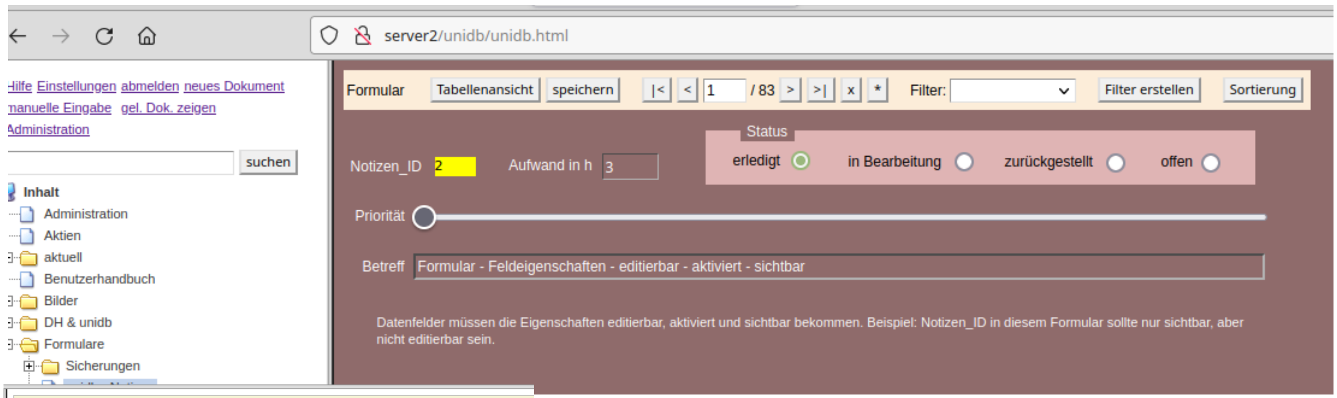
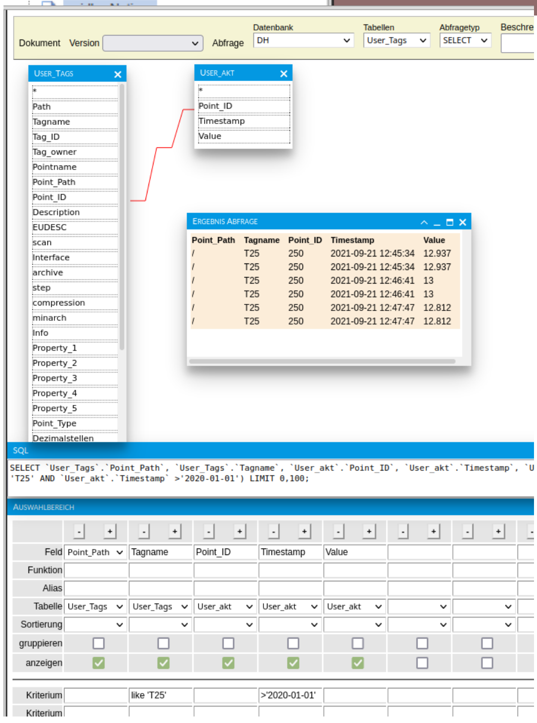
DH Interfaces programming

Version 1.0



Notizen_ID	2
Aufwand in h	3
Status	[✓ erledigt] in Bearbeitung zurückgestellt offen
Betreff	Formular - Feldeigenschaften - editierbar - aktiviert - sichtbar

