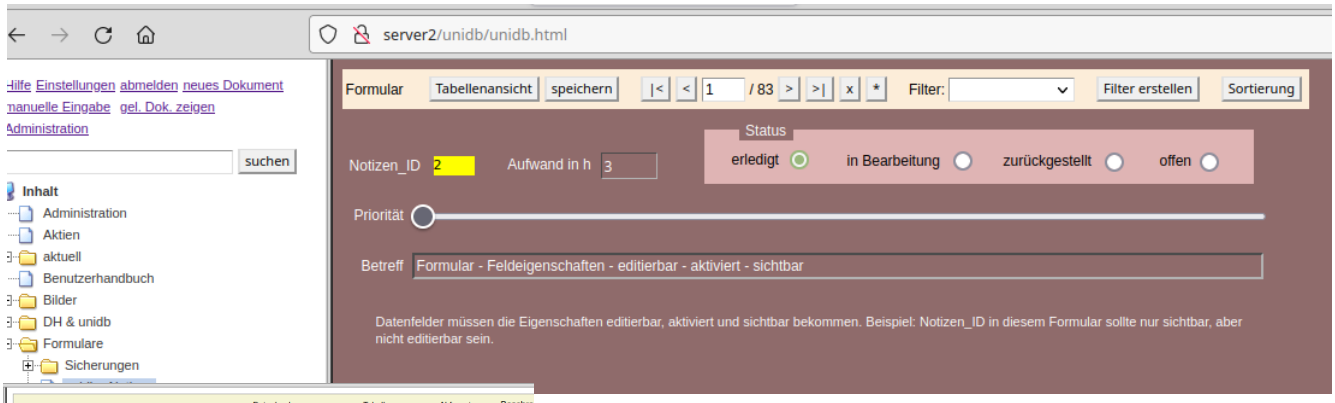
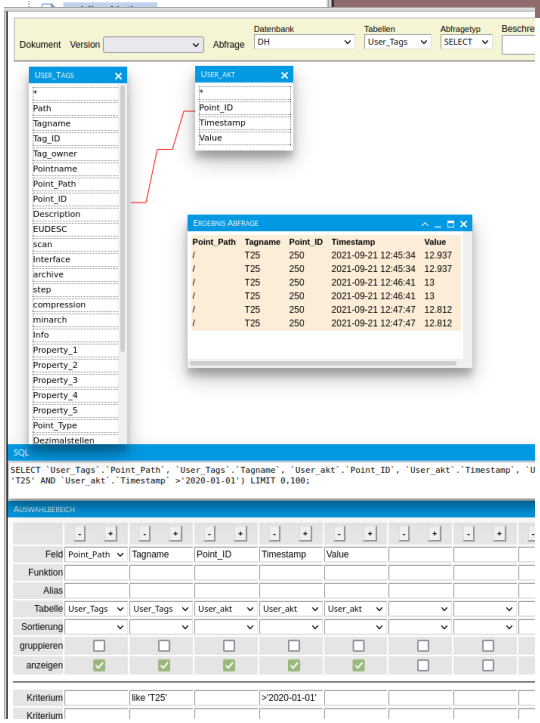
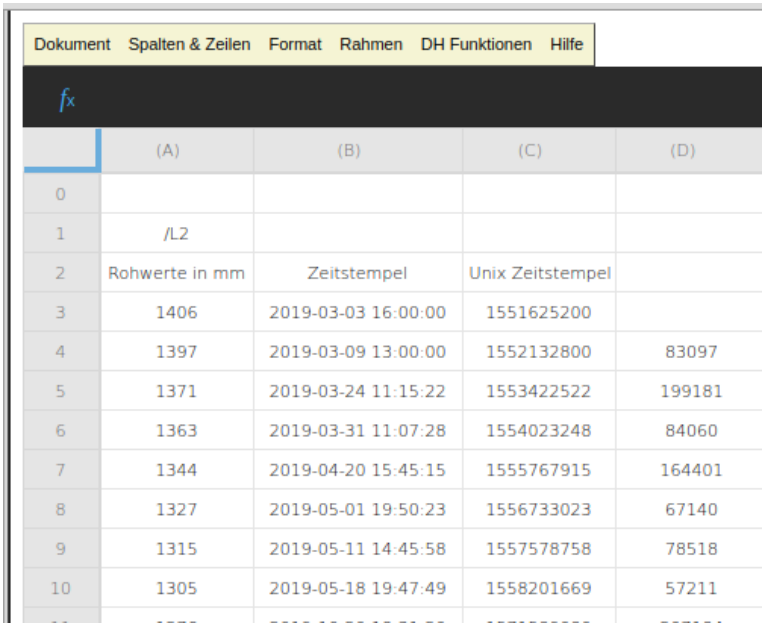
unidb und DH
Administration Guide

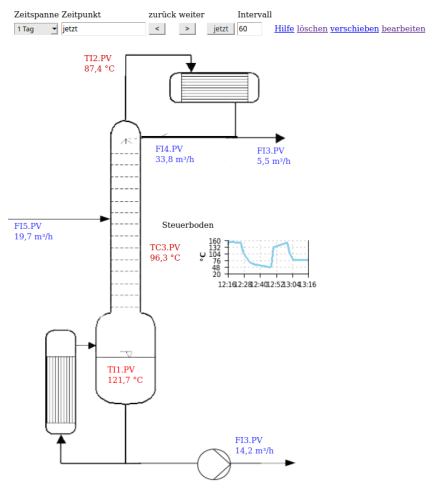
Version 1.0

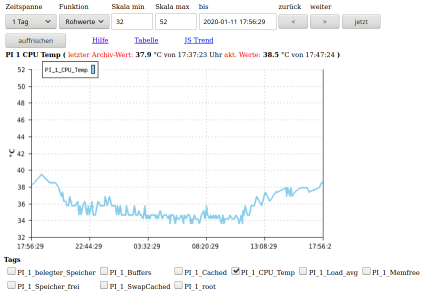






	(A)	(B)	(C)	(D)
0				
1	/L2			
2	Rohwerte in mm	Zeitstempel	Unix Zeitstempel	
3	1406	2019-03-03 16:00:00	1551625200	
4	1397	2019-03-09 13:00:00	1552132800	83097
5	1371	2019-03-24 11:15:22	1553422522	199181
6	1363	2019-03-31 11:07:28	1554023248	84060
7	1344	2019-04-20 15:45:15	1555767915	164401
8	1327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





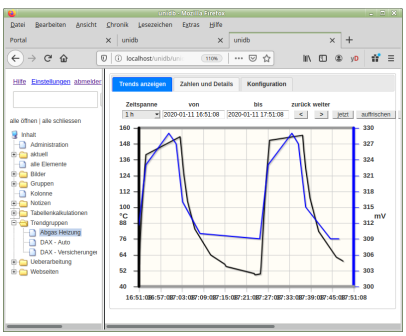


Table of contents

Terms and abbreviations

DataHistorian	Database application which continuously collects data in the format data point - time stamp - value and makes this available again in a processed form.
DH	Figreviation for DataHistorian
TimeSeriesDatabase	An alternative name for a DataHistorian.
DB	Figreviation for database.
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.
WYSIWYG	**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et (What you see is what you get.) There are two types of HTML editors. The simplest way is the one where you can see the HTML code and have to write it yourself. The second type is the type that is easier for the user. The HTML code remains hidden and instead you can see the fully formatted text (WYSIWYG).

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.

1. generally

Administration takes place almost exclusively via the administration page. This can either be accessed directly by appending /admin to the unidb web address or via the **Administration** link. The link is to the right of the logo. If you cannot find the link there, then you do not have administration rights. In this case, a direct call to the admin page will only show you the registration form. In exceptional cases and if you know exactly what you are doing, you can also use the two databases (DH and unidb) with a general database administration tool. Typical representatives of such applications are phpMyAdmin, DBeaver or DBGate. However, keep in mind that in a collective you have to make every change on each server in the collective separately.

1.1. Archive

The data archive is usually located in the /var/lib/DH folder. The data of each point is stored in files whose names correspond to the Point_ID of the point. Each file contains the values of one month. The files for each data type are located in their own folder. This results in the following folder hierarchy under /var/lib/DH:

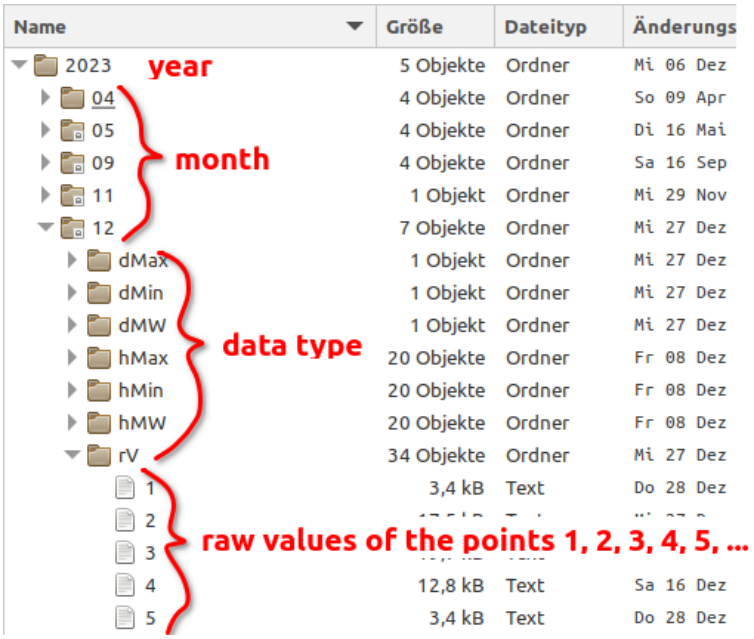


Fig. 1: Hierarchical structure of the archive.

1.2. Collective and resiliency

Both the unidb and the DH always work in a collective. In the simplest case, the collective consists of just one computer. This doesn't sound sensible at first, but it plays a role in data buffering. However, you usually run a collective with several computers. These are servers (DH and/or unidb) and interface computers. The latter only runs interfaces for the DH.

DH - Collective:
An interface for the DH delivers the data in parallel to all servers in the collective. If a server is not accessible, the data for this server is stored locally on the interface computer. As soon as the server is accessible again, the cached data is delivered to the server.
Any changes you make on the administration page will be reflected on all servers in the collective. If a server is not accessible, the changed configuration is subsequently written to the server in question. Although this allows you to make changes if a server is unavailable, it is always safer to avoid this.
Servers working in parallel as a collective ensure a relatively high level of reliability. There are three options to achieve this security on the interface computers:

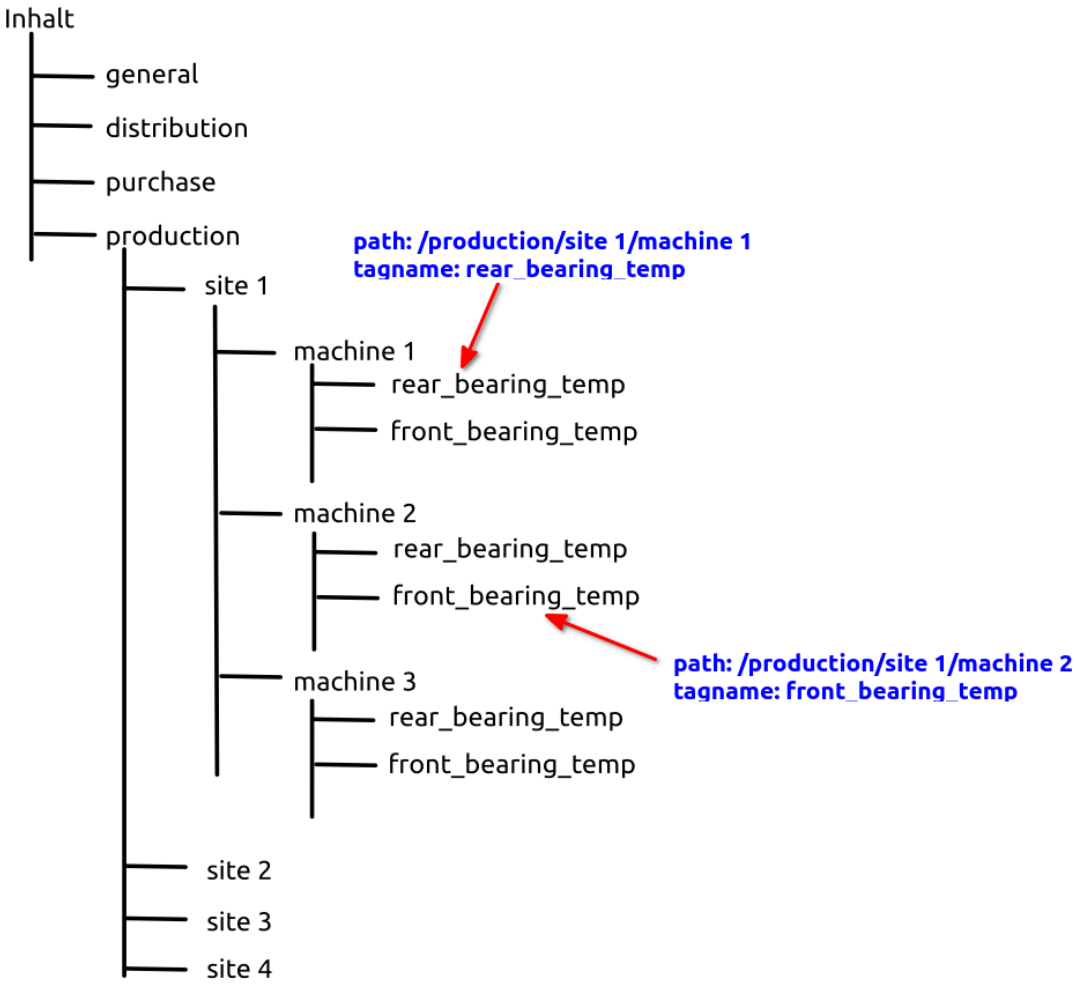
cold backup: For each interface computer there is a second computer that has the same configuration as the first interface computer. It is started manually when the first interface computer fails.
warm backup: The second interface computer runs parallel to the first computer. A script on this computer constantly checks whether the first computer is still providing data. If this is no longer the case, it starts its own interfaces.

hot backup: Both interface computers run parallel to each other and constantly deliver data to the servers. The compressor on the server ensures that the data does not go into the archive twice. Important: If you use method 2 or 3, you must give the interfaces different names and also configure each interface separately.

unidb - Collective:
Any changes made to the documents by users will be propagated to all members of the collective. If a server is not accessible, the changes are cached on the server on which the change was made. As soon as the server in question is accessible again, the changes will be added. Even if several servers are not accessible and changes have been made to the documents on several servers, the synchronization takes place as soon as a server is accessible again. All servers in the collective write their changes to an input buffer on the previously failed server. As soon as the buffers (output) on all members of the collective are empty, the previously failed server begins to adopt the changes from the input buffer in chronological order. This means that the last change to a document wins. So if at least two changes were made to a document on different servers, the most recent change will be applied to the previously failed server. This is not always entirely correct, but it should happen very rarely.

1.3. Planning

Before you set up a new server or a new collective, you should think about the organization. It's about building the tree structure. If you later realise that the structure of the tree structure is not as successful as you would have liked, you will either have to do a lot of work or have to live with it. So if you don't like working, you have to think a lot!
All user permissions depend on the tree structure. The tree structure also plays a major role in assigning names for points and tags from the DH. Example: You feed the DH with measured values from similar machines in different locations. All machines have a generator. Two bearing temperatures are measured on each generator. The structure of the tree structure could then look something like this:



or

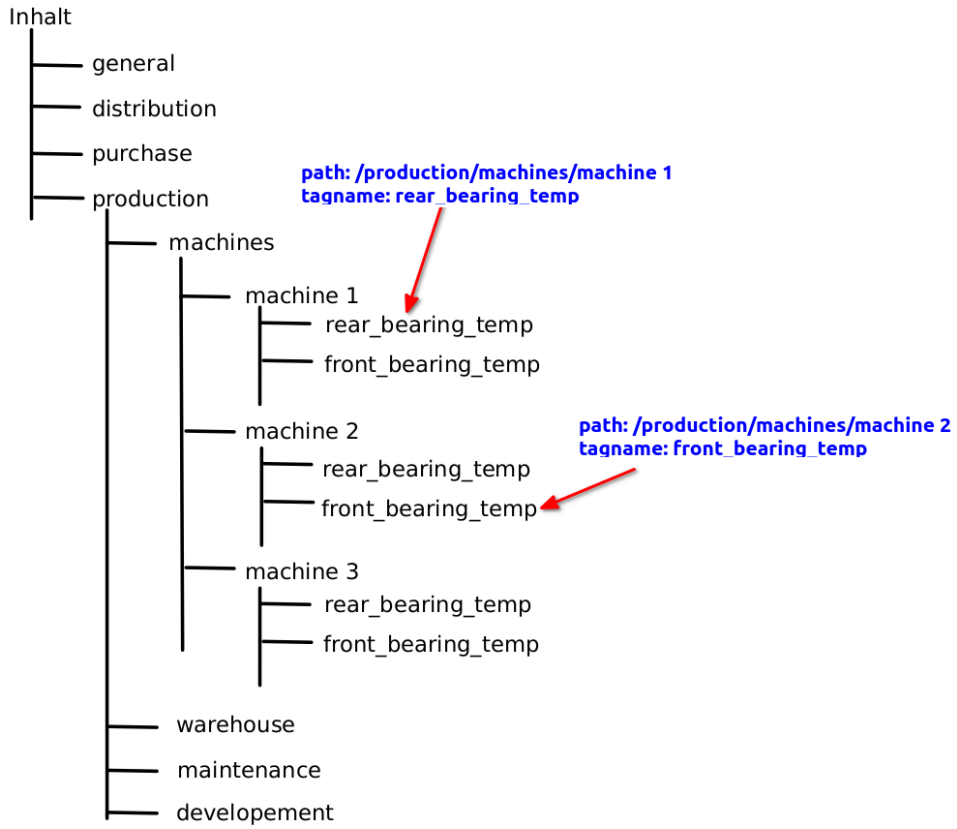


Fig. 2: Organisation of the unidb and the DH

If you want to change the tree structure later, you will have to change the paths of the affected tags and points. Additionally, you need to adjust user access rights. However, the effort required to reorganise the documents in the tree structure is usually completed quickly.

