seriennummer.

Verwendete Formatierungen

fette Schrift	Name eines Steuerelementes (Schalter, Textfeld, ...)
kursiv	Namen von Eigenschaften o.ä.
farbig hinterlegt	Einem farbig hinterlegten Text sollte man etwas mehr Aufmerksamkeit widmen.
`courier`	Programmcode

1. Einführung

Wie Sie Ihre Daten in den DH bekommen, bleibt grundsätzlich Ihnen überlassen. Es ist aber gute Praxis, für eine neue Schnittstelle das Script **DH_basic_interface.py** zu verwenden. Es enthält bereits alle Komponenten, die zur Kommunikation mit dem DH erforderlich sind. Sollten Sie jedoch Ihren eigenen Weg gehen, dann schreiben Sie die Daten bitte auf allen Servern im Kollektiv nur in die Tabelle **akt**. Jeder Eintrag in diese Tabelle besteht aus einer Point_ID, einem Zeitstempel im Format JJJJ-MM-TT hh:mm:ss und dem Wert im Format float oder integer.
Wir gehen ab hier davon aus, dass Sie Ihre Schnittstelle auf Grundlage des Scriptes **DH_basic_interface.py** schreiben.
Die Schnittstelle durchläuft nach der Initialisierung den Teil, in dem die Daten von der Quelle gelesen werden und verweilt dann so lange in einer Schleife, bis die Zeit für einen Intervall abgelaufen ist. Dabei schaut es all fünf Sekunden nach, ob irgend eine Meldung für das Script vorliegt. Ist das der Fall, dann unterbricht es die Warteschleife und verarbeitet die Meldung.
In jeder Schnittstelle ist das Modul *DH_biblio2.py* eingebunden. Dieses Script enthält die Funktionen zur Kommunikation mit dem DH. Es kümmert sich somit um die Meldungen vom DH, das Schreiben der Daten zum DH, das Einlesen der Konfiguration und der Points. Falls ein Server nicht erreichbar ist, sorgt es auch für die Pufferung in der lokalen Datenbank.
Wenn Sie eine neue Schnittstelle bauen möchten, dann brauchen Sie oft nur noch den Teil schreiben, der die Daten aus der Quelle liest. Bei Bedarf können Sie aber auch im Bereich Initialisierung noch weitere Module importieren oder globale Variablen festlegen.