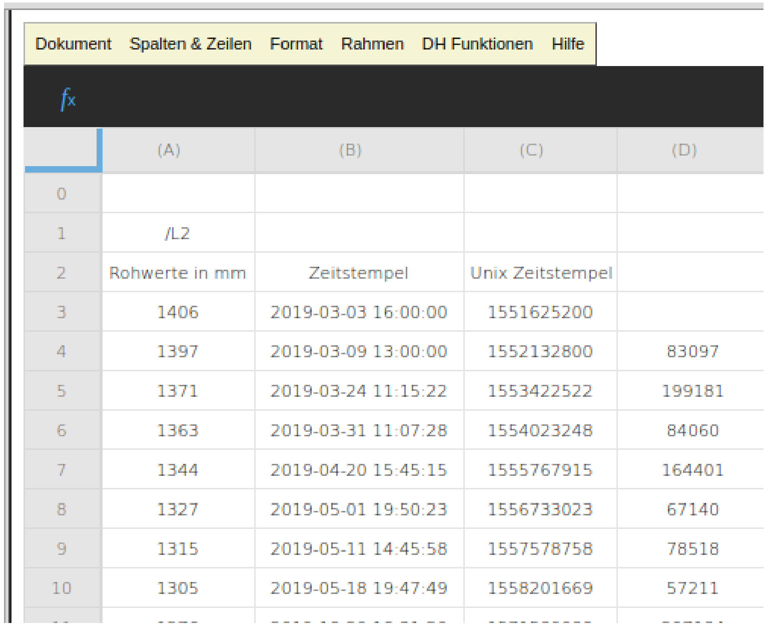
Datenfelder müssen die Eigenschaften editierbar, aktiviert und sichtbar bekommen. Beispiel: Notizen_ID in diesem Formular sollte nur sichtbar, aber nicht editierbar sein.



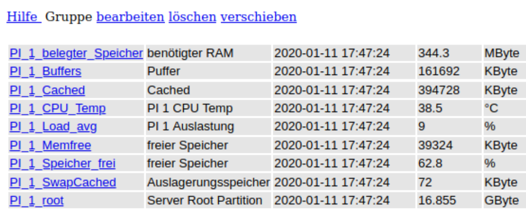
Point_Path	Tagname	Point_ID	Timestamp	Value
/	T25	250	2021-09-21 12:45:34	12.937
/	T25	250	2021-09-21 12:45:34	12.937
/	T25	250	2021-09-21 12:46:41	13
/	T25	250	2021-09-21 12:46:41	13
/	T25	250	2021-09-21 12:47:47	12.812
/	T25	250	2021-09-21 12:47:47	12.812

```
SELECT `User_Tags`.`Point_Path`, `User_Tags`.`Tagname`, `User_akt`.`Point_ID`, `User_akt`.`Timestamp`, `U
'T25' AND `User_akt`.`Timestamp` >'2020-01-01') LIMIT 0,100;
```

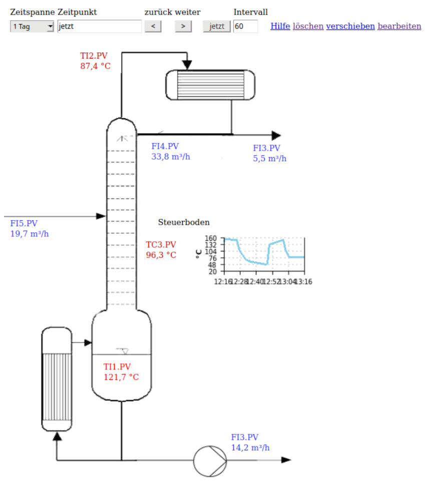
Feld	Point_Path	Tagname	Point_ID	Timestamp	Value		
Funktion							
Alias							
Tabelle	User_Tags	User_Tags	User_akt	User_akt	User_akt		
Sortierung							
gruppieren	☐	☐	☐	☐	☐	☐	☐
anzeigen	✓	✓	✓	✓	✓	☐	☐
Kriterium		like 'T25'		>'2020-01-01'			
Kriterium							

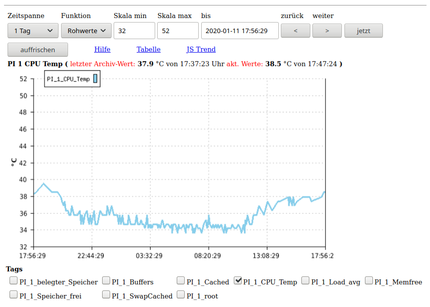


	(A)	(B)	(C)	(D)
0				
1	/L2			
2	Rohwerte in mm	Zeitstempel	Unix Zeitstempel	
3	1406	2019-03-03 16:00:00	1551625200	
4	1397	2019-03-09 13:00:00	1552132800	83097
5	1371	2019-03-24 11:15:22	1553422522	199181
6	1363	2019-03-31 11:07:28	1554023248	84060
7	1344	2019-04-20 15:45:15	1555767915	164401
8	1327	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