2. unidb

2.1. Users

2.1.1. Paths and user rights

Both in the unidb and in the DH everything is arranged hierarchically. At which point in the hierarchy a document from the unidb, a tag or a point from the DH is located is determined by the assigned path. A tag, a point or a document can only be assigned to one path. The combination of path and tag, point or document is always unique. For example, several tags with the same name can exist if they have different paths. It is always a 1:1 relationship. A user, on the other hand, is always unique. No user name can be assigned twice. However, multiple paths can be assigned to a user. So here we have a 1:n relationship.

An example makes the reason for this clear:
Assume that you have adapted the hierarchy of paths to the structure of your company. The company is spread across three locations. Each location is divided into several departments. Spring location has a shipping department.
The employee Mustermann works in shipping at location 1, but must also be able to see the data from the shipping departments at the other two locations. Here is the assignment of the paths for Mr. Smith:

/Company general <-- This path is assigned to every employee.
/User/Doe <-- The employee's personal directory.
/Location 1 <-- All data from location 1.
/Location 2/Shipping <-- Shipping data location 2.
/Location 3/Shipping <-- Shipping data location 3.

2.1.2. Users of MariaDB and unidb

If a new user is created in the unidb, a new user with the same name is automatically created for the MariaDB. The user name in MariaDB is written completely in lower case.
In addition, there are two other users in the MariaDB in connection with the unidb and the DH. These are the users DH_readonly and DH_write. DH_readonly is used by the web interface to read the data from the DH. DH_write can also write data to the DH, but cannot change the database structure.

2.1.3. Read-and write-rights

Every unidb user can basically write data into the unidb hierarchy or delete data. The user can read all data from his assigned paths. It can also write to the assigned paths. However, he can only delete or edit documents that he has created himself. Excepted from this are users who have been configured as administrators. These can delete or edit all documents that are in the paths assigned to them. The most powerful user is therefore an administrator, to whom the path / has been assigned. The / is the root of the entire hierarchy.
Even if a unidb user has been configured as an administrator and therefore may have access to all points and tags in the DH, he cannot create, edit or delete tags or points in the DH because the unidb users in the MariaDB are configured so that they do not have write permission to the corresponding tables in the DH.

2.1.4. User management

New users can apply for a user account using the **Create user account** link in the unidb registration form.

Fig. 3: Registration form

The new user appears in the unidb administration area under the **User Management** link.

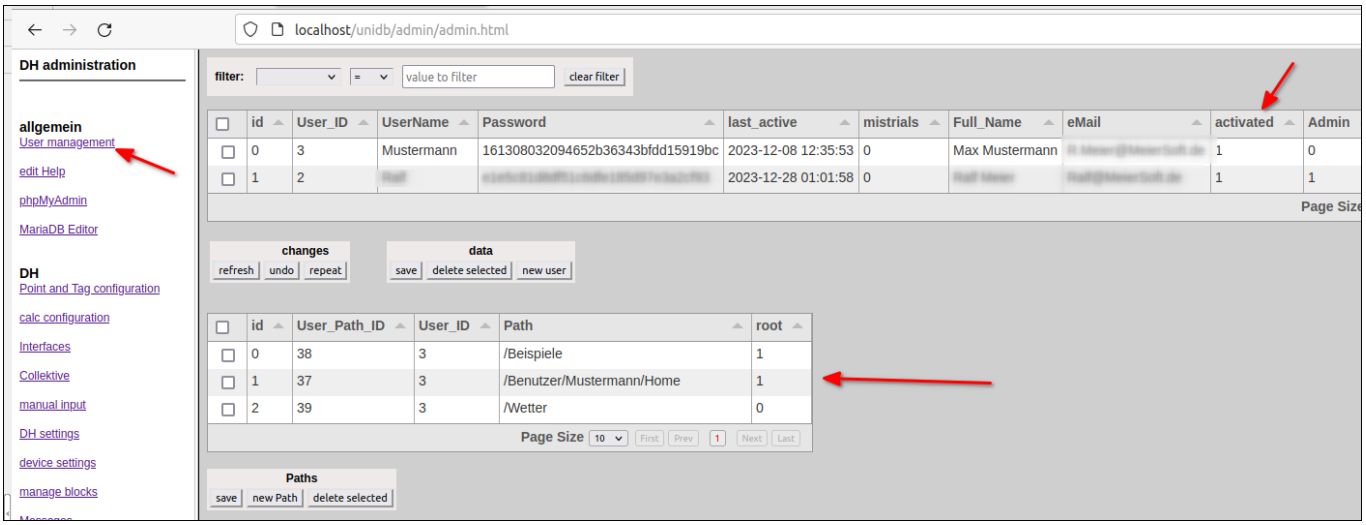


Fig. 4: New user in the user table.

At the same time, an email will be sent to the administrator with information about the new user.

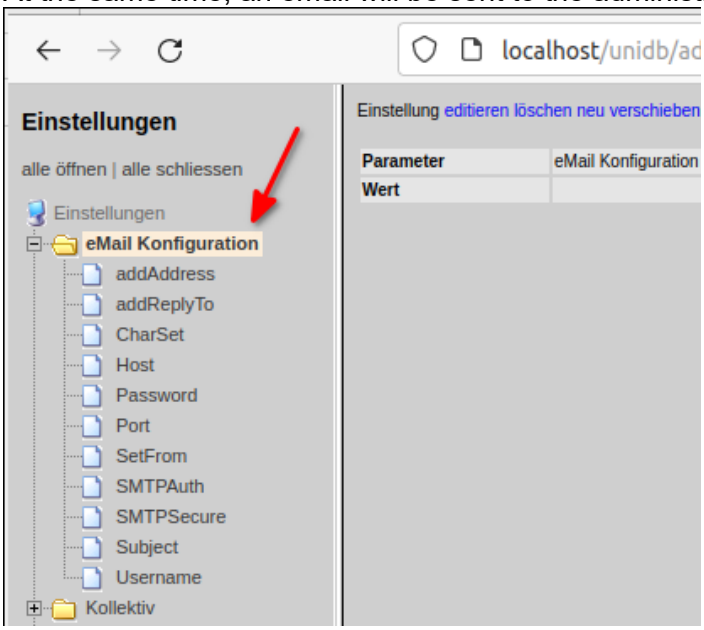


Fig. 5: Configuration of sending emails to the administrator

The admin's job is now to assign at least one path to the new user and then release the account. The release takes place in the user table in the activated column. If the value in this column is set to 1, then the user can log in. Please do not forget to inform the user about the activation.
In Figure 4, some paths have been assigned to the new user “John Doe”. The assignment of a user's usual paths can be automated. Which paths are assigned is determined in the settings under Templates for new users. Each entry under this point represents a path to be assigned automatically. If a path contains the name of the new user, this is represented by the variable $Benutzer. See Fig. 6.

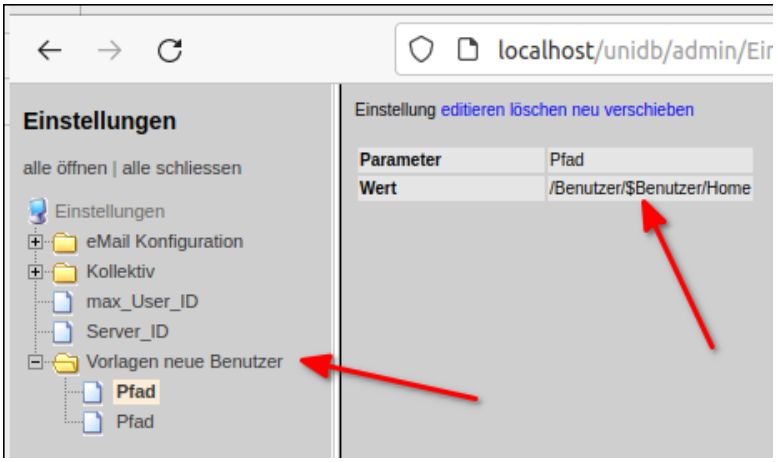


Fig. 6: Settings for automatically assigning paths for new users.

The user management form:
The form contains two tables. The top table shows the existing users and the bottom table lists the paths associated with the selected user from the top table.
The following applies to both tables: If you change the content of a field, the field is highlighted in color. The change is only written when you click on the “**save**” button below the table.
The first column id in the tables is only used for administration within the form. There is no corresponding field for them in the database.

Table Users:
The **mistrials** field contains the number of failed login attempts. If login has failed 5 times, then the user account will be locked by setting the **activated** field to 0. If the login is successful, the **mistrials** field is set to 0. The **Admin** field specifies whether the user has administration rights for their paths. The default setting for this is 0, i.e. no administration rights.

Table paths:
The meaning of the columns in this table should actually be clear. Only the last column, root, requires explanation. The name root can be taken literally here. For each line in which root = 1, a separate tree structure appears for the user.

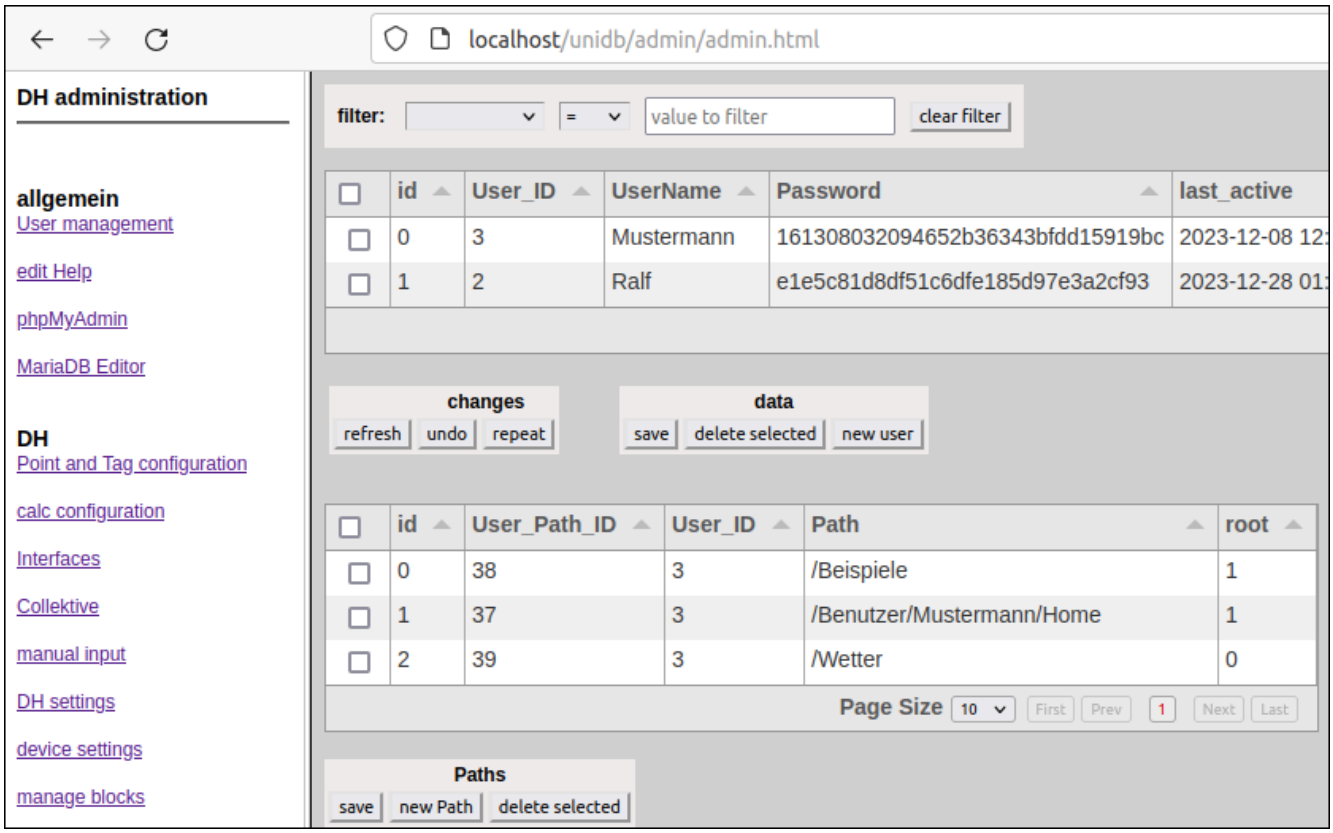


Fig. 7: User paths

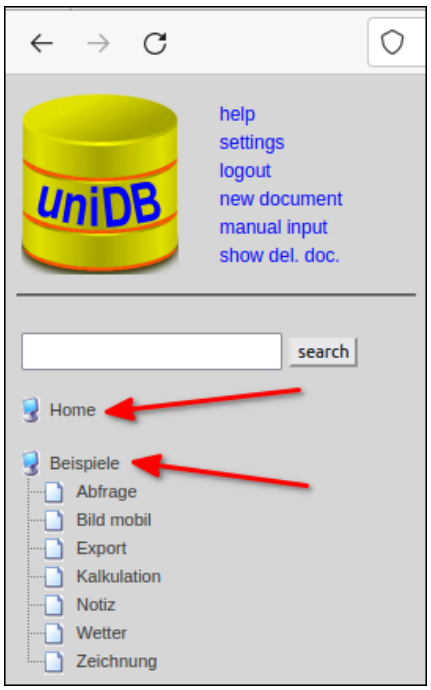


Fig. 8: root - user paths

The path/examples is marked root = 0 in Fig. 4 and therefore does not appear in the user's navigation area.

Attention:
If you assign a path to a user, it must actually exist in the hierarchy. Otherwise, this can lead to an entry without a parent ID and the tree structure will no longer be displayed correctly to the administrator. Paths that are automatically assigned when creating a new user account are created automatically if they do not already exist.

2.2. Settings

The Settings link opens a new tab in the browser in which all setting options are listed in a tree structure on the left. This form should only be used if there is no other way to change a setting. This is the case, for example, if you want to change the access data for emails to the administrator. Changing the automatically assigned paths for new users can only be done here.

Attention:
All changes are written to all servers in the collective. Change the value of Server_ID, then all servers will have the same Server_ID. This results in a corrupt tree structure.

2.3. Collective

The purpose and function of a collective has already been described in Chapter 1.1. This is about configuration.

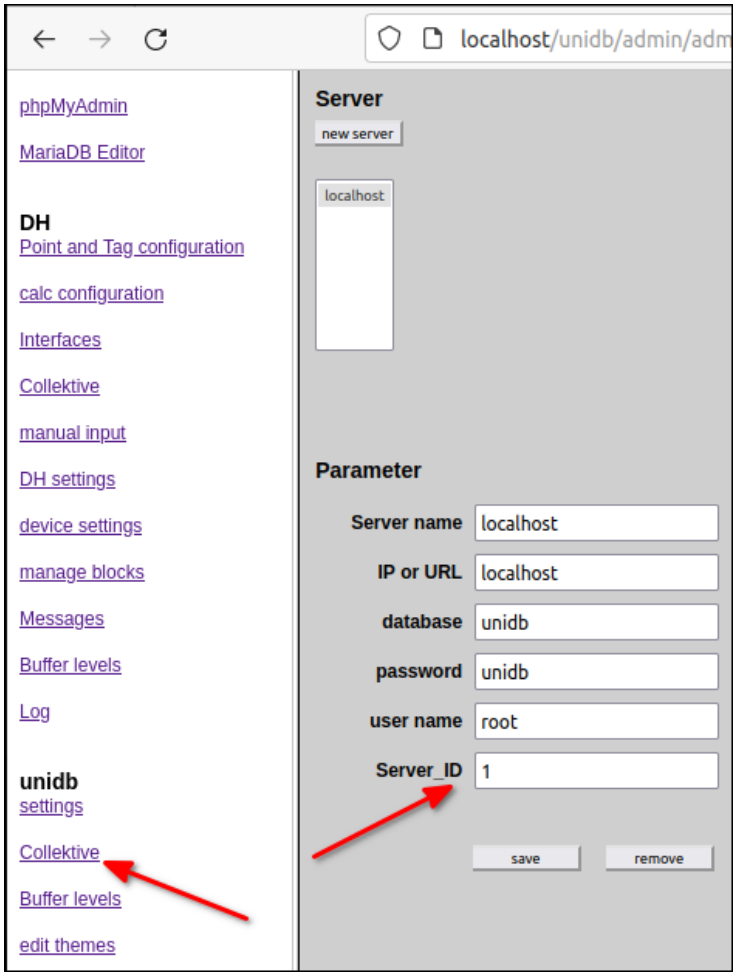


Fig. 9: Configuration of the collective for the unidb.