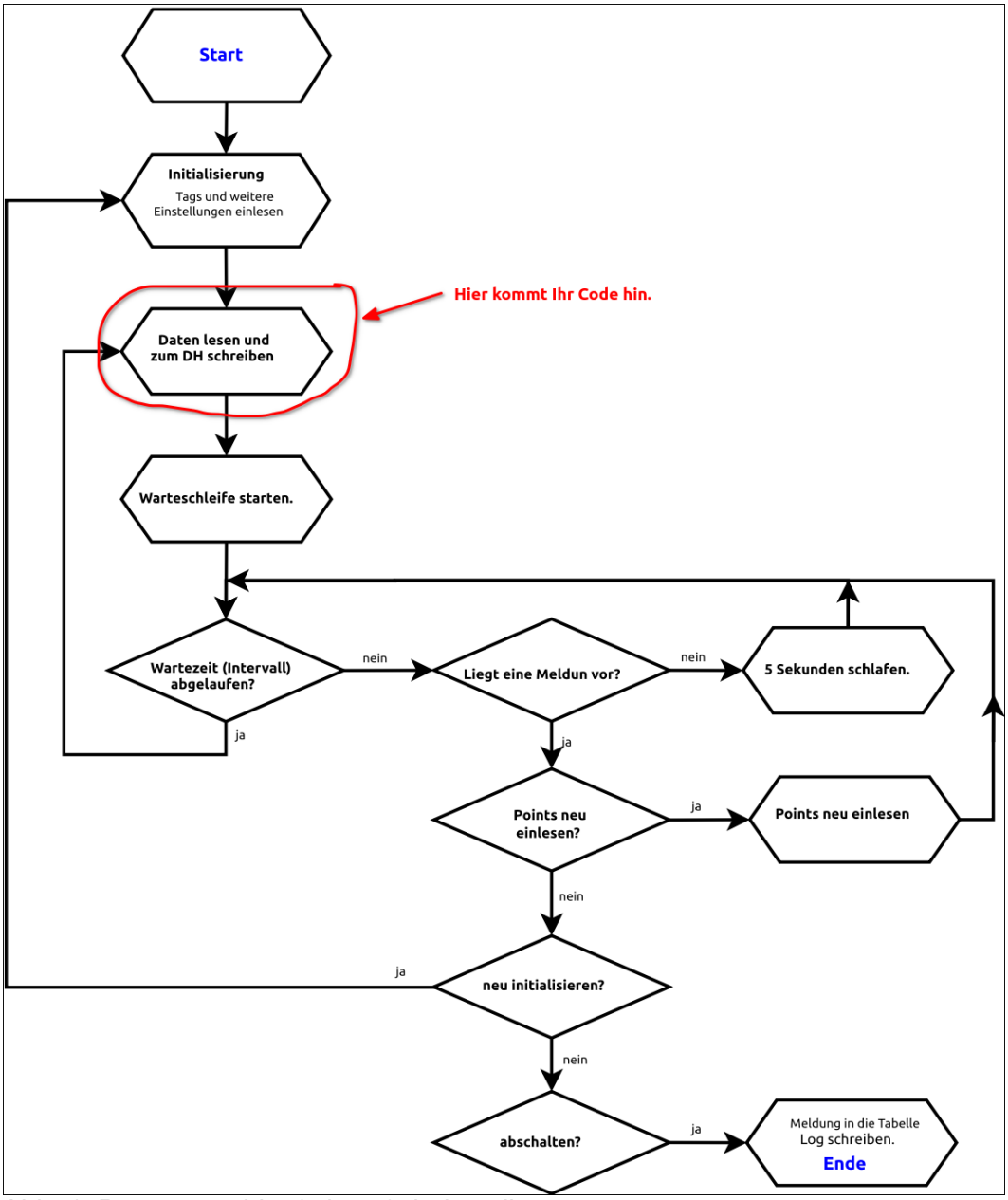


Abb. 1: Programmablauf einer Schnittstelle

2. Konfiguration der Schnittstelle

Bei der Initialisierung der Schnittstelle liest die gleichnamige Funktion aus dem *DH_biblio2.py* - Script die Einstellungen, so wie sie im Formular Schnittstellen (Admin - Bereich) eingetragen wurden.