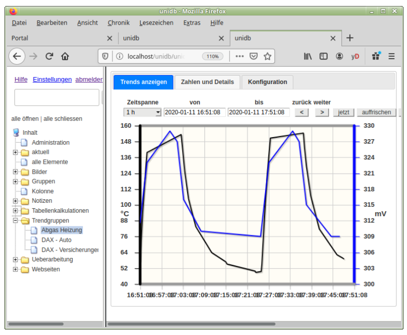


Table of contents

Terms and abbreviations

Tag	Data point in a DataHistorian
Point	Another name for a tag / data point.
Point_ID	Unique number of a tag, comparable to a serial number.

Formatting used

bold font	Name of a control element (switch, text field, ...)
italic	Names of properties, etc.
highlighted in color	You should pay a little more attention to a text with a colored background.
`courier`	code

1. Introduction

How you get your data into the DH is basically up to you. However, it is good practice to use the **DH_basic_interface.py** script for a new interface. It already contains all the components required to communicate with the DH. However, if you go your own way, please only write the data to the table **akt** on all servers in the collective. Each entry in this table consists of a Point_ID, a timestamp in YYYY-MM-DD hh:mm:ss format and the value in float or integer format.
From here we assume that you are writing your interface based on the **DH_basic_interface.py** script. After initialisation, the interface goes through the part where the data is read from the source and then stays in a loop until the time for an interval has expired. It checks every five seconds to see whether there is any message for the script. If this is the case, it breaks the queue and processes the message.
The *DH_biblio2.py* module is integrated into every interface. This script contains the functions for communication with the DH. It therefore takes care of the messages from the DH, writing the data to the DH, reading in the configuration and the points. If a server is not reachable, it also ensures buffering in the local database.
If you want to build a new interface, you often only need to write the part that reads the data from the source. If necessary, you can also import additional modules or set global variables in the initialisation area.