There's actually not much to explain about the form. Only the Server_ID field could be a bit confusing, as you can find an entry with the same name in the Settings form. Under Settings you can see the Server_ID of the server from which the settings were loaded. This is usually the server you are currently logged into. Please only set this value on this server using a tool such as phpMyAdmin. This is a field in the unidb Settings table.
The Server_ID that you see in the form in Figure 9 should be the same ID that you entered into the server's database using phpMyAdmin or similar. The entry here only serves to inform the collective members which server has which server_ID.

2.4. Bufferlevels

Ideally, all buffers are empty. If at some point a buffer has not been completely emptied, the Buffer Levels form will first show you how many entries are in the server buffers. Clicking on a server name opens a table that shows the entries in that server's buffer. Entries can be edited or deleted here.

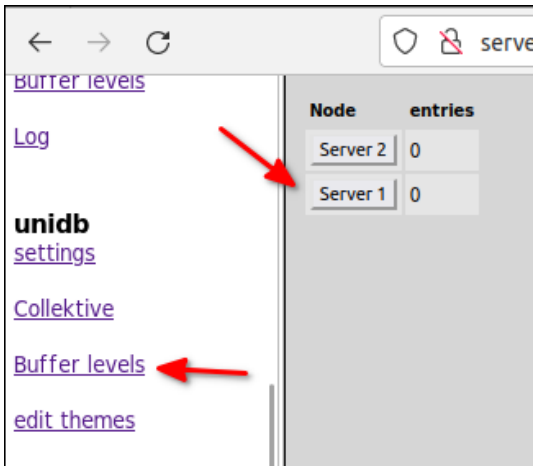


Fig. 10: Overview of the buffer levels.

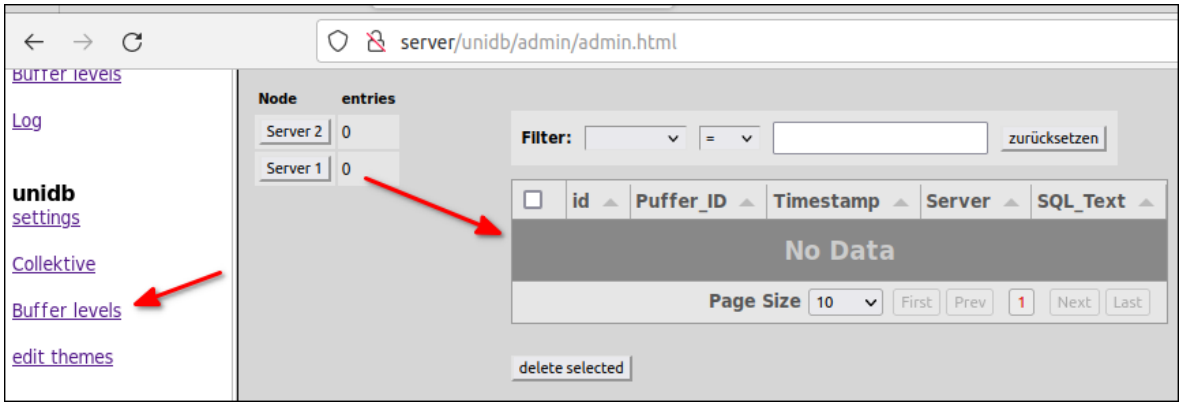


Fig. 11: Table of entries in the buffer of the clicked server.

Possible malfunctions:
If changes to documents are not transferred to other servers in the collective, this is probably because not all buffers are empty and therefore the input buffers of the collective members are not emptying.
Remedy:
Check whether any server still has entries in the buffer. If yes, then please check the contents of the SQL_Text column. If there is an incorrect SQL test here, please correct it or delete the relevant lines. The input buffers of the collective members should now empty within a short period of time. If this is not the case, please check on each collective member whether the scripts unidb_Abgleich.py and unidb_watchdog.py are running there. You do this in the terminal with ps -e|grep unidb. If one of the scripts mentioned above is not listed here, then start the missing script in the terminal. The scripts should be in the /opt/unidb directory. So in the terminal write /opt/unidb/unidb_Scriptname.py &. The & character is used to run the script in the background.

A notice:
It is not only convenient but also safer if you start the scripts automatically via an entry in the /etc/init.d directory. Otherwise you will forget it when you start the computer.

2.5. edit Themes

Each user can select a theme for the user interface design in their personal settings.
Here you can change the design of the topics or create a new topic as a copy of the currently selected topic.

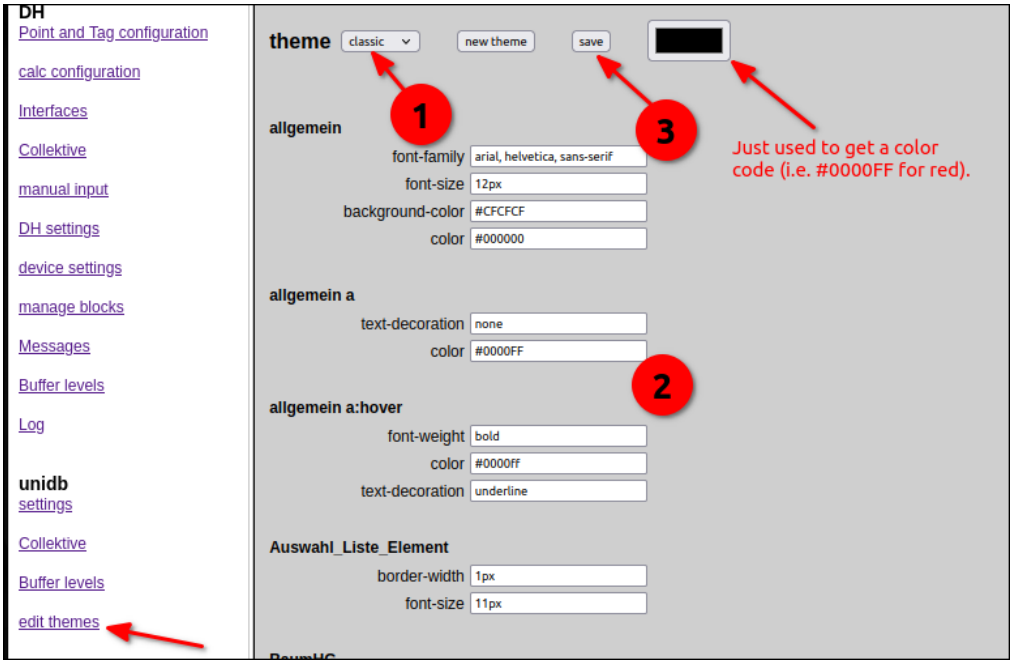


Fig. 12: Edit themes.

Select a topic from the combo box (1). Change what you want to change (2) and finally save the changes (3). The black button at the top right is only used to select the desired color for a field that expects a color code (e.g. color or background-color).

2.6. MariaDB Editor

The documents in the unidb are data records from the tree table. Each record contains a field called Content. This field contains other fields that are treated like fields in a NOSQL database. The only difference between the document types is the dynamic fields within the Content field. Most SQL editors can edit the “normal” database fields without any problems. These fields are therefore only displayed in this form. The editor is built to make the dynamic fields selectable so that they can be edited.
If a unidb document is edited, the previous version of the document is stored in the tree history table. These documents can also be edited using the form.

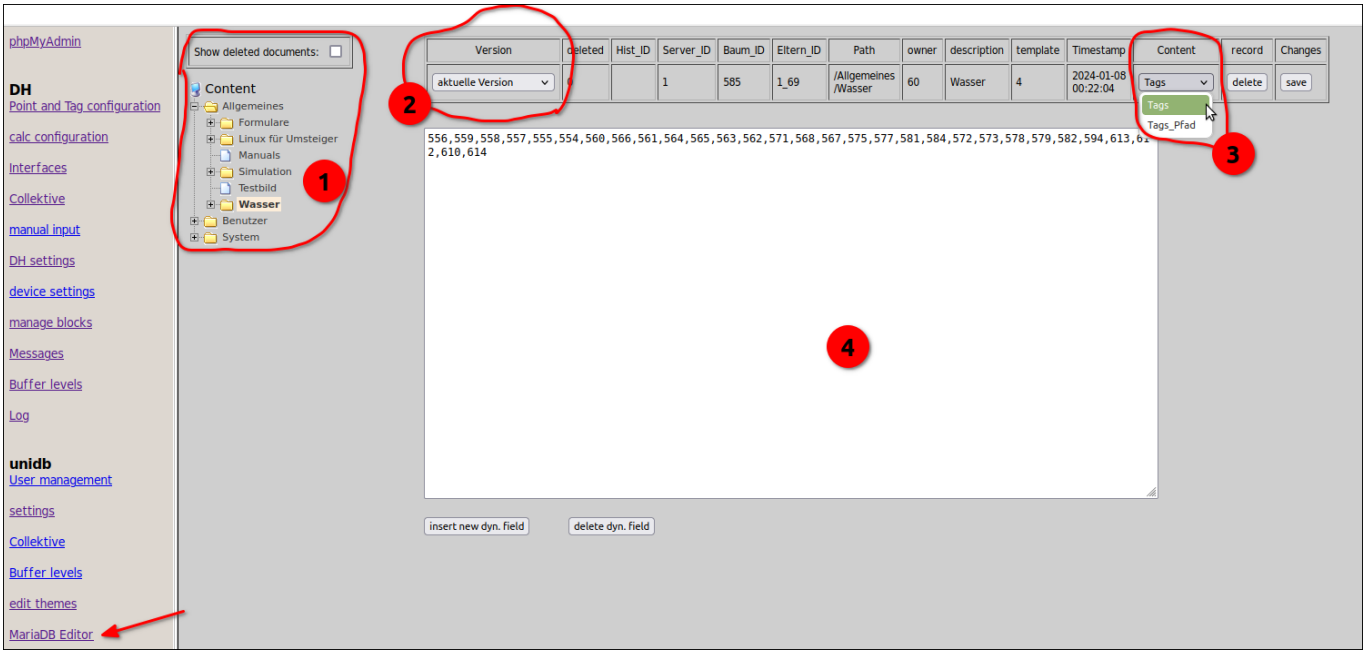


Fig. 13: MariaDB Editor form

Area 1 in Fig. 13 shows the tree structure of the documents, as you know it from the normal user interface. Area 2 contains a list field from which you can select either the current version of the document from the Tree table or an archived version from the Tree History table. Usually you want to edit the current version of a document. Therefore this is also preselected. In area 3 there is a list field that makes all dynamic fields of the document selectable. In Fig. 13 the document Water has been selected. This is a group type document. This document type contains only two dynamic fields. The Tags field contains all Tag_IDs of the tags from the group, each separated by a comma. The Tags_Path field contains the path to the tags. For more information about the Group document type, please refer to the Developer Guide. You can now make your changes in area 4. Please don't forget to save the changes using the button at the top right of the form.

A notice:
Just like making changes in the “normal” way via the Edit link in the document in the user view, the previous version of the document is also written to the archive and a new version is created.

Attention:
But if you delete a document here, it is really gone, whereas when you delete it in the “normal” user view, only a deletion marker is set.

3. DH

The DataHistorian consists of three components. These are the interface scripts and various help programs in the /opt/DH directory, the archive files in the /var/lib/DH directory and the DH database (MariaDB).
The interfaces read the data from their sources and store it in the database table akt. The compressor (/opt/DH/DH_comp.py) compresses the data and writes it to the /var/lib/DH directory. The data is finally visualised by the unidb user interface.

3.1. Point- and Tagconfiguration

Points are the objects to which the data is assigned. Tags are simply links to these objects. It often happens that users need tags that have different names or paths but are based on the same point. These are usually organisational reasons. With DataHistorians that only have points, the admin has no choice but to create a new point if a user needs the data from a point under a different name. This then has two disadvantages. The biggest disadvantage is that the newly created point has no history because it first has to collect data. The second disadvantage is that this way you produce data garbage because you collect the same data twice from the time the new point is created. If you have the option to simply create a new day instead, it has the complete history and does not collect duplicate data.
If you create a new point in the DH, a tag with the same name is automatically created, which also has the same path as the point. Users only ever see the tags, never the associated point.

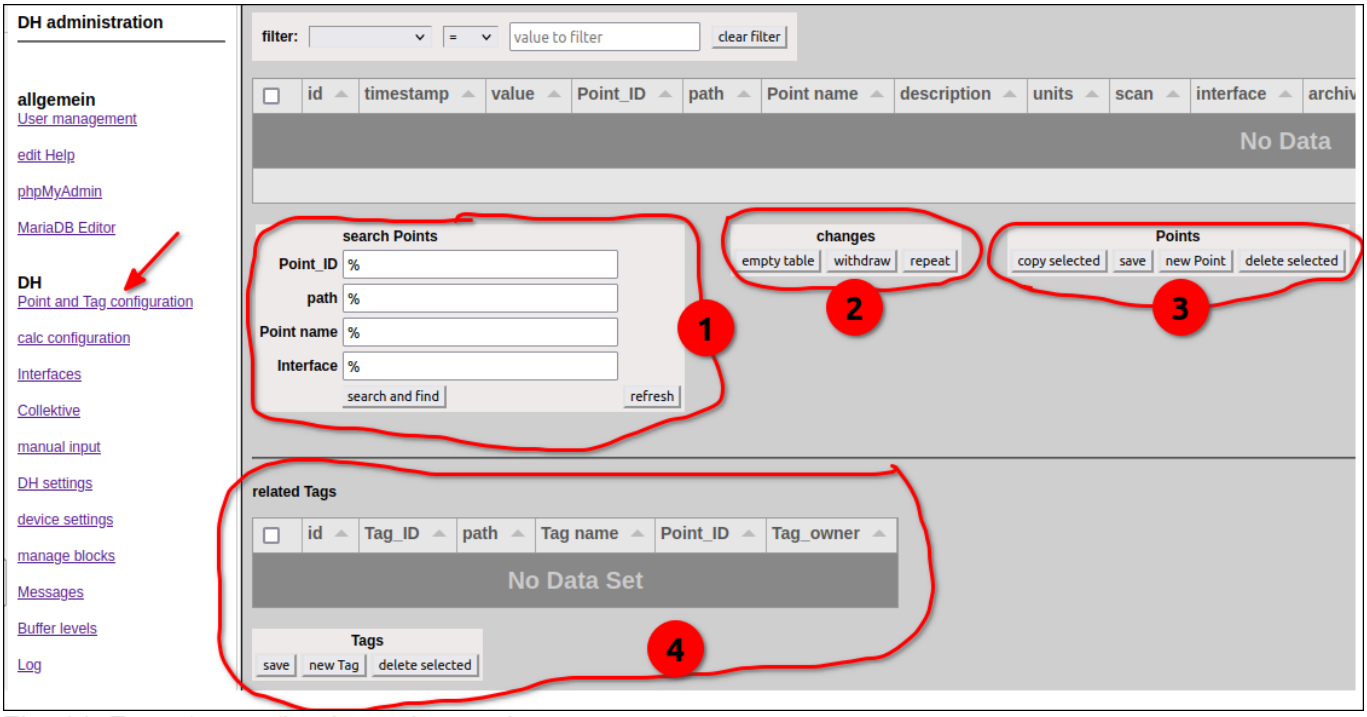


Fig. 14: Form for configuring points and tags.

If you call up the form for the point and tag configuration, it initially contains two empty tables. In the Search Points area (1) you now have the option to search specifically for one or more points. All criteria offered are OR-linked during the search. The % character serves as a place holder for any number of characters. After clicking on the “search and find” button, the points found appear in a pop-up window. Any number of points can be marked there and added to the form.

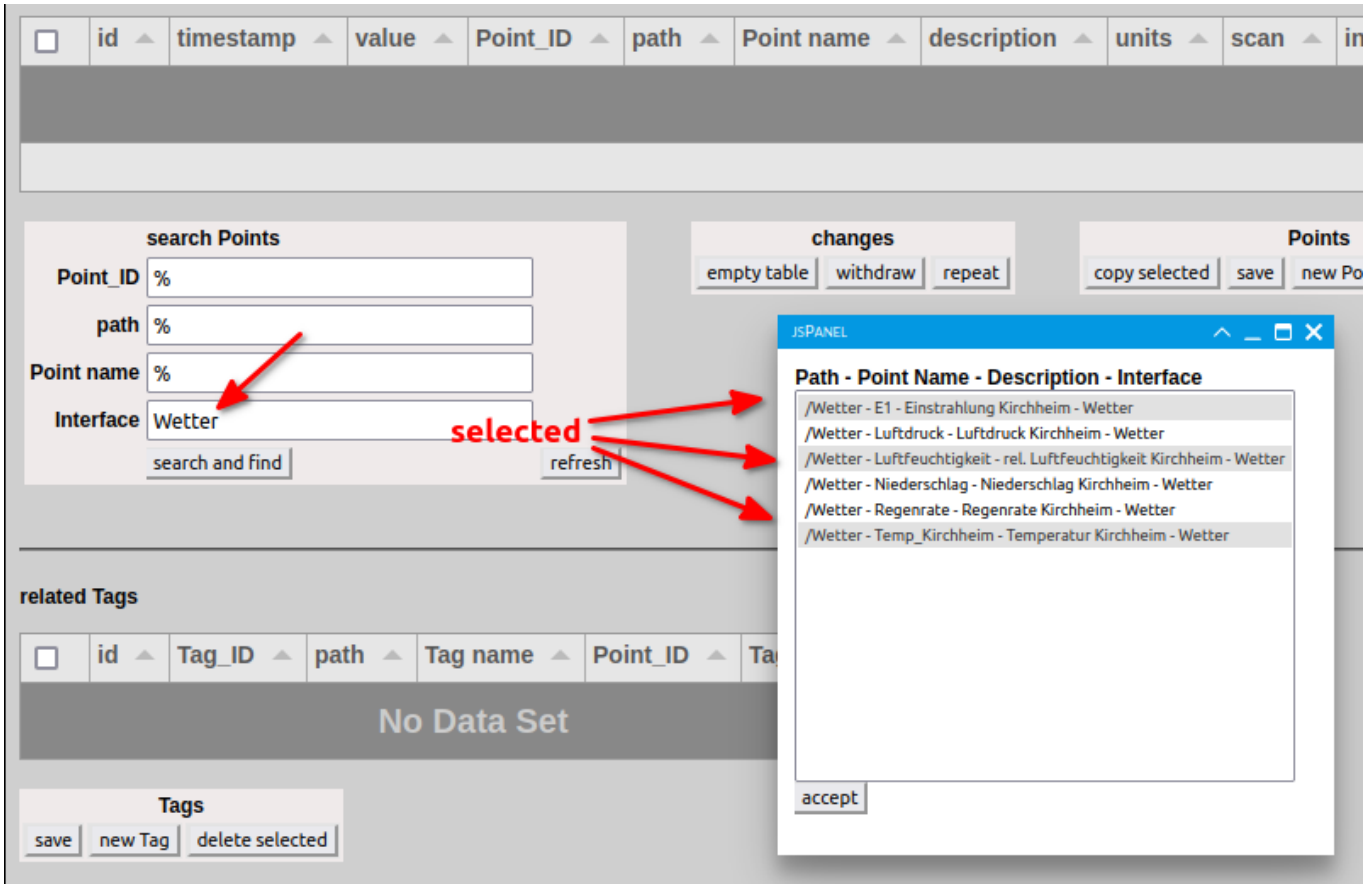


Fig. 15: Selection of the points to be edited.