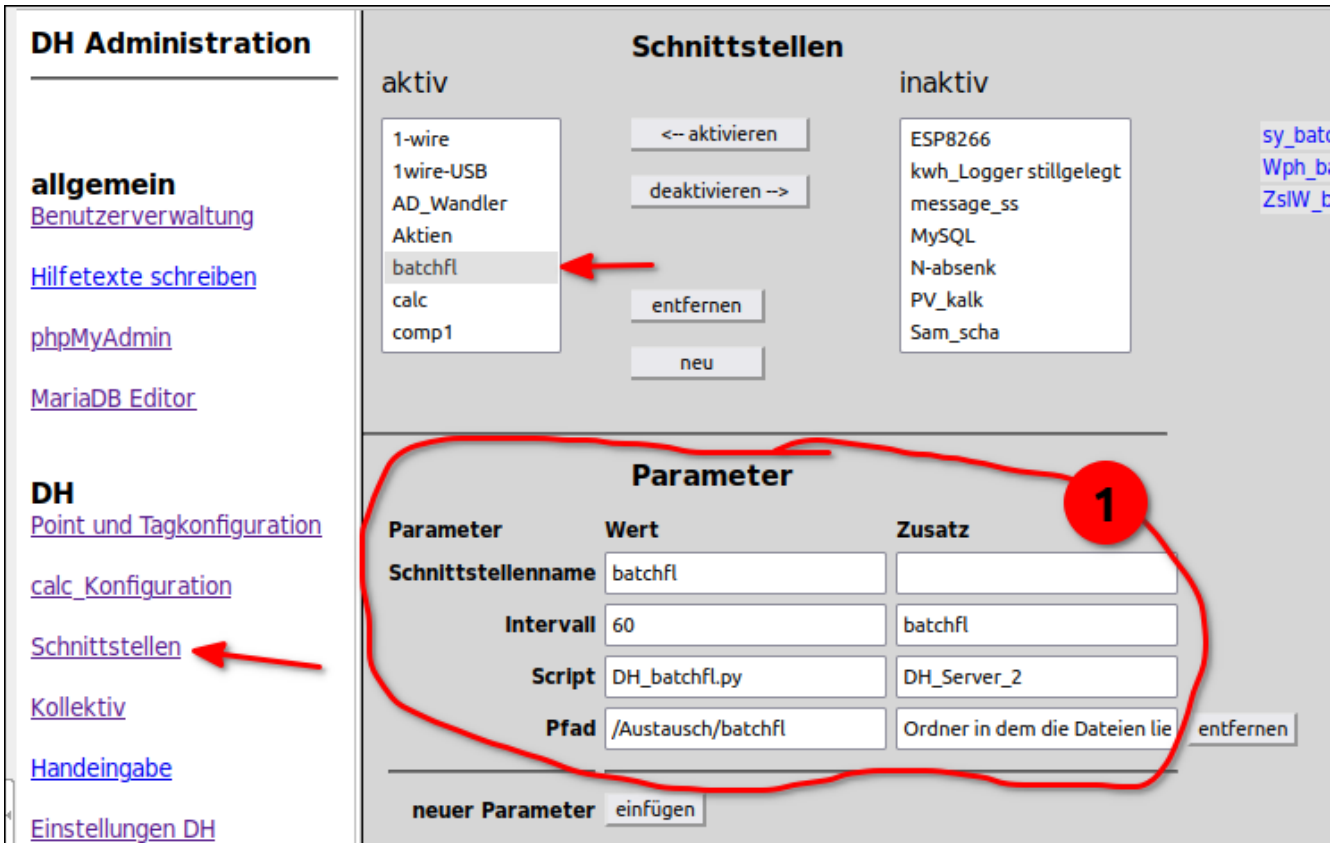


Abb. 2: Konfiguration der Schnittstelle *batchfl*.

Die Parameter, die Sie im markierten Bereich der Abb. 2 sehen, können Sie in der Schnittstelle mit der Funktion **DH_biblio2.Einstellung_lesen('Parameter')** lesen. *DH_biblio2.Einstellung_lesen('Pfad')* liefert somit den Wert *Austausch/batchfl.*
Der Intervall wird von jeder Schnittstelle benötigt und wird als Variable im Modul DH_biblio2 vorgehalten (*DH_biblio2.Intervall*).
Jede Schnittstelle liest während der Initialisierung die Konfiguration aller Points ein, deren Eigenschaft **Schnittstelle** gleich dem Namen der Schnittstelle ist und die mit **scan = 1** konfiguriert sind. DH_biblio2 legt die Pointkonfiguration im Array **DH_biblio2.TL** ab. DH_biblio2.TL[0]['Point_ID'] liefert Ihnen die Point_ID des ersten Points im Array. Die Reihenfolge der Points im Array können Sie über den Parameter *Property_1* in der Pointkonfiguration festlegen. *DH_biblio2* sortiert die Points nach diesem Parameter in aufsteigender Reihenfolge.

3. DH_biblio2 und deren Funktionen

DH_biblio2.py besteht aus insgesamt 7 Funktionen, von denen eine Funktion *Kollektiv_verbinden()* nur für interne Zwecke bestimmt ist.

Hier die restlichen 6 Funktionen:

1. **meldung(Interface):**
   Liest die Meldungen der Server, die vom *watchdog* als kleine Textdatei im Unterordner *tmp* abgelegt werden. Diese Funktion wird in der Warteschleife alle 5 Sekunden aufgerufen:
   DH_biblio2.meldung(Interface_Name)

2. **Initialisierung(Interface):**
   Liest die Konfiguration der Points, den Intervall, den Zwangsintervall und weitere individuelle Parameter aus der Schnittstellenkonfiguration ein. Die Points zur Überwachung der Schnittstelle werden hier auch noch festgelegt.

3. **Wert_schreiben(Point_ID, Zeitstempel,Wert):**
   Schreibt einen Wert in den DH. Der Wert wird auf allen Servern im Kollektiv in die Tabelle *akt* geschrieben. Ist ein Server nicht erreichbar, dann schreibt sie den Wert in den Puffer der lokalen Datenbank.

4. **log_schreiben(Interface, Meldung):**
   Schreibt eine Zeile in die Tabelle Log der Server. Der Zeitstempel für den Eintrag entspricht stets dem aktuellen Zeitpunkt. Die Meldung kann bis zu 65535 Zeichen lang sein. Ich hoffe doch sehr, dass Sie diese nicht voll ausnutzen ;-).

5. **Einstellung_lesen(Parameter):**
   Liest zusätzliche Parameter aus der Schnittstellenkonfiguration.

6. **schreiben(SQL_Text):**
   Das ist die allmächtige Funktion, die ohne weitere Nachfrage das übergebene SQL - Statement auf den Servern des Kollektivs ausführt. Es ist sicherlich sinnvoll, diese Schnittstelle nicht mit root - Rechten zu starten.