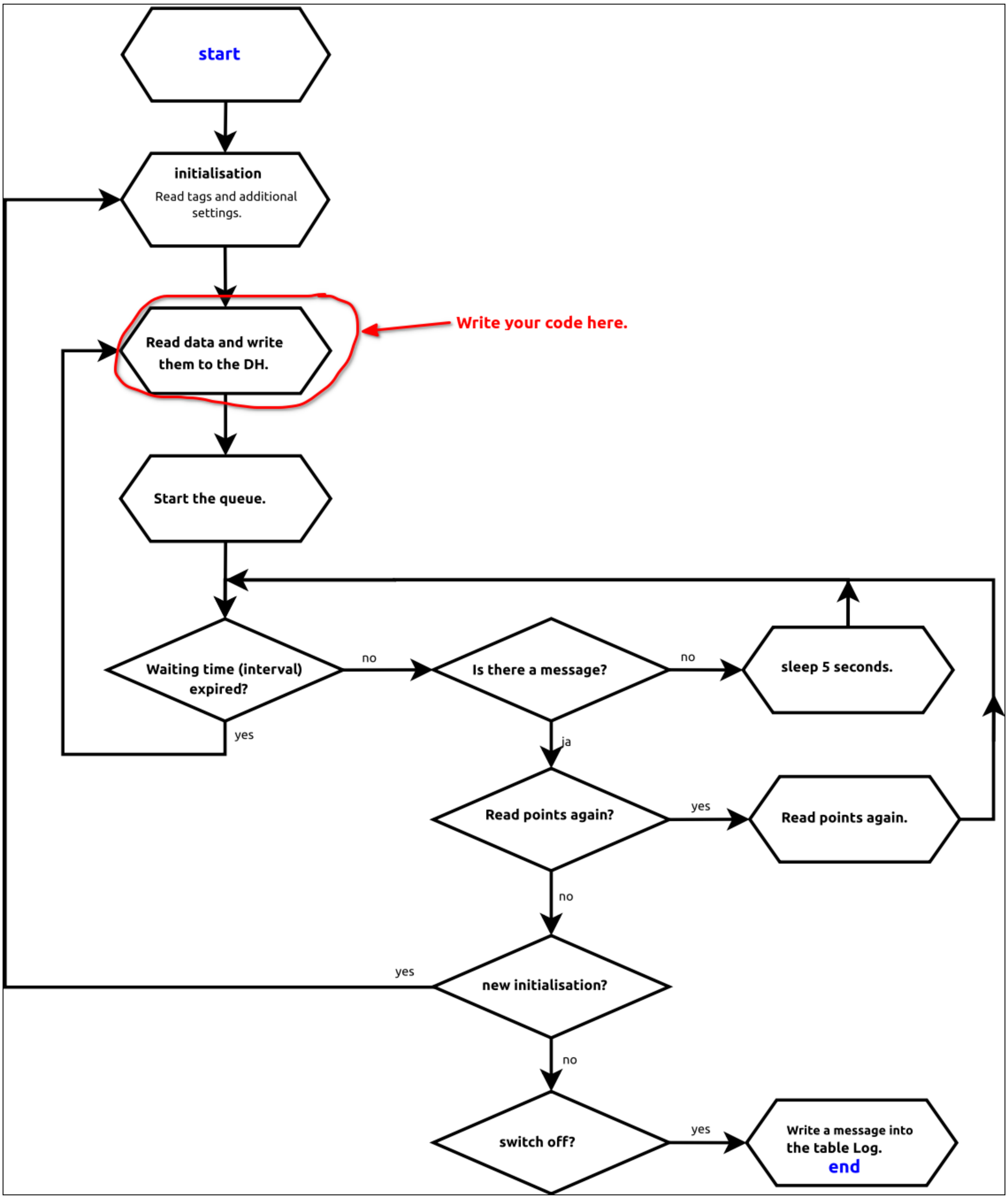
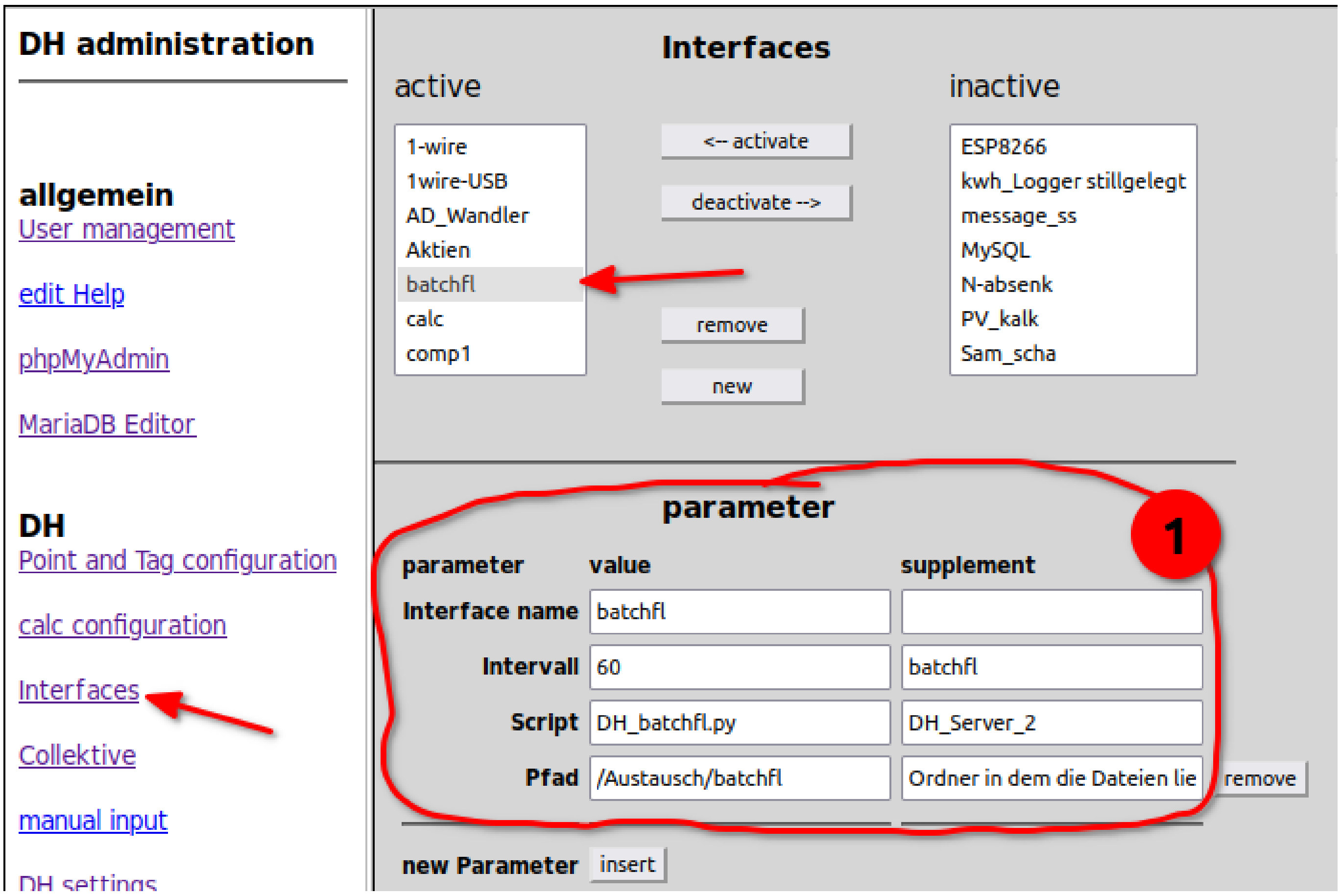