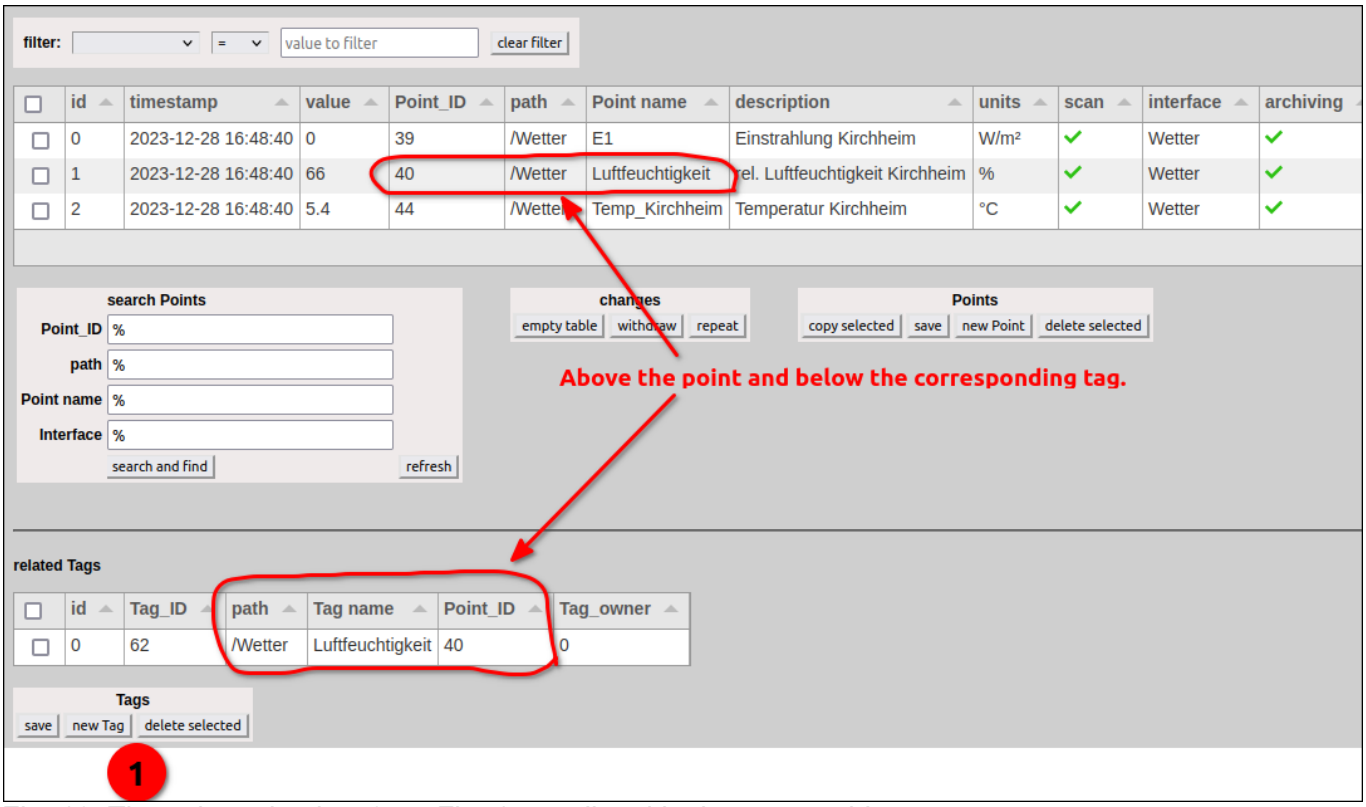


Fig. 16: The selected points from Fig. 15 are listed in the upper table.

The second line is highlighted in the table above. A tag associated with the highlighted point is displayed in the table below. If you want to create another day, press the “**new tag**” button (1).

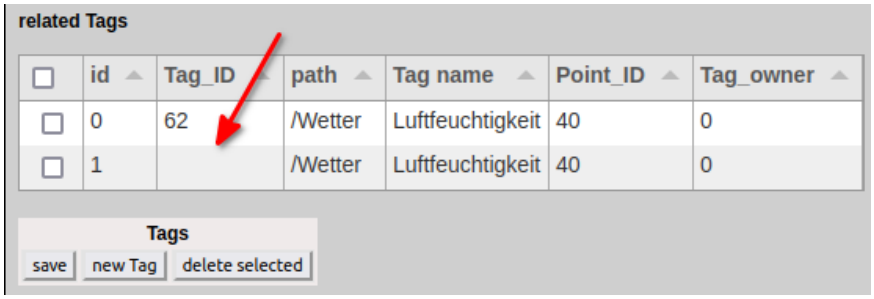


Fig. 17: New line for a new Tag.

A new line appears in the lower table in which almost all fields are prefilled. Only the field for the Tag_ID is empty. This has to be the case, as the day is only created once you have clicked on “save”. The Tag_ID is assigned automatically. However, at least one of the path or tag name fields must be edited beforehand, as the new tag must have a unique combination of path and tag name. If this is not the case, the day will not be created.

Configuration of the points:

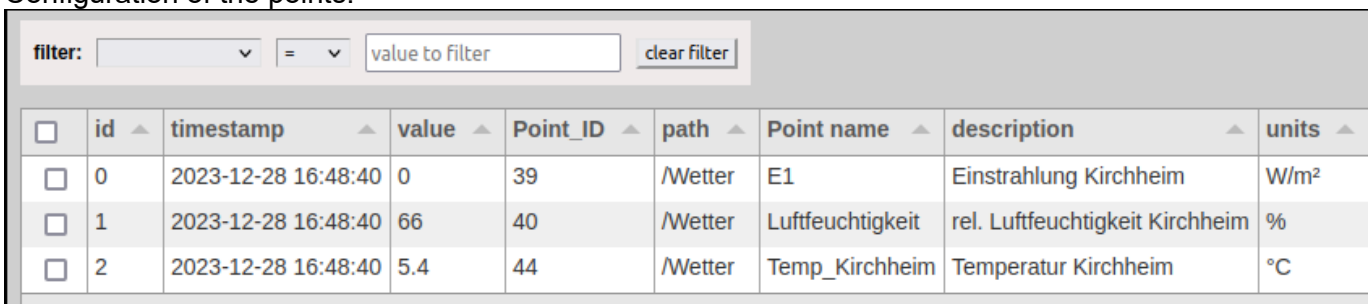


Fig. 18: Point configuration part 1

The first two columns in Figure 17 are used to manage the table and have nothing to do with the actual configuration. The timestamp and Value columns display the latest value for the point from the row. This is useful if you have reconfigured or created a new point to find out whether the point actually works. The Point_ID is automatically assigned for a new point. It cannot be edited here either. The next two parameters, path and pointname, must be unique in their combination. The Description parameter certainly requires no further explanation. You can also edit the unit, but you should exercise caution here. If you change the unit, it may be that the archived values no longer fit. Example: A measurement previously delivered the values in dm³/h and now delivers in m³/h. If you only change the unit in the point configuration, the already archived values will be displayed too high by a factor of 1000. Only a new point helps here. However, if it is a differential temperature that was previously displayed in °C, then the unit can be changed to the usual unit K (Kelvin) without hesitation, as this does not change the value. Differential temperatures are usually given in K.

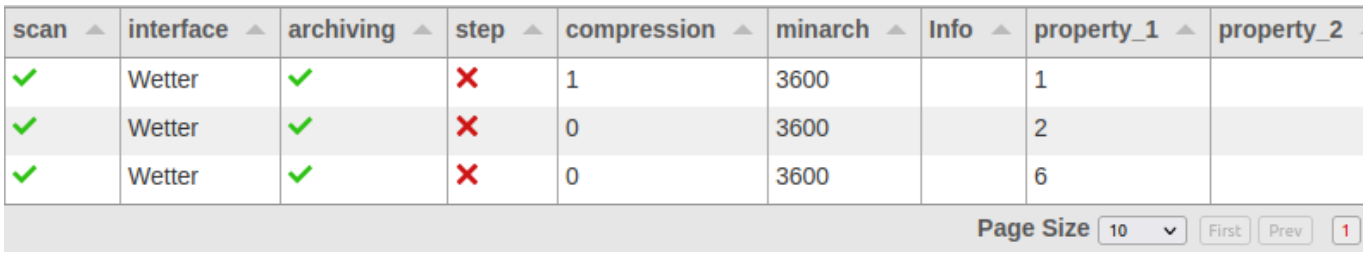


Fig. 19: Point configuration part 2

A tick in the scan column ensures that the point is recognised by the interface as active and that data should be collected for it. The interfaces use the interface parameter to recognise which points belong to them. The tick in the Archiving column tells the compressor that it should write the values for this point to the archive. It happens that the values of a point are only needed to calculate other values, but are otherwise uninteresting. In this case you don't want to archive them and put an X in this column. The step parameter specifies whether a value should be interpolated for a point in time if there is no value in the archive at exactly that point in time. When trending the values of a point that is configured with step = 1 (tick), they appear in the form of a staircase. If there is an X in this field, the values will be interpolated. The compressor only writes a value to the archive if it differs from the previous value by the amount that you enter here under compression. This prevents unnecessary numbers of values from being stored in

the archive, which is good for performance. This also removes the noise. If a value that is too high is entered here, then the archived values no longer provide a usable image of the actual time course of the value. In the worst case, the data is unusable. Example: The outside temperature changes only slowly. Without compression you will probably get around 800 values in 24 hours. Most values differ from the previous value by less than 0.1 °C. If you ask yourself whether it is really necessary to archive these values in such high resolution, then the answer is probably NO. If you enter 0.09 in the compression field for such a point, then only values that differ from the previous value by at least 0.1 °C will be written to the archive. The number of values for a period of 24 hours is reduced to approx. 200 - 300. If you look at a trend over a period of a year, then this is very noticeable in the performance. Trends over a shorter period of time are easier to view when the noise is gone. If 0 is specified for compression, then only unchanged values are excluded from archiving.
The minarch parameter specifies the period in seconds after which a value is definitely written to the archive, even if it has not changed. This makes later calculations easier and has a positive effect on the trend display.
The Info field contains additional information for the interface that it needs to read the values. This can be, for example, the ID of a 1-wire sensor or the specification of a URL. Whether something is in this field and if so, what depends on the interface. This field accepts text type data. The following five parameters behave in the same way as the Info parameter. The parameter Property_1 expects an integer value and the remaining property fields each expect an alphanumeric value.

Point type	decimal places	scale min	scale max	intervall	mean values	last change	first value	Point_owner
double	0	0	1000	300	✓	2023-04-09 23:16:28	2023-04-09 23:16:28	0
double	0	20	100	300	✓	2023-04-09 23:16:28	2023-04-09 23:16:28	0
double	1	10	20	300	✓	2023-04-09 23:16:28	2023-04-09 23:16:28	0

Page Size 10 First Prev 1 Next Last

Fig. 20: Point configuration part 3

Point type is always double. The parameter dates back to ancient times and is actually no longer needed. Decimal places indicates how many decimal places a value is displayed to users. The Scale_min and Scale_max fields contain specifications for scaling trends. Each user can set their own scaling limits for each point. Only if a user has not set any limits for the point will these two fields be used when creating a trend.
The Interval parameter can be used by the interfaces to use an individual polling interval for each point. Very few interfaces make use of this.
In the Average values field you can specify whether hourly and daily average values, as well as the min and max values for each hour and each day, should be archived for this point. Whether you put a tick or an X here depends on the point itself. If counter readings are collected on this point, then it certainly makes sense to put an X here. However, if temperatures or flow rates are written to this point, then it certainly makes sense to calculate and archive the mean, minimum and maximum values.
The next two fields are for informational purposes and cannot be edited. The “first value” field is also important for querying archive values. Trends can be created more quickly if the calculator knows not to expect a value before the specified time. This saves him the trouble of searching through the entire archive.
The last field Point_owner is not yet used. But that can change at some point.

A few practical experiences:

1. If you want to create a new point, you can save yourself work by first looking for a similar, existing point and simply copying it. To do this, mark it in the first column and press the **copy marked** button below the table in the **Points** area. A new line with the copied values is entered into the table. The Point_ID field is empty, which indicates a new, unsaved point. Now you probably just need to adjust the Path, Pointname and Description parameters.
2. Sometimes you spend what feels like an eternity searching for the reason why a point doesn't deliver any values. What has proven to be very helpful here is to load the non-functioning point into the table along with at least one other, similar point and then compare the points column by column. Ideally, the comparison point should belong to the same interface as the problem point.

Hints:

1. In the Find Points area there is a switch at the bottom right that says **refresh**. Admittedly, this switch is not conveniently placed. It is used to read in the latest values for the points in the upper table again. Once you have reconfigured a point, you like to make frequent use of this switch to determine

whether fresh values are arriving for the point in question or not.

2. When a changed configuration of a point is saved, a message is automatically sent to the associated interface so that it rereads the configuration of its points. This usually happens within a few seconds.

3.2. calc Configuration

The values of the points assigned to the calc interface are calculated at regular intervals. The expression to be calculated is in the Info field in the configuration of these points. The interval for the calculation is written in the Interval field.

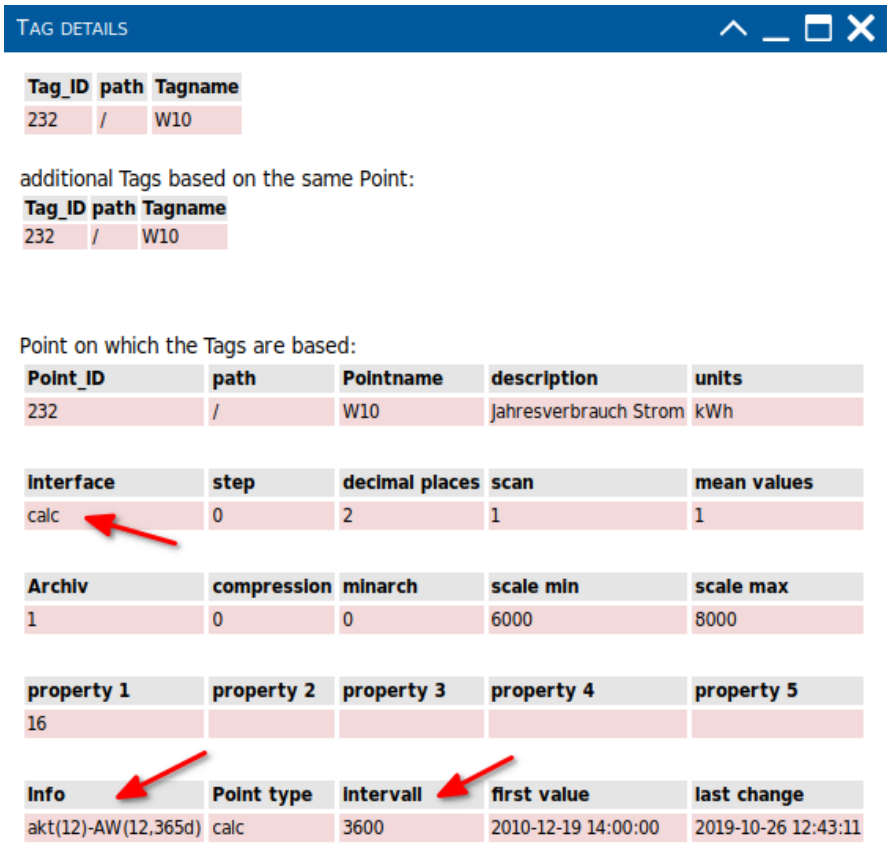


Fig. 21: Configuration of a point of the calc interface.

You can therefore create a point to be calculated just like any other point. In practice, however, it often happens that a calculation does not work or does not produce the expected result. In such cases, the admin can easily help themselves using the **calc configuration** form. In this form you can select the point to be calculated and test the calculation. A few more configuration parameters for the point are also displayed, which can also be edited here.
In images, some elements can also be configured with calculations instead of tags. The syntax is the same as for calculated points.