4. DH_basic_interface.py

```
#!/usr/bin/python3 <-- Info für den Python Interpreter
import DH_biblio2, sys, time <-- Import der benötigten Module
Script = sys.argv[0] <-- Name des Scripts ermitteln
Interface_Name = Script[Script.find("DH_") + 3:len(Script) - 3] <-- Name der Schnittstelle
                                                                  aus dem Scriptname
                                                                  ermitteln.
#initialisation
Initialisierung = DH_biblio2.Initialisierung(Interface_Name) <-- Points und Einstellungen
                                                                einlesen.
#possibly further initialisation


Punkt 1


#end of initialisation
#start of the endless loop
Meldung = "weiter" <-- Meldung != „abschalten“ setzen.
while Meldung != "abschalten": <-- so lange im Kreis drehen, bis die Meldung == „abschalten“ ist.
    #Here starts your code



     Punkt 2



    #Here ist the end of your code
    #Listen to the message subsystem <-- Punkt 3
    horchen_bis = time.time() + DH_biblio2.Intervall
    while time.time() < horchen_bis:
        Meldung = DH_biblio2.meldung(Interface_Name)
        if Meldung != "weiter":
            horchen_bis = time.time() - 1000000
        else:
            time.sleep(5)
DH_biblio2.log_schreiben(Interface_Name, "Interface gestoppt") <-- Log - Eintrag schreiben
                                                                  und das Programm
                                                                  beenden.
```

Punkt 1:
Hier ist Gelegenheit, weitere Einstellungen zum Programmstart vorzunehmen. Sie können hier z.B. Variablen generieren oder auch Funktionen unterbringen. Hier gehört alles hin, was beim Start der Schnittstelle erledigt werden muss.

Punkt 2:
Das ist der Teil, in dem die Daten aus der Quelle gelesen und in den DH geschrieben werden. Es ist also der individuelle Teil der Schnittstelle.

Punkt 3:
Hier beginnt die Warteschleife. Die Variable *horchen_bis* repräsentiert den Zeitpunkt, bis zu dem die Warteschleife läuft. Innerhalb der Schleife wird geprüft, ob eine Meldung vorliegt. Ist das nicht der Fall, dann liefert die Funktion `DH_biblio2.meldung(Interface_Name)` den Text „weiter“ zurück. Wurde jedoch eine Meldung empfangen, dann wird der Wert der Variablen *horchen_bis* so weit herabgesetzt, dass der aktuelle Unix - Zeitstempel größer ist als *horchen_bis* und somit die Schleife vorzeitig beendet.
Mit Ausnahme der Meldung „abschalten“ werden alle Meldungen innerhalb der Funktion `DH_biblio2.meldung(Interface_Name)` verarbeitet.

5. Beispiel

Nehmen wir an, wir wollen alle 10 Minuten Werte aus einer Datei lesen, die von irgend einer anderen Anwendung regelmäßig aktualisiert wird. Irgendwo in dieser Datei steht „*Status: 123*“. Dieser Zahl weisen wir den Point „*Status*“ zu. Die Schnittstelle bekommt den Namen Status. Wir benötigen also ein Script mit dem Namen *DH_Status.py*.
Wenn wir uns nun überlegen, wie wir die Schnittstelle und den Point konfigurieren, dann kann die Konfiguration der Schnittstelle etwa so aussehen:

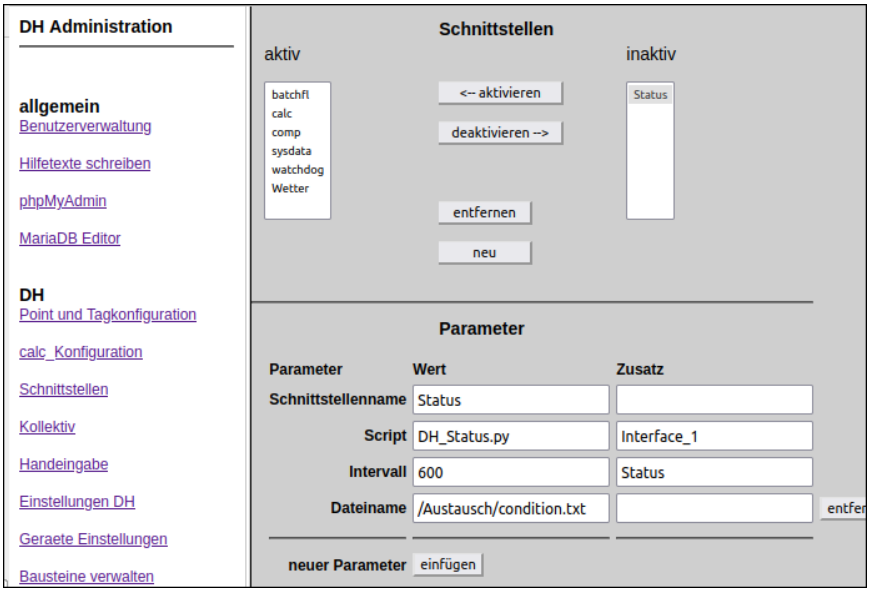


Abb. 3: Konfiguration der Schnittstelle Status.