Fig. 1: Program flow of an interface

2. Configuration of the interface

When initialising the interface, the function of the same name from the *DH_biblio2.py* script reads the settings as they were entered in the *interfaces* form (admin area).



DH administration

allgemein
User management
edit Help
phpMyAdmin
MariaDB Editor

DH
Point and Tag configuration
calc configuration
Interfaces
Collektive
manual input
DH settings

Interfaces

active		inactive
1-wire	<-- activate	ESP8266
1wire-USB	deactivate -->	kwh_Logger stillgelegt
AD_Wandler		message_ss
Aktien		MySQL
batchfl		N-absenk
calc	remove	PV_kalk
comp1	new	Sam_scha

parameter

parameter	value	supplement	
Interface name	batchfl		
Intervall	60	batchfl	
Script	DH_batchfl.py	DH_Server_2	
Pfad	/Austausch/batchfl	Ordner in dem die Dateien lie	remove

new Parameter insert

Fig. 2: Configuration of the *batchfl* interface.

The parameters that you see in the highlighted area of Fig. 2 can be read in the interface with the function **DH_biblio2.Setting_read('Parameter')**. *DH_biblio2.Setting_read('Path')* thus returns the value /Austausch/batchfl.
The interval is required by every interface and is kept as a variable in the *DH_biblio2* module (**DH_biblio2.Interval**).
During initialisation, each interface reads the configuration of all points whose Interface property is the same as the name of the interface and which are configured with **scan = 1**. *DH_biblio2* stores the point configuration in the **DH_biblio2.TL** array. *DH_biblio2.TL[0][‘Point_ID’]* gives you the Point_ID of the first point in the array. You can specify the order of the points in the array using the **Property_1** parameter in the point configuration. *DH_biblio2* sorts the points according to this parameter in ascending order.

3. DH_biblio2 and its functions

DH_biblio2.py consists of a total of 7 functions, one of which is *Kollektiv_verbinden()* for internal purposes only.