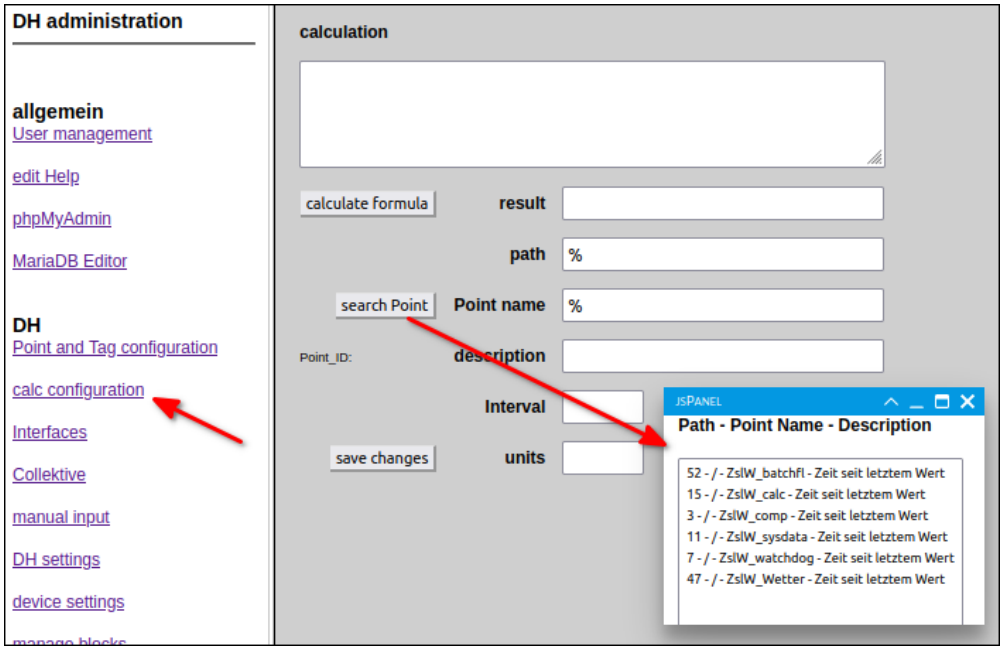


Fig. 22: **Calc configuration** form with selection dialog for calculated points.

The Search Point switch in Figure 22 only searches for points that are assigned to the calc interface. The selection in the dialog can be further restricted using the Path and Pointname fields.
If a point has been selected, the fields are filled in with its data. The top field contains the content of the Info parameter, i.e. the expression to be calculated.

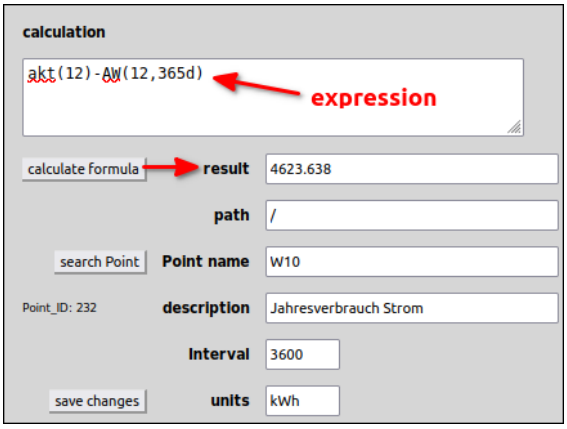


Fig. 23: Parameters of point W10 with the result of the calculation.

The **Calculate Formula** button writes the result of the calculation into the Result field. If the expression is incorrect, the field remains empty.
If the result is plausible, then nothing further needs to be done. If the field remains empty or just contains nonsense, the expression in the form can be edited and tested again by clicking on the **calculate formula** button. The changed configuration of the point is only applied when you click on the **save changes** button.

To test an expression that is used in an element of an image, you can copy it into the top field and then also have it calculated using the **calculate formula** button.

A notice:
The calc interface is a Python script. You can therefore use all the functions that Python has implemented.
An expression that calculates data for an element in an image is calculated by the web server. This uses PHP for calculation. The differences in the functions of Python and PHP are not great, but it can happen that an expression that is evaluated with Python cannot be evaluated in the calc configuration form. In such a case, please use Python notation for testing. The function to get the current UNIX timestamp is written in Python time.time(). In PHP, however, time() is sufficient. Since this function is used frequently, the PHP script contains an adjustment that translates time.time() to time().

Special DH functions for the calc interface:
Functions that require one or more times as arguments accept them in two different ways:
1. Absolute time in the format YYYY-MM-DD hh:mm:ss (2023-12-15 17:45:13).
2. Relative time as a combination of a number followed by a letter. The letter indicates whether it is days (d), hours (h), minutes (m) or seconds (s). The number in front of it is the multiplier. Relative time always refers to the current time and is in the past.

Examples:
- One day in the past: 1d. If the current time = 2023-12-1**5** 17:35:12, then 1d stands for 2023-12-1**4** 17:35:12.
- Three hours in the past: 3h

- **act(Point_ID):**
  This function returns the latest value for the specified point.
  Example: akt(4711) returns the latest value for the point with the Point_ID 4711.
- **AW(Point_ID,Time):**
  Reads the value of the specified point from the archive at the desired time. If there is no value in the archive for the specified time, then the value of the entry immediately before it is returned.
  Example:
  The following values were archived for Point 4711:
  ...
  2023-12-15 17:10:12, 17.4, ...
  2023-12-15 17:15:26, 18.3, ...
  2023-12-15 17:18:02, 18.9, ...
  ...

AW(4711,2023-12-15 17:14:28) returns the value 17.4 because no value was archived for the desired time and the last value before that was 17.4 from 5:10:12 p.m.

- **intp(Point_ID,Time):**
  Returns the interpolated value for the point, which was calculated from the archived values for the specified time. It would be pure coincidence if a value for the point was archived for exactly the desired time. Therefore, whenever there is no value for the point with the exact matching timestamp in the archive, the value at that point in time is calculated from the value immediately before and the value immediately following.
  Example: AW(4711,2023-12-15 17:48:19) returns the interpolated value of point 4711, which it approximately had at the specified time. Please make sure that you do not put the date in quotation marks.

- **MW(Point_ID,Time_from,Time_to):**
  Calculates the average of the point in question for the specified period. It is important that the calculation also takes into account the time intervals between the values.

- **ZS(Point_ID):**
  Returns the timestamp of the latest value for the specified Point.

- **ZP(Point_ID):**
  Returns the Unix timestamp of the latest value for the specified Point. The Unix timestamp indicates the number of seconds that have passed since January 1, 1970 00:00:00.

3.3. Interfaces

I think the word has gotten around now that the interfaces collect data from somewhere and write it to the *akt* table of every server in the DH collective.
The only exception is the *comp* interface. This interface reads the data from the table and writes it compressed into the archive. This interface must run on every server. The *watchdog* interface is also a special case. It looks on the computer on which it is running to see which interfaces should be running there and which are actually running. It also reads the messages from the server and passes them on to the other interfaces. If the message requires it, it also starts an interface. The *watchdog* interface runs on every computer in the collective.

3.3.1. Program Sequence

Figure 24 shows the program flow that each interface follows. After the start, initialisation (1) follows, during which the points for the interface are loaded. All points are read in whose Interface parameter is the same as the name of the *interface* and which are configured with *scan* = *1*. The interface settings are also read in. This always includes the *Interval* parameter. The interval determines the time interval at which the data is read in. An interface can optionally load additional parameters. The settings are made via the **interfaces** form in the admin area.

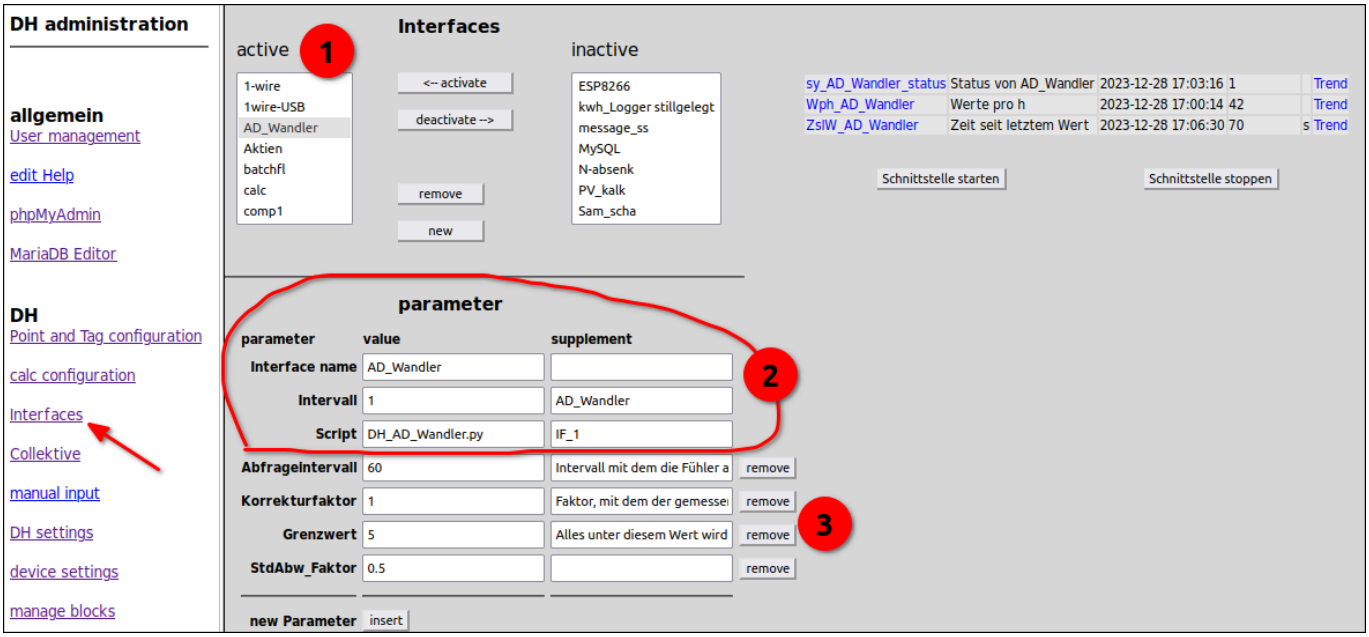


Fig. 24: Form for configuring the interfaces.

In the form in Fig. 24, the parameters of the interface that was selected under (1) are displayed under (2) and (3). In the example above this is “AD_Converter”. The parameters in the marked area (2) are required by every interface. The last four lines (3) are specific settings for the selected interface.
The interface name can be freely assigned (line 1, area (2)). The interval must always be specified. Most often the value is 60, 300 or 600 seconds. In the Additional field, the name of the script must appear without the DH_ part at the beginning and without the file extension (line 2, area (2)). That's how it happened at some point. Please don't try to understand it. I do not understand it either ;-). In the last line of the area (2), the full file name of the script is expected as the value and the name of the computer on which the script is running is expected as an addition. In the example above, this is the interface computer IF_1. As a reminder: The names of the collective's servers and interface computers are assigned in the **collective** form.

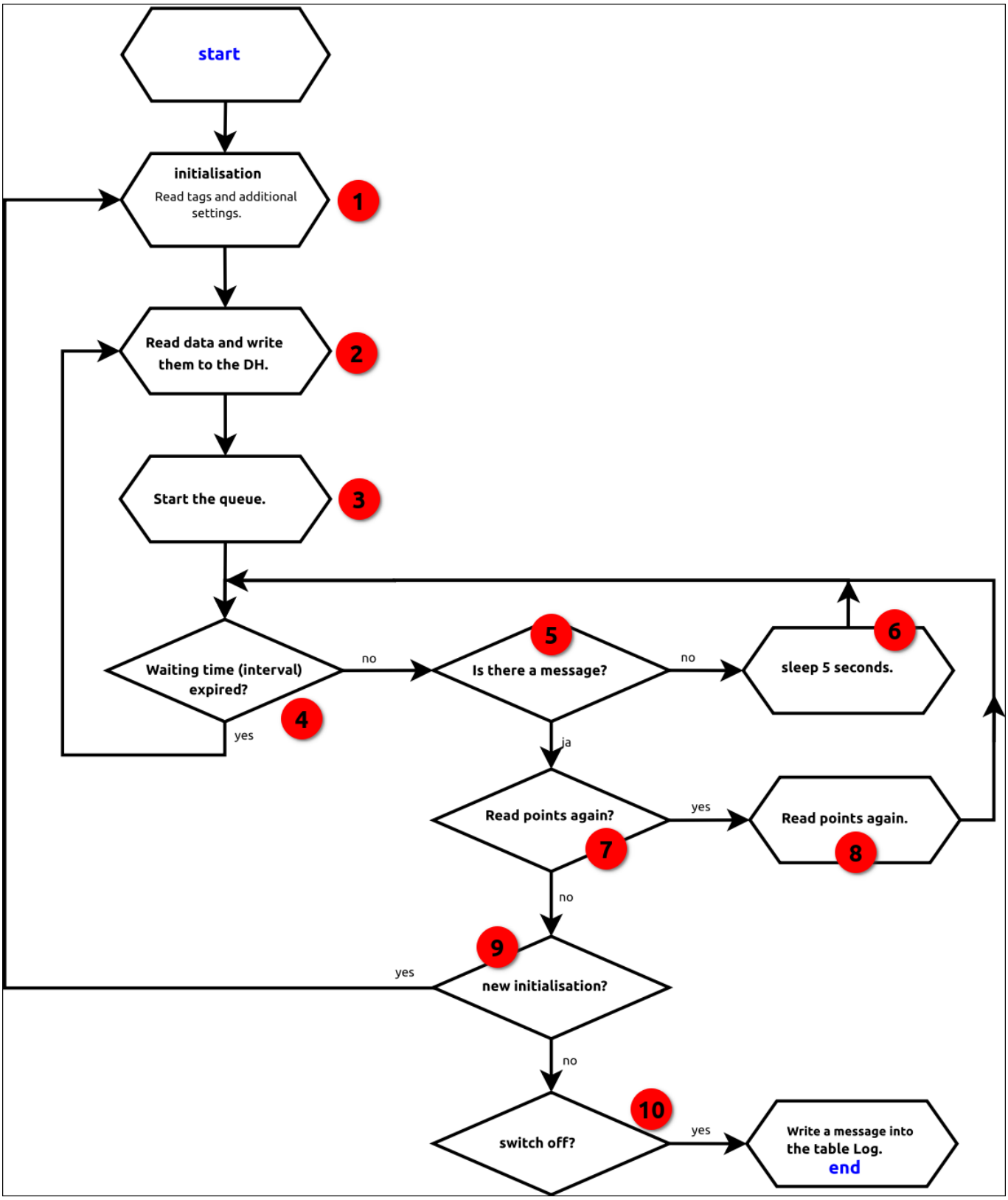


Fig. 25: Program flowchart interfaces

We are back to Fig. 25. After the initialisation (1) is over, the loop of reading data and writing to the DH (2), pause (3) to (9), reading and writing again, ... begins.
After point (2) has been processed, the interface waits for a maximum of the time configured as an interval in seconds. During this time she checks every 5 seconds to see if there is any message for her. Messages are stored in the temporary directory by the watchdog interface. The */opt/DH/tmp* directory is usually used for this. However, you can also specify a different directory in the admin area under *Settings / Server components / temp directory*. Messages to which the interfaces react are „neu initialisieren", „Points einlesen" and „abschalten". The *watchdog* reads the messages from the database and creates a file for each message whose name begins with DH_ce_. This is followed by the interface name and finally the current Unix time stamp.

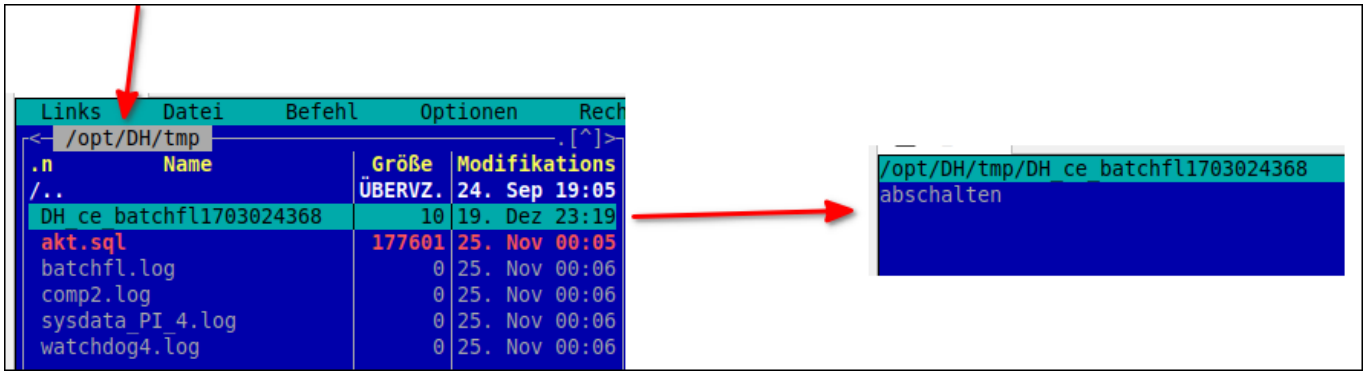


Fig. 26: Message to the batchfl interface. Message: abschalten (switch off)

As soon as an interface is in the waiting loop, it checks whether one of the three messages (7), (9), (10) is present and acts accordingly. If there is no message, she goes to sleep for 5 seconds (6).
If the pause time (interval) has expired, the cycle begins again with the program jumping back to point (2).