Die Schnittstelle soll auf dem Rechner *Interface_1* laufen. Der Intervall beträgt 600 Sekunden und als zusätzliche Einstellung fügen wir den Parameter „*Dateiname*“ ein, welcher den Dateinamen inklusive dem Pfad zur Datei angibt, aus der wir den Wert lesen möchten.
Wir könnten ebenso gut den Dateinamen als Variable am Anfang des Scriptes eintragen. Das würde auch funktionieren, ist aber weniger elegant, da ein Administrator das Script ändern müsste, falls sich beim Pfad oder Dateiname mal etwas ändern sollte. Die erste Stelle, an der ein Admin sucht, ist sicherlich die Konfiguration im Admin - Bereich.
Den Point konfigurieren wir etwa so:

- Dezimalstellen: 0
- Mittelwerte: 0
- step: 1
- scan: 1
- Interface: Status

Der Rest der Konfiguration ist nicht so wichtig.

Jetzt zum Code:

Initialisierung:
Hier fügen wir folgende Zeilen ein:

```
import re
Dateiname = DH_biblio2.Einstellung_lesen('Dateiname')
```

Daten lesen und schreiben (Punkt 2):

```
#Datei oeffnen und in die Variable Inhalt einlesen
Datei = open( ‘Dateiname‘, 'r')
Inhalt = Datei.read()
Datei.close()

#Den Text aus der Variablen Inhalt bis zum Anfang des Wertes abschneiden.
Ergebnis = re.search(’Status: ’,Inhalt)
Inhalt = Inhalt[Ergebnis.span()[1] + 8:len(Inhalt)]

#jetzt die Zahlen lesen
Wert = ’’
i = 0
while ord(Inhalt[i:i + 1]) > 47 and ord(Inhalt[i:i + 1]) < 58:
    Wert = Wert + ord(Inhalt[i:i + 1])
    i = i + 1

#Dateityp der Variablen Wert von Text nach int wandeln.
Wert = int(Wert)

#Den aktuellen Zeitstempel ermitteln.
Zeitp = str(time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(time.time())))

#Wert in den DH schreiben
DH_biblio2.Wert_schreiben(DH_biblio2.TL[0]['Point_ID'], Zeitp, Wert)
#Da nur ein Point fuer unsere Schnittstelle konfiguriert wurde, ist
#DH_biblio2.TL[0] der richtige Point. Sollten noch weitere Points
#hinzukommen, dann muss hier eine andere Loesung her. Aber fuer das #Beispiel
#sollten wir die Sache nicht unnoetig kompliziert machen.
```

Erweiterung der Schnittstelle:
Möchten Sie Werte für mehrere Points mit dieser Schnittstelle einlesen, dann müssen Sie wissen, welcher Point an welcher Stelle im Array *DH_biblio2.TL* steht. Das können Sie über die Konfiguration der Points erledigen. Im Array *DH_biblio2.TL* sind die Points nach dem Parameter *Property_1* aufsteigend sortiert.

Plausibilitätsprüfung einbauen:
Wenn es vorkommt, dass die Anwendung, welche die Datei */Austausch/condition.txt* schreibt, dort ab und zu Müll reinschreibt, dann sollte man noch eine Plausibilitätsprüfung einbauen. Wenn Sie z.B. für den Status einen Wert zwischen 100 und 200 erwarten, dann bietet es sich an, zu überprüfen, ob der eingelesene Wert tatsächlich in diesem Bereich liegt. In der Pointkonfiguration gibt es fünf Property - Felder. *Property_1* benutzen wir bereits für die Sortierung der Points. Die restlichen sind noch frei. Konfigurieren wir die Points nun so, dass in *Property_2* der untere Grenzwert und in *Property_3* der obere Grenzwert steht, dann kann man eine Plausibilitätsprüfung recht einfach einbauen:

```
if Wert >= DH_biblio2.TL[0]['Property_2'] and Wert <= DH_biblio2.TL[0]['Property_3']:
      DH_biblio2.Wert_schreiben(DH_biblio2.TL[0]['Point_ID'], Zeitp, Wert)
```

Der Code einer echten Schnittstelle ist selbstverständlich deutlich komplexer. Das Beispiel ist aber bewusst sehr einfach gehalten, um nicht unnötiger Weise für Irritation zu sorgen.