Here are the remaining 6 functions:

1. **meldung(Interface):**
   Reads the messages from the servers, which are stored by the *watchdog* as a small text file in the *tmp* subfolder. This function is called every 5 seconds in the queue:
   DH_biblio2.meldung(Interface_Name)

2. **Initialization(Interface):**
   Reads the configuration of the points, the interval, the forced interval and other individual parameters from the interface configuration. The points for monitoring the interface are also defined here.

3. **Wert_schreiben(Point_ID, timestamp,value):**
   Writes a value to the DH. The value is written to the *akt* table on all servers in the collective. If a server is not accessible, it writes the value to the buffer of the local database.

4. **log_schreiben(interface, message):**
   Writes a line to the server's Log table. The timestamp for the entry always corresponds to the current time. The message can be up to 65535 characters long. I really hope you don't take full advantage of them ;-).

5. **Einstellung_lesen(parameter):**
   Reads additional parameters from the interface configuration.

6. **schreiben(SQL_Text):**
   This is the all-powerful function that executes the passed SQL statement on the collective's servers without further request. It certainly makes sense not to start this interface with root privileges.

4. DH_basic_interface.py

```
#!/usr/bin/python3 <-- Info for the Python interpreter
import DH_biblio2, sys, time <-- Import the required modules
Script = sys.argv[0] <-- Determine the name of the script
Interface_Name = Script[Script.find("DH_") + 3:len(Script) - 3] <-- Determine the name of
                                                                     the interface from the
                                                                     script name.
#initialisation
Initialisierung = DH_biblio2.Initialisierung(Interface_Name) <-- Read in points and settings.

#possibly further initialisation


point 1


#end of initialisation
#start of the endless loop
Meldung = "weiter" <-- set Meldung != „abschalten“.
while Meldung != "abschalten": <-- Turn in circles until Meldung == „abschalten“.
    #Here starts your code



    point 2



    #Here ist the end of your code
    #Listen to the message subsystem <-- point 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") <-- Write log entry and exit
                                                                  the program.
```

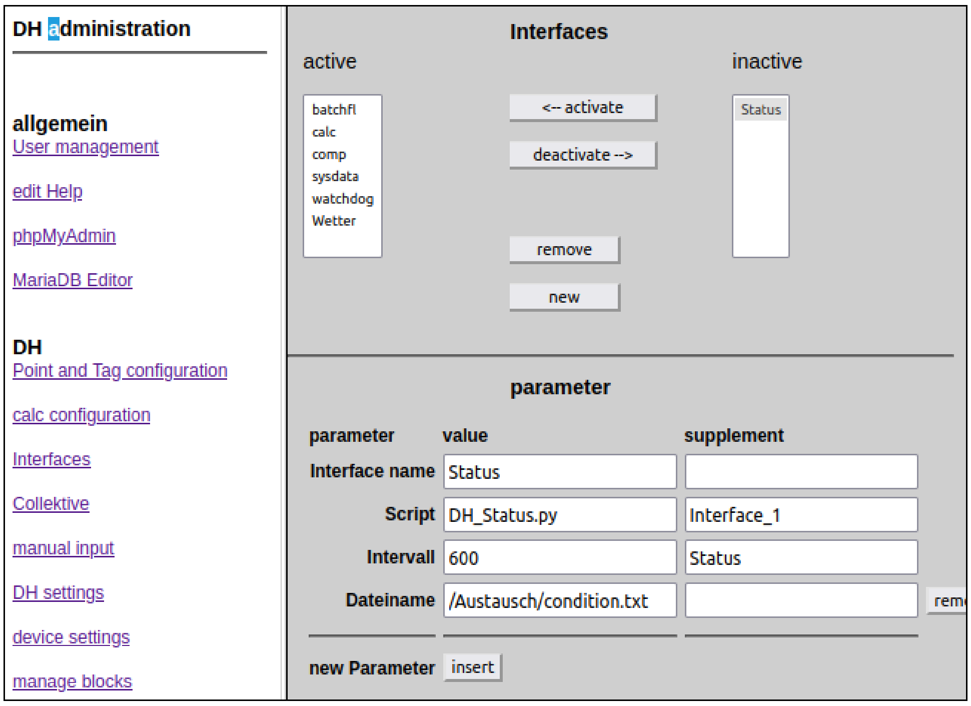
Point 1:
This is an opportunity to make further settings at the start of the program. Here you can, for example, generate variables or accommodate functions. Everything that needs to be done when starting the interface belongs here.