3.3.2. Start and stop interfaces

There are three ways to start or stop interfaces.

1. Directly from the console:
   Open a console (terminal) and start the interface like any other program. To avoid having to keep the console open, you should append a **&** character to the command to keep the program running in the background. The interface scripts are usually located in the */opt/DH* folder. Your file name always begins with DH_. There is a good reason for this, as you will find out below. Suppose the script to be started is called Wasser. The command is: */opt/DH/DH_Wasser.py &*.
   You can only stop it via the console by killing the process. The command *ps -e|grep DH_* lists all scripts running on the computer. The PID is next to the name of the script. The command *kill 4711* now ends the script with the PID 4711. However, this is the brutal method that you should actually avoid. If you do this at the wrong moment, it can lead to a broken table in the database.
2. In the */opt/DH* folder there are two scripts to start and stop all interfaces on the computer. The **DH_start.py** script starts every interface that is in the list **active** in the Interfaces form (admin area). The script **DH_stop.py** simply sends the shutdown message to every running interface. After you have started the **DH_stop.py** script, the *ps -e|grep DH_* command should no longer list any running interfaces after 15 seconds at the latest.
   If you want to start the interfaces automatically when you start the computer, simply create a service that runs the start script. You can do the same when shutting down. This ensures that all interfaces are terminated cleanly.
3. You can start and stop an interface using the Start interface button or Stop interface button in the Interfaces form (admin area). See point (4) and point (5) in Figure 26.

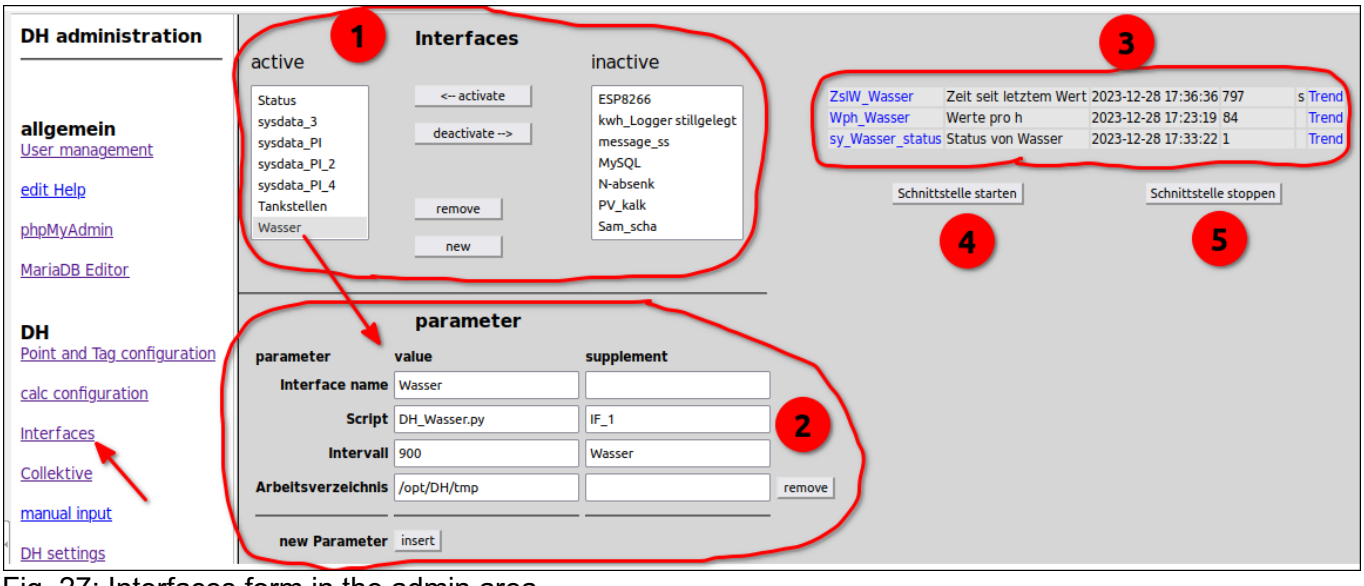


Fig. 27: Interfaces form in the admin area

3.3.3. Configuration

The configuration is made in the **interfaces** form in the admin area.
In area (1) you will see two lists active & inactive. The interfaces in the active list are started by the DH_start.py script and monitored by the watchdog interface. Use the two switches **<--- activate** and **deactivate --->** to move an interface into the other list. If an interface is deactivated, the three adjacent tags (3) are also set to scan = 0. When activated, the tags are reconfigured to scan = 1 in the background. You should careful use the **remove** switch, as it deletes the entire configuration of the selected interface. It doesn't matter which of the two lists the interface is in.

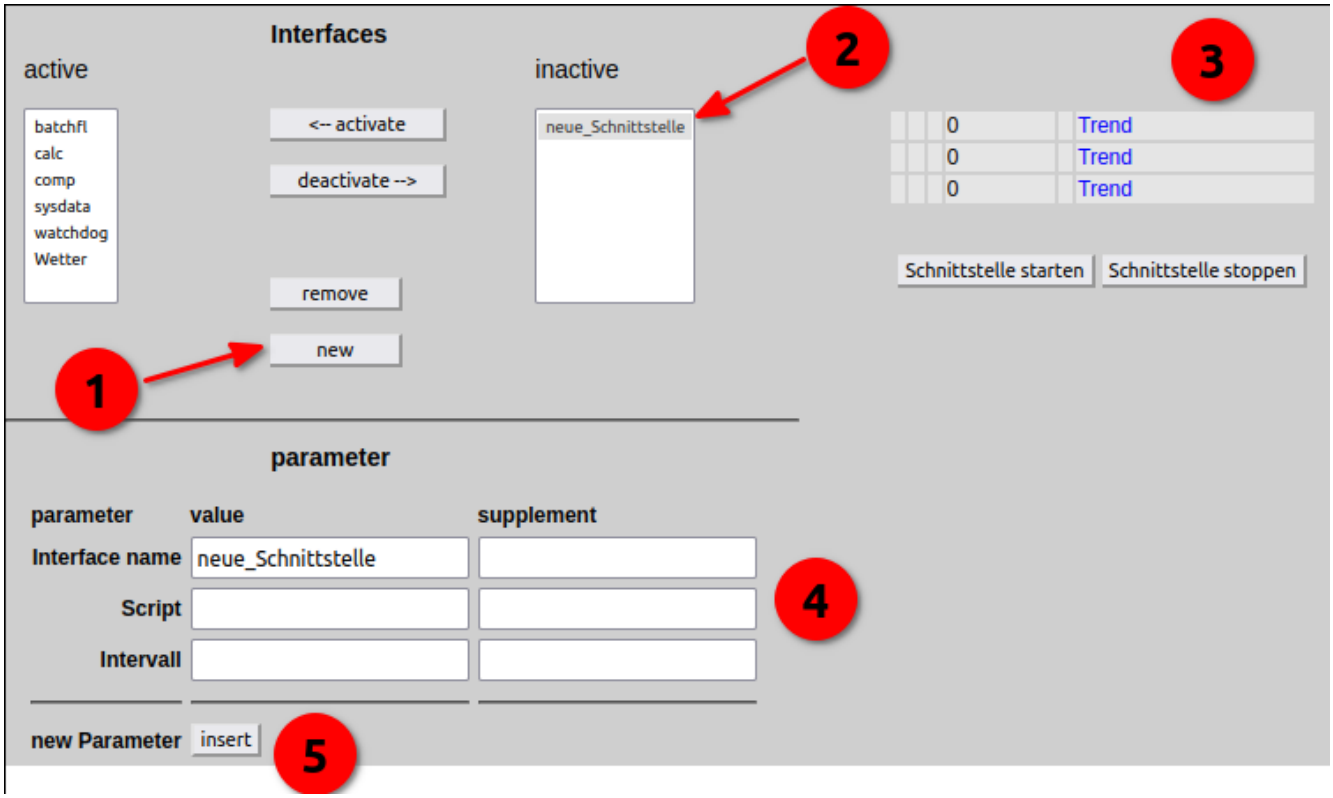


Fig. 28: new interface

A new interface is created using the **new** switch (1). You can find the new interface in the **inactive** list (2). In the next step, the three standard parameters *interface name*, *script* and *interval* must be configured. What belongs there has already been explained in Fig. 24. If you need additional parameters for the interface, simply insert them using the **insert** button (5).
Each interface has four tags that provide information about the operation of the interface.
The most convenient way to create these points is to search for the points for an existing interface in the Point and Tag Configuration form.

Pfad	Pointname	Beschreibung	Einheit	scan	Schnittstelle	Archivierung	step	Kompression	minarch	Info
/	sy_1-wire_status	Status von 1-wire		✓	watchdog1	✓	✓	0	3600	IF_1
/	Wph_1-wire	Werte pro h		✓	System	✓	✗	0	28800	1-wire: Werte_pro_h_Tag
/	ZslW_1-wire	Zeit seit letztem Wert	s	✓	calc	✓	✓	0	0	time.time()-akt(1999)
/	ZlW_1-wire	Zeitpunkt des letzten Wertes	s	✓	System	✗	✓	0	0	1-wire: Zeitpunkt_letzter_Wert

Fig. 29: Points for interface *1-wire*.

Example:
In Fig. 29 you can see the configuration of the points for the *1-wire* interface.
And this is how you do it:

1. Enter the search text %*Interface name*% in the Point name field. Four points should be found.
2. Now mark all the lines by putting a cross in the first column. These points can be copied using the **copy marked** button below the table.
3. In the Pointname column, replace the name of the interface. If you want the new interface to be called ADWandler, change the point name *sy_1-wire_status* to *sy_ADWandler_status*. Do the same with the other points.

4. Adjust the **Info** parameter in each case. Replace the name here.
5. Press the **save** switch below the table.
6. The **Info** parameter of the *ZslW_ADWandler* point still needs to be adjusted. Here replace the Point_ID in the expression *time.time()-akt(1999)* with the Point_ID of the new point *ZlW_ADWandler*.
7. Click **save** again.

If you now return to the Interfaces form and click on the new interface, you should see three of the four new points in the area (3, Fig. 28).
The configuration is now complete.

A notice:
An interface **comp** must be running on each server in the collective. But interface names must also be unique. Good practice is to number the comp interfaces, i.e. *DH_comp1.py*, *DH_comp2.py*, *DH_comp3.py*, etc. It is important that the name of the comp interfaces always begins with **DH_comp**.

3.3.4. Points for monitoring the interfaces

As already described above, each interface requires four points for monitoring.

1. **sy_NAME_status:**
   The *watchdog* interface regularly checks whether the NAME interface is still running and writes a 1 for running or a 0 for not running.
2. **Wph_NAME:**
   Wph stands for values per hour. The interface writes to this point the number of values that it sends to the servers per hour. The *watchdog* that runs on the same computer as the interface itself must be specified as the interface. The name of the computer is in the *Info* parameter. This point provides information about how regularly data is delivered from the interface.
3. **ZslW_NAME:**
   ZslW stands for time since last value. The **calc** interface periodically calculates the time that has passed since the last value of the interface was written to the akt table. If the script of an interface is not carefully written, an interface can end up going in circles. It's still running, but it no longer delivers any values. It's like a dog trying to bite its own tail. The status point always shows a 1. The value of the ZslW - Point, however, continues to increase.
4. **Zlw_NAME:**
   This is a special point because its values are not written into the archive. The compressor regularly deletes the values from the *akt* table. It only leaves the latest value. The point only serves as a supplier for the point *ZslW_NAME*. The value of the point is the Unix timestamp of the last transfer of values from the interface. The Point_ID of the point *ZlW_NAME* is required to calculate the point *ZslW_NAME* and must therefore be in the expression that is in the Info field of the point *ZslW_NAME*.

3.3.5. Malfunctions and their elimination

If an interface no longer supplies data, you can simply restart it. But it is better if you check beforehand why the interface no longer delivers anything.
If there is an error in the program, you should find an entry in the log table.

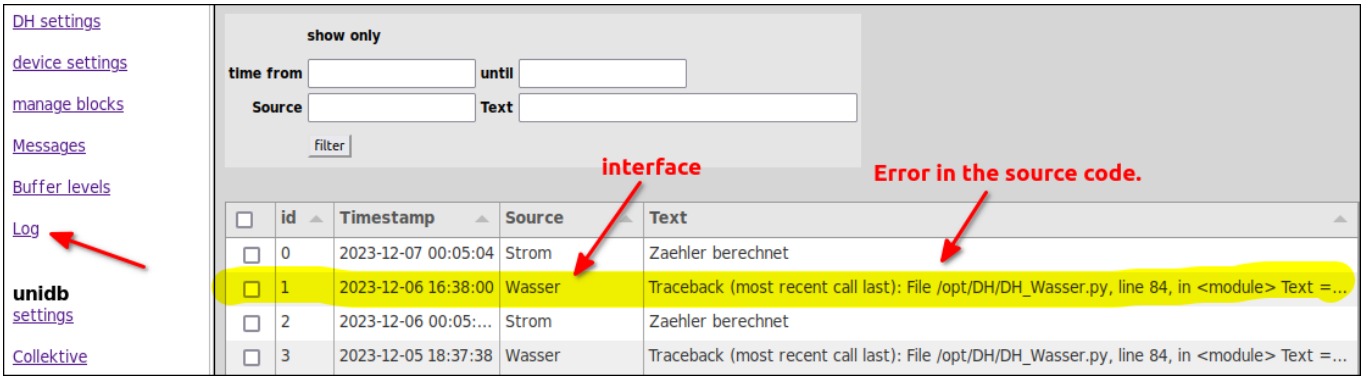


Fig. 30: Wasser interface has failed, error in the script.

However, it could also be the case that the interface no longer had a connection to the database when it failed. Or the script is simply not programmed to write a message to the log table. In such cases, it is

worth taking a look at the temporary directory, which is also used to write messages to the interfaces. Usually this is the */opt/DH/tmp* folder. In this folder you will find a file with the name of the interface and the extension .log. This contains the text that a script normally writes to the console. The interfaces redirect the output to the console to this file.
Before restarting an interface, please make sure that there is not another instance of it still running. You can do this either with the command **ps -e|grep DH_** via the console or via the admin area on the website in the *interfaces* form.

3.3.6. Standard interfaces

- **comp:**
  This interface must run on every server. It reads the values from the table *akt* and writes them compressed into the archive. It then deletes the values that are no longer needed in the table *akt*. There are no points directly assigned to it. It processes all points whose parameters are **scan**=1 and **archive**=1.
  Configuration:
  von_Point_ID and **bis_Point_ID** indicate the range of points for which the interface is responsible. If *comp* needs too much time to process all points, then a second *comp* interface can be configured. The interfaces then only process the area that was assigned to them via **von_Point_ID** and **bis_Point_ID**.

- watchdog:
  This interface must run on every computer in the collective. It monitors the other interfaces and distributes messages to them. If necessary, it can also start another interface.
  The points of this interface require the name of the computer on which the interface is running in the **Info** field.

- **calc:**
  It doesn't matter which computer in the collective runs *calc*. The interface calculates the expression that is in the **Info** field in the point configuration for each of the points assigned to it. The time interval at which the values are calculated is determined for each point using the **Interval** parameter in the point configuration.

- **batchfl:**
  No points are directly assigned to this interface. It reads values from flat text files and writes them to the table *akt*. The text files must have the file extension **.dat**. The batchfl changes the file extension to **.xxx** as soon as it has read in the values. This prevents a file from being read in multiple times. On the other hand, you can always see which data was read in via the *batchfl* interface.
  Each value must be on a separate line in the file. In addition, a timestamp and the Point_ID are required in the same line. The three pieces of information are each separated by a comma. This results in the following format for the data in a **.dat** file: Point_ID, timestamp, value

  Important: The decimal separator is a period.

  Example of the contents of a .dat file:
  4711,2023-12-27 15:17:19,36.47
  815,2023-12-27 15:15:33,151.3

  Plain text: For the point with the Point_ID, the value 36.47 is written to the table *akt* together with the timestamp 2023-12-27 15:17:19. Another entry in the current table: Point_ID: 815, timestamp: 2023-12-27 15:15:33, value: 151.3.

  The interface requires the **path** parameter in the configuration, which specifies the folder in which the interface looks for new **.dat** files.

3.4. Collective

Interfaces do not write their data directly to a server. You use a function instead. This function looks at which servers are present in the collective and then attempts to write the data to each of these servers. If

a server is not accessible, it writes the data to its own local database and then delivers it to the server as soon as it is accessible again. The function doesn't care how many servers it has to supply.
It follows that from the perspective of the interfaces, we are always dealing with a collective, even if there is only one server. Since an interface can also run on a server, you only need at least one computer for the collective. Actually, you can't call a single computer a collective, but that doesn't matter about the interfaces.