Point 2:
This is the part where the data is read from the source and written to the DH. So it is the individual part of the interface.

Point 3:
The queue begins here. The variable *horchen_bis* represents the time until which the waiting loop runs. The loop checks whether there is a message. If this is not the case, the function `DH_biblio2.melde(Interface_Name)` returns the text “weiter”. However, if a message was received, then the value of the variable *horchen_bis* is reduced to such an extent that the current Unix timestamp is greater than *horchen_bis* and thus the loop ends prematurely.
With the exception of the “abschalten” message, all messages are processed within the `DH_biblio2.message(Interface_Name)` function.

5. Example

Let's say we want to read values every 10 minutes from a file that is regularly updated by some other application. Somewhere in this file it says “*Status: 123*”. We assign the point “*Status*” to this number. The interface is named Status. So we need a script called *DH_Status.py*.
If we now think about how we configure the interface and the point, then the configuration of the interface can look something like this:



Interfaces

active: batchfl, calc, comp, sysdata, watchdog, Wetter

inactive: Status

parameter

parameter	value	supplement
Interface name	Status	
Script	DH_Status.py	Interface_1
Intervall	600	Status
Dateiname	/Austausch/condition.txt	

Fig. 3: Configuration of the interface status.

The interface should run on the computer Interface_1. The interval is 600 seconds and as an additional setting we add the “Filename” parameter, which specifies the file name including the path to the file from which we want to read the value.
We could just as easily enter the file name as a variable at the beginning of the script. That would also work, but is less elegant because an administrator would have to change the script if something changes in the path or file name. The first place an admin looks is certainly the configuration in the admin area.
We configure the point something like this:

- Decimal places: 0
- Mean values: 0
- step: 1
- scan: 1
- Interface: Status

The rest of the configuration is not that important.
Now for the code:

Initialisation:
Here we add the following lines:

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

Read and write data (point 2):

```
#Open the file and read the content into the variable
Datei = open( ‘Dateiname‘, 'r')
Inhalt = Datei.read()
Datei.close()

#Trunk the text from the Content variable to the beginning of the value.
Ergebnis = re.search('Status: ',Inhalt)
Inhalt = Inhalt[Ergebnis.span()[1] + 8:len(Inhalt)]

#read the numbers now
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

#Convert file type of variable value from text to int.
Wert = int(Wert)

#Get the current timestamp.
Zeitp = str(time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(time.time())))

#Write the value into the DH.
DH_biblio2.Wert_schreiben(DH_biblio2.TL[0]['Point_ID'], Zeitp, Wert)
#Since only one point was configured for our interface, DH_biblio2.TL[0] is
#the correct point. If more points are added, then a different solution will
#have to be found. But for the example we shouldn't make things unnecessarily
#complicated.
```

Expansion of the interface:
If you want to read in values for several points with this interface, you need to know which point is at which point in the *DH_biblio2.TL* array. You can do this by configuring the points. In the *DH_biblio2.TL* array, the points are sorted in ascending order according to the *Property_1* parameter.

Install plausibility check:
If it happens that the application that writes the file */Austausch/condition.txt* occasionally writes garbage into it, then you should add a plausibility check. For example, if you expect a value between 100 and 200 for the *status*, then it is a good idea to check whether the value read is actually in this range. There are five property fields in the point configuration. We already use *Property_1* for sorting the points. The rest are still free. If we now configure the points so that the lower limit value is in *Property_2* and the upper limit value is in *Property_3*, then you can easily incorporate a plausibility check:

```
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)
```

The code of a real interface is of course much more complex. However, the example is deliberately kept very simple so as not to cause unnecessary confusion.