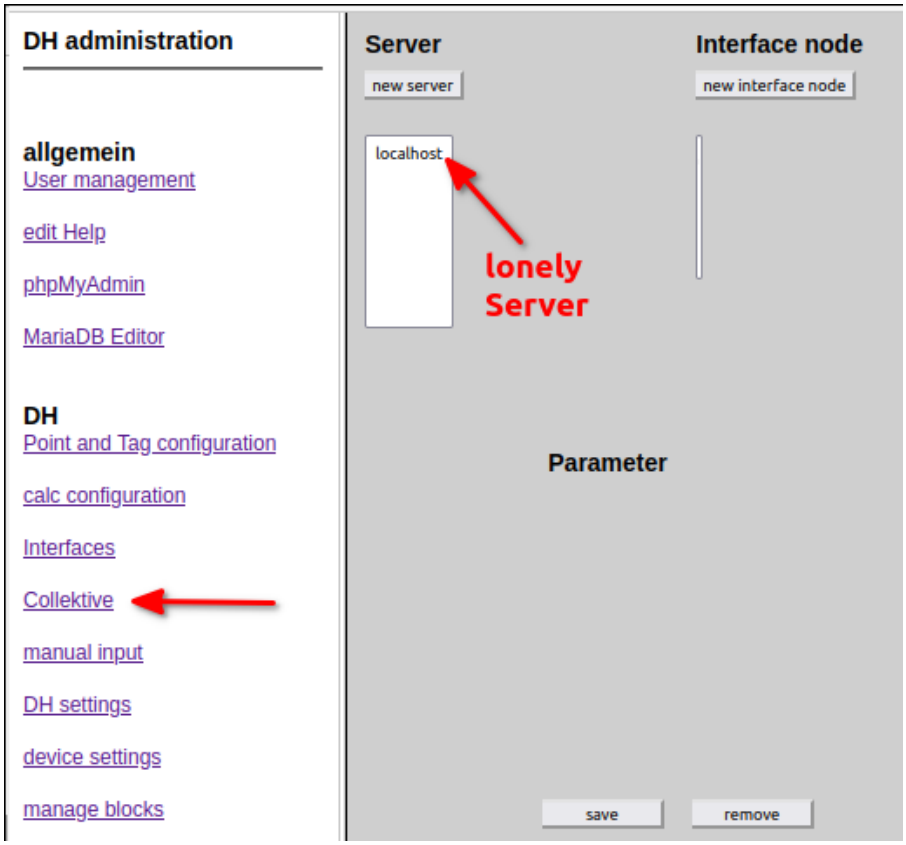


Fig. 31: *Collective* form in the administration area

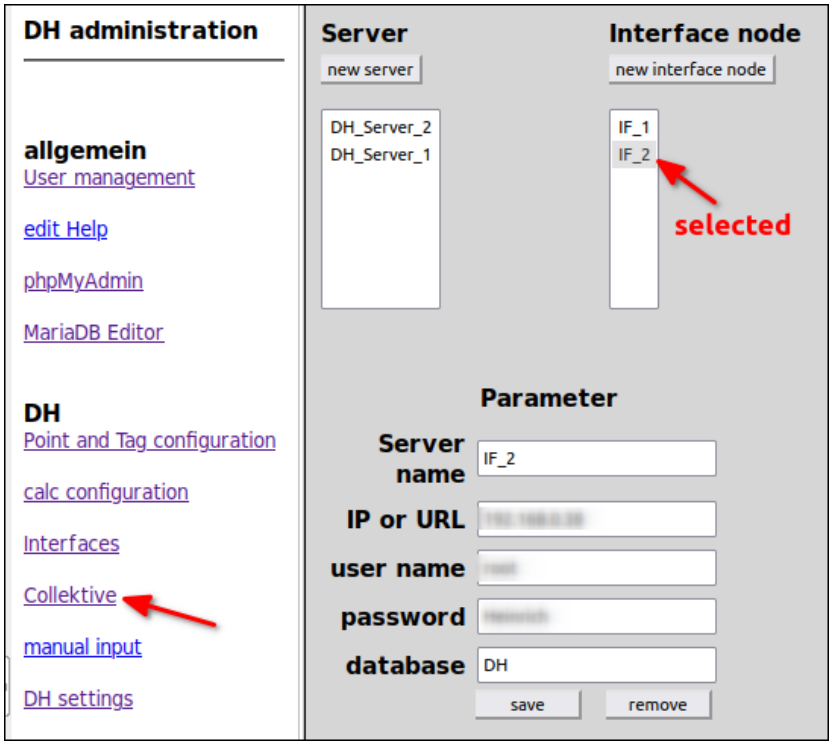


Fig. 32: Collective of two servers and two interface computers.

The difference between a server and an interface computer is that an interface computer requires a local database, but does not create an archive there. It also does not contain any tables for configuring points, tags, interfaces, etc. The local database is only used to buffer values in case a server is not reachable.

3.4.1. Interfaces in the collective

The computer on which an interface is running is specified in the Interfaces form as an addition to the Script parameter.

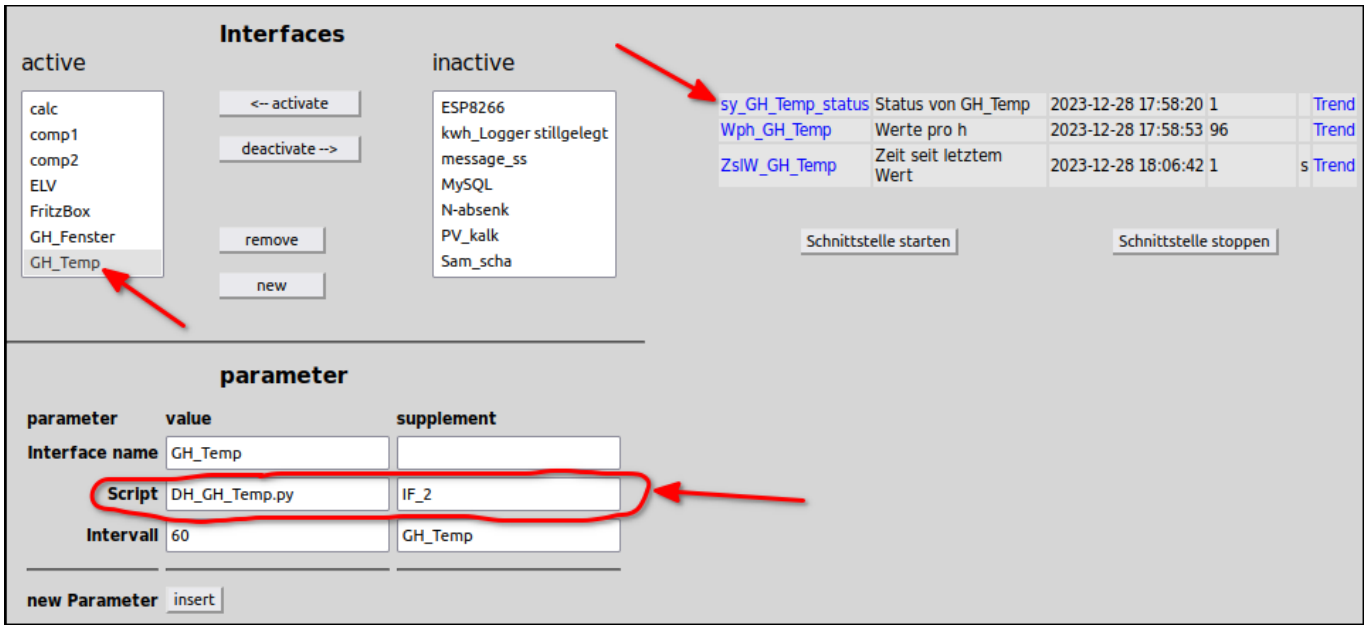


Fig. 33: Configuration of an interface that runs on the interface computer IF_2.

Furthermore, the computer name must be entered in the *Info* field of the *sy_NAME_status* point.
In the example above it looks like this:

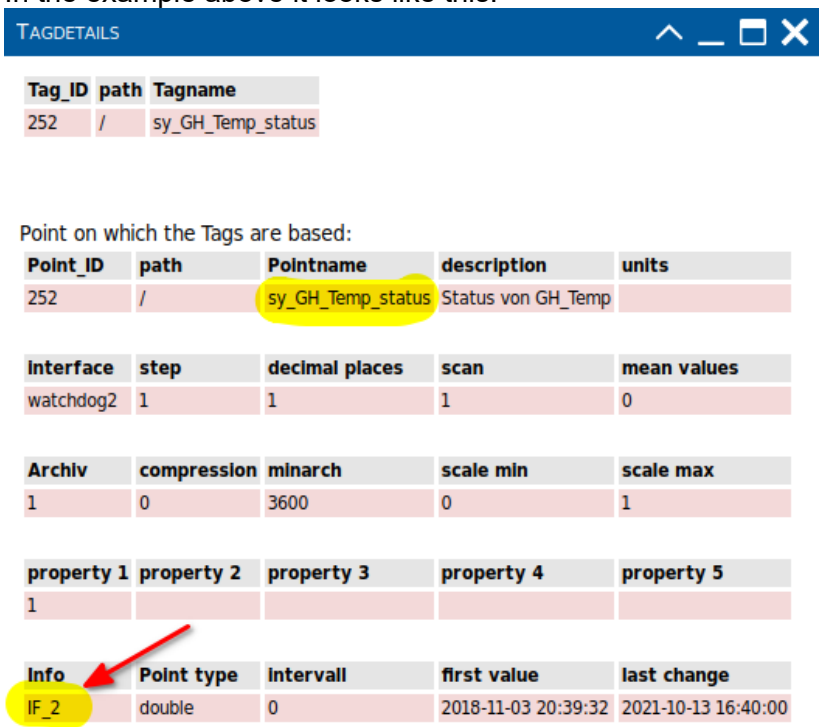


Fig. 34: Configuration of the *sy_NAME_status* point.

The *watchdog* interface must run on every computer in the collective. However, the name of an interface must be unique across the collective. It is a good practice to simply number the *watchdog* interface.
You should do the same with the *comp* interface. This must run on every server in the collective.

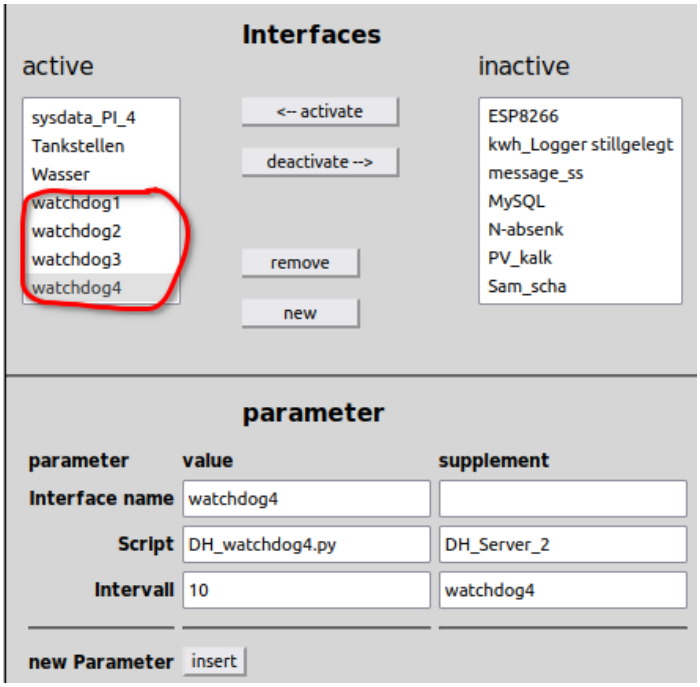


Fig. 35: Four watchdogs, simply numbered.

3.4.2. Add or remove interface computers from the collective.

To add an interface computer to the collective, there are only a few things that need to be done.

1. Copy the scripts for the interfaces, the scripts *DH_start.py*, *DH_stop.py* and *DH_biblio2.py* into the */opt/DH* folder.
2. Create a new *tmp* directory in the */opt/DH* folder.
3. Create a database called *DH* on the new interface computer and import the SQL file for interface computers there.
4. In the *collective* form (admin area), press the **new interface** switch at the top right. Then fill in the *Server Name*, *IP or URL*, *User*, *Password* and *Database* fields.
5. In the Interfaces form, create a new interface starting with the name *watchdog*.

Removing an interface computer from the collective is very easy. In the *collective* form (admin area), simply select the interface computer and press the **remove** switch at the bottom of the form. That was almost it. It looks nice if you now deactivate the interfaces that are assigned to the former interface computer in the Interfaces form.

3.4.3. Add or remove servers from the collective.

Adding a server to the collective requires a little more effort than a new interface computer.
First proceed as described in Chapter 3.4.2.. However, in step 3, select the SQL file for server. In point 5 you create another interface of type *comp*.

The larger the archive in the collective, the longer it takes to copy this archive to the new server. So copying may take up most of the time when setting up a new server. Before you start, take a moment to think about server synchronisation. All servers should hold the same data. A gap in a server's data isn't particularly nice. Most data is historical and will no longer be changed. Therefore, all files from the */var/lib/DH* directory from an existing server can be copied to the new server. You should only leave out the directory for the current month. So, as the root user, create a new DH directory in the */var/lib* folder and copy the archive except for the current month to the new server. Once that's done, stop the *comp* interface on the server from which you copied the archive. From now on no new data will be written to the archive. Instead, the data is buffered in the table *akt*. Now you can also copy the current month's folder to the new server.

Run the *DH_start.py* and *DH_stop.py* scripts on all computers in the collective to restart the interfaces. This is necessary so that the interfaces also know that they also have to send their data to the new server.

Copy the contents of the *akt* table from the server from which you copied the archive to the *akt* table on

the new server. This means that the data gap on the new server should only last a few minutes. Please don't forget to restart the *comp* interface that you just stopped.

To remove a server from a collective, first proceed as described for interface computers: select server in the *collective* form, press **remove** switch, deactivate interfaces in the *interfaces* form. Now you should run the scripts *DH_start.py* and *DH_stop.py* on each computer in the collective. If you don't do this, the interfaces will continue to diligently write data to the *akt* table in the database. Turn off the server and then write the data to its buffer. But you will never get the buffer empty because the server no longer exists.

3.5. additional forms in the admin area

3.5.1. Settings DH

In principle, you can also carry out all configuration via the Settings_DH page. However, please leave the names below the root of the tree structure unchanged. Otherwise pretty much nothing works anymore.

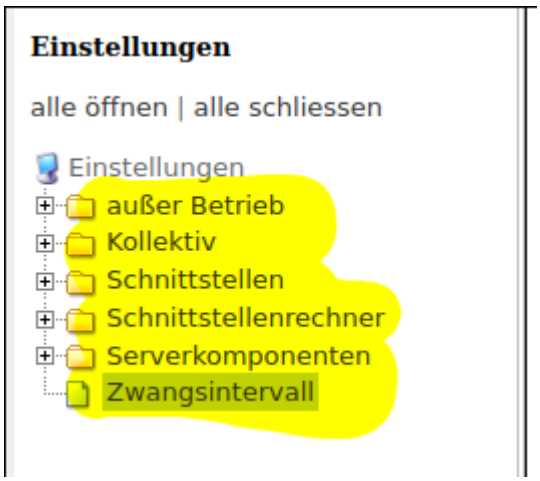


Fig. 36: Never change designations marked in yellow!

The *Zwangsintervall* (forced interval) setting is also worth mentioning here. This may sound like coercion and violence, but it is harmless. The *Zwangsintervall* simply specifies after how many seconds an interface should definitely send a value for a point to the servers, even if the value has not changed. Normally, values that have not changed since they were last read in are not even sent from the interface to the server. But there are points whose value rarely changes. Example: A price for a product read from some site on the Internet. The interface queries the website every minute, but the price only changes 1-2 times a day. If the compulsory interval did not exist, then you would not be sure whether the point is “still alive” if you see a value there that may already be two days old. But this way the user knows that the point is still being supplied with fresh values and that the current value displayed is not older than 10 minutes. The default *Zwangsintervall* setting is 600 seconds.

3.5.2. Device settings

If you want, you can make settings here, but so far only a few prototypes of these devices exist. Series production, and therefore the availability of the devices, is not yet planned (as of the end of 2023).
The devices are economical microprocessors with a built-in WLAN module. 1-wire sensors can be connected to these devices. They also have one analog and several digital inputs. They are operated either with a 5V power supply or with a battery pack, which can be replaced during operation. Depending on the frequency of data transfer, one battery pack is sufficient to operate the device for up to four weeks. The 1-wire sensors are equipped with plugs. This means they can be connected to the devices quickly and easily. Extension cables and distributors are also provided for the 1-wire sensors.
The devices are intended to temporarily collect data from one location and send it to the DH or to permanently send data from a location where a network cable is not available. In the latter application, however, it is more convenient if at least one power source is available nearby. Replacing the battery pack every two to four weeks can be annoying.

3.5.3. Manage building blocks

Here, as in the developer manual under 2.3.1. described how to create building blocks for images. The difference is that here you create building blocks that are visible to every user.

3.5.4. Messages

Displays the *Meldungen* table. This table contains all messages waiting to be picked up by the *watchdog* interfaces. This table should be as empty as possible. However, if you find messages here with a time stamp that is older than a minute, then some *watchdog* is no longer running. Please check the status of the *watchdog* interface first before deleting anything here.

3.5.5. Buffer levels

The buffer levels of all computers in the collective are displayed here. The buffer levels usually show 0. If this is not the case, then the computer in question cannot reach a server or there is nonsense in the buffer. This is usually the case if you only see very few entries there. You can use the buttons with the names of the computers to display the respective contents of the buffer. If you find any nonsensical entries, please delete them. Otherwise the computer desperately tries to get rid of the contents of the buffer every minute.

3.5.6. Log

Interfaces do not only write to the log table when they fail or are switched off. Depending on how they are programmed, they also write information here that is otherwise relevant in some way.

4. DH_SMT

DH_SMT is a Python program with GUI. It is used to edit the archives directly and should be treated with some caution. You can usually find it in the */opt/DH/DH_SMT* folder. Start the **DH_SMT.py** file there. The configuration for connections to the servers is in the */home/USERNAME* folder. There is a hidden folder (*./DH*) which contains a file called *DH_SMT.ini*. You can make the settings for the connections to the servers either directly here or via the Connections tab in the application.

4.1. Verbindungen

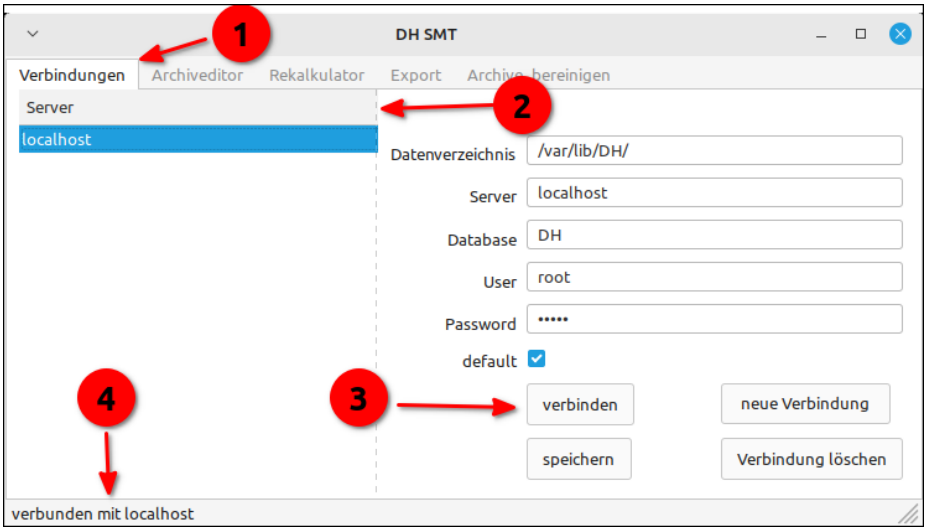


Fig. 37: Connections tab

The first card, Verbindungen (1), lists all available connections to servers on the left (2). It doesn't matter whether these servers belong to the same collective or not. On the right there is a switch that you can use to create additional connections. Select a server on the left and the connection details will be displayed on the right. The connection is established to the selected server by clicking on the **verbinden** button (3). You can always see which server the application is currently connected to in the status line below (4).

important:

If you are connecting to a remote server, you must specify the complete path starting from the local

computer as the data directory. Before establishing a connection, it must be ensured that the data directory is also accessible. To do this, the file system, or at least the required part of it, must be mounted via NFS.

4.2. Archive editor

Sometimes it happens that values are written to the archive that are not plausible. That's usually not a bad thing. But if these values are used somewhere for calculations, then this is no longer trivial. In the archive editor on the second tab, the values can be corrected or deleted. The archive editor also corrects the mean values, vt and vt_interpol.

Example:
For the air pressure, which is usually between 970 and 1040 mbar, a 0 is written into the archive twice in succession. Since the average values are also calculated for this point, this also has an impact on the hMW, dMW, hMin and dMin values.

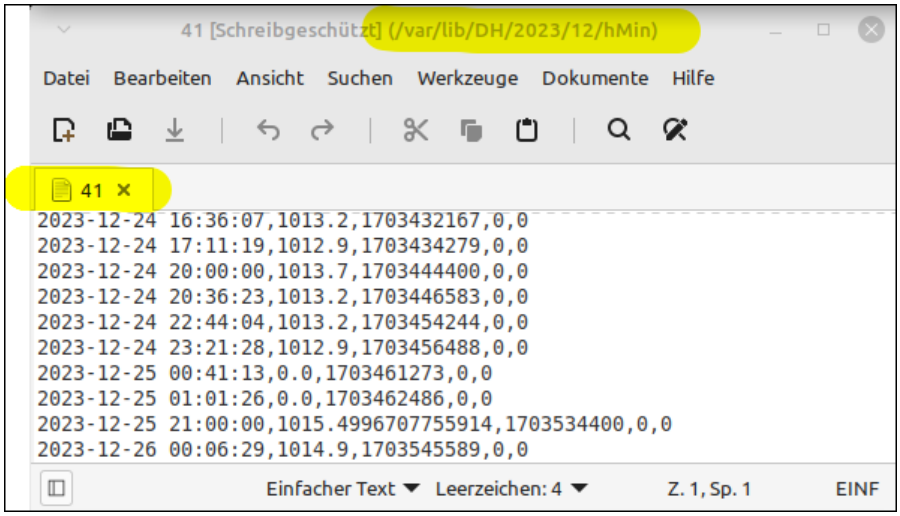


Fig. 38: The hMin values are also incorrect.

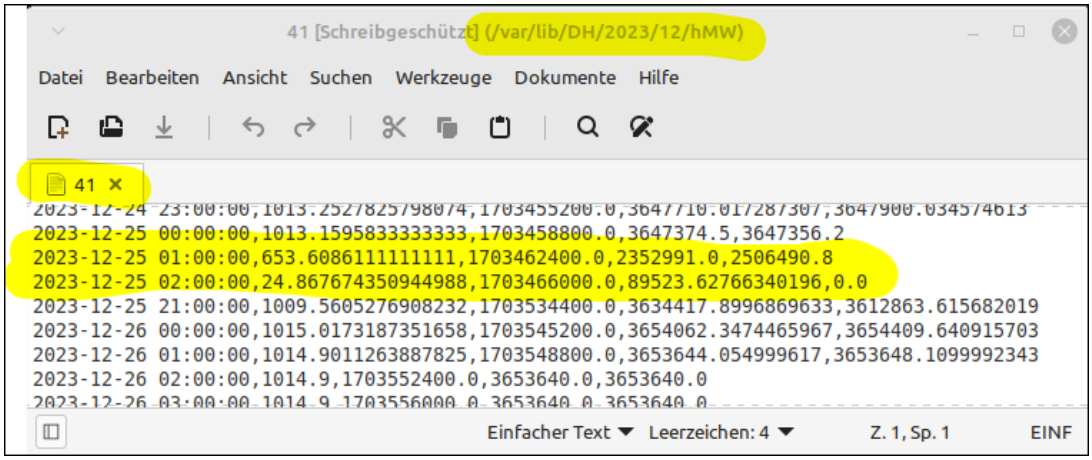


Fig. 39: The hMW is not credible during this period.

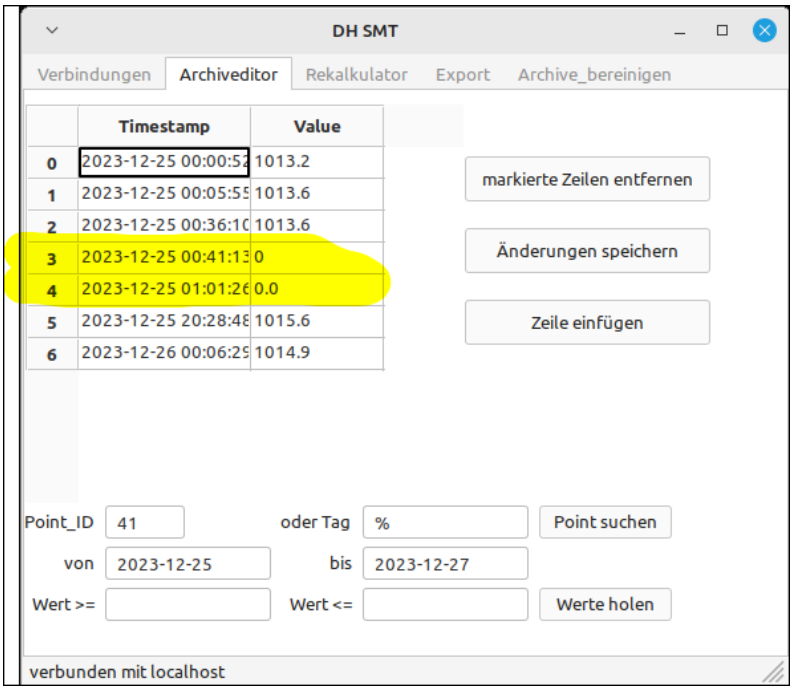


Fig. 40: Archive editor with incorrect values (Point_ID 41).

Now the implausible values are replaced by plausible values.

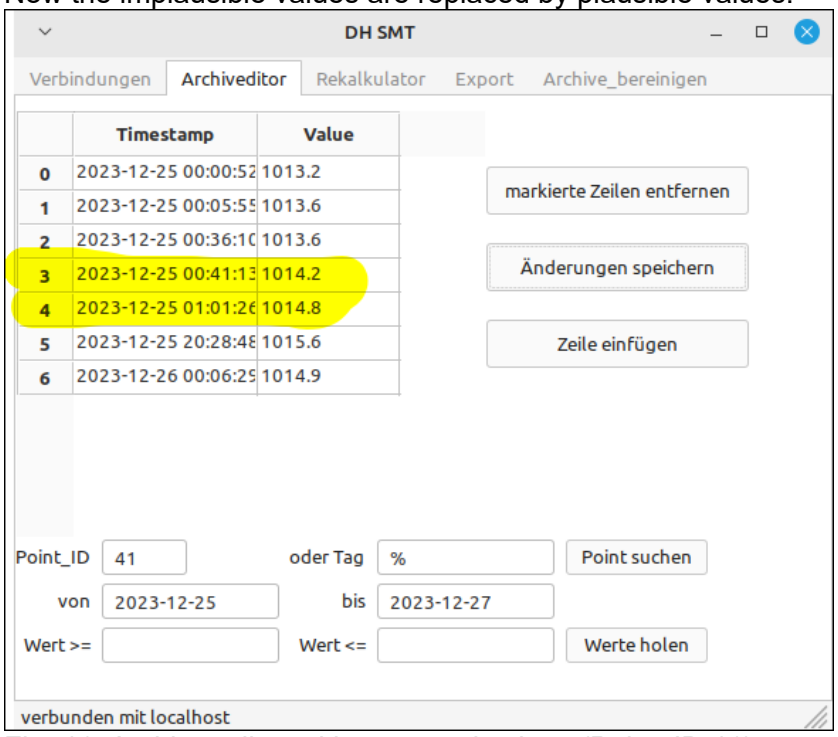


Fig. 41: Archive editor with corrected values (Point_ID 41).

The values for the vt and vt_interpol have also changed significantly. If you calculate with the vt or vt_interpol in a spreadsheet, it has significant effects.

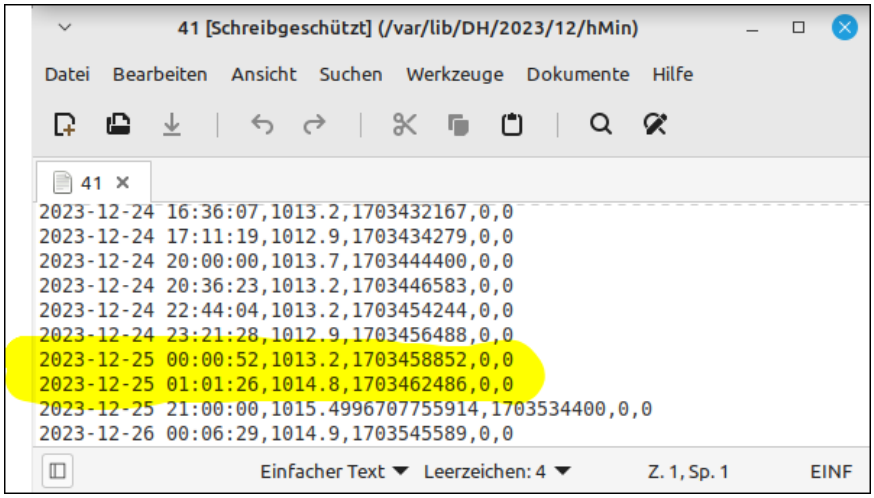


Fig. 42: new hMin values

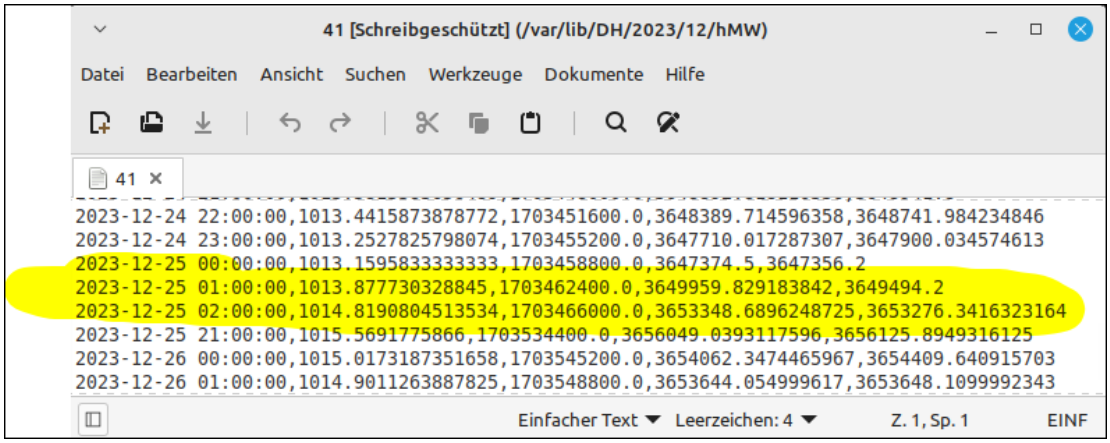


Fig. 43: new hourly average values with changed vt and vt_interpol.

However, it is better than manual correction if the interface carries out a plausibility check and therefore does not send any garbage data to the DH in the first place.

4.3. Rekalkulator

On this tab one or more points of the **calc** interface can be recalculated. All values of these points for the specified period are first deleted from the archive and then recalculated. The more points you want to recalculate and the longer the period selected, the longer you have to wait.

Attention!
Please look at the formula of each point before recalculating. You can find the expression to be calculated in the point configuration in the *Info* field. If *time.time()* is written somewhere there, then the current Unix timestamp is always used for the calculation. If you recalculate such a point, you will basically get a nonsensical result.

The situation is different for expressions that use the DH function *akt()*. During the recalculation the function is replaced by *AW()*. *AW()* uses the timestamp that corresponds to the current time at the calculation time and thus simulates the *akt()* function at this time.

4.4. Export

It often happens that as an admin you are asked by users whether they can get the values over a long period of time for a point in a table. Basically, every user has an export function available, which outputs the values as a csv file. However, exporting may take some time. It is then more convenient to write the values to a file using the **Export** tab. Even if you want to completely remove a point from the DH, using this tab is a good option to write all of a point's values somewhere before permanently deleting the values from the archive. This export is also very useful when exporting the values to another system.

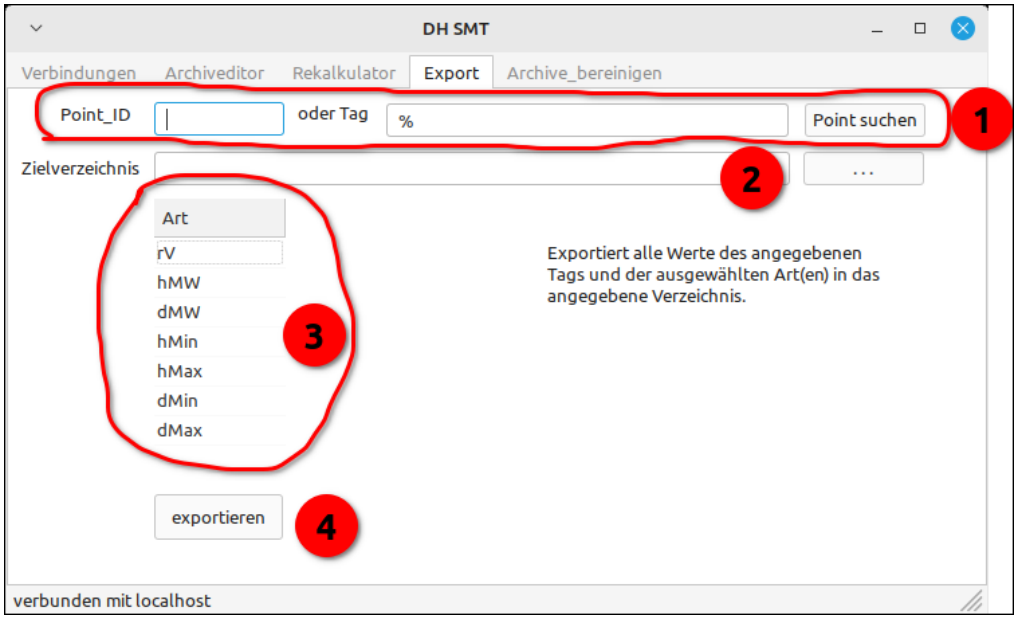


Fig. 44: Export tab

Either enter the desired Point_ID directly or search for it using the search field (1). Enter the directory in which you want to write the values (2). Now select whether you want to export the raw values (rV) and/or the statistical values (3). By clicking on the **exportieren** button you will quickly receive all the values of the point in a csv file (4).

4.5. Archive bereinigen.

Unfortunately, it sometimes happens that archive files were not written properly. There can be empty lines at the beginning or end of the files. It has also happened that values from the previous month have crept in between the values of the following month.
Whatever the case, it never hurts to clean up the archives every now and then. The data is sorted in the specified period. Empty lines are removed and entries from other months are deleted. Only the first line of each file contains the last value of the previous month and that is how it should be.

A notice:
If a trend looks a bit strange because it is supposed to show one day's values, but instead the timeline extends over several weeks and the trend line occasionally makes a trip to the beginning of the timeline, then values from the previous month have most likely crept in. Just clean up the current month's archive files and everything will be fine again.