whdp Documentation

Release 0.1

Uwe Schmitt

May 21, 2019

Contents

1 Instructions for data users **3**
- 1.1 Connect to water hubgdata pool . . . 3
- 1.2 Example SQL queries . . . 4

2 Instructions for data provider **5**
- 2.1 System Architecture . . . 5
- 2.2 General workflow . . . 5
- 2.3 Add Raw Data to Existing Source . . . 6
- 2.4 Add Site . . . 6
- 2.5 Add or Modify Parameters . . . 7
- 2.6 Add a new Source Type . . . 7
- 2.7 Add Source Instance . . . 7

3 Instructions for Admin **15**
- 3.1 Installation on Ubuntu 18.04. LTS . . . 15

4 Database layout **19**
- 4.1 signal . . . 19
- 4.2 site . . . 20
- 4.3 picture . . . 20
- 4.4 source . . . 20
- 4.5 source_type . . . 21
- 4.6 special_value_definition . . . 21
- 4.7 parameter . . . 21
- 4.8 parameter_averaging . . . 21
- 4.9 comment . . . 22
- 4.10 signal_quality . . . 22
- 4.11 quality . . . 22
- 4.12 Design priniciples . . . 22

5 Command references **25**

6 How to contribute to documentation **27**
- 6.1 Initialization . . . 27
- 6.2 Typical workflow . . . 27

7 Obsolete documentation **29**

7.1 Run server in test mode 29
7.2 Workflow example 30

8 Indices and tables **33**

If you want to work with data stored in the water hub datapool, read the data user instruction. If you want to load measurements data in the water hub data pool, read the data provider instructions.

Content:

CHAPTER 1

Instructions for data users

Data users are scientists analysing data from the water hub data pool.

All data is store in a PostgrSQL database so that arbitrary queries can be performed. The figure below shows the database layout.

1.1 Connect to water hubgdata pool

To connect with the database you need the following information from your admin:

- The host, e.g. “a.server.com”
- The port of the database, e.g. 5432
- The name of the database, e.g. “whdp”
- A database user name, e.g “data_user”
- The database password

You can connect directly to the database via psql. However, it is more convenient to load the data required directly in the environment used for further analysis:

1.1.1 R

With the RPostgreSQL package data can be loaded directly into R.

However, for *Eawag internal* usage the DatapoolR package should be preferred. It connects automatically to the database and provides convenient helper functions.

1.1.2 Python

Different options exist, psycopg2 is widly used.

1.2 Example SQL queries

The SQL language may look cumbersome at first. However, it gives a lot of flexibility and allows to express even very complex queries. This SQL tutorial is a helpful reference.

Also note that the DatapoolR package provides functions to simplify common queries.

List all data sources:

```
SELECT srctype.name, src.name, src.serial, srctype.description, src.description
FROM source_type AS srctype, source AS src
WHERE src.source_type_id = srctype.source_type_id;
```

List all sites and check how many images a site has:

```
SELECT name, coord_x, coord_y, coord_z, street,
  postcode, COUNT(picture.filename), site.description
FROM site
  LEFT JOIN picture ON site.site_id=picture.site_id
GROUP BY site.site_id;
```

Get all signals measured at site `site_A` within a given time interval:

```
SELECT signal.timestamp, value, unit, parameter.name, source_type.name, source.name
FROM signal
  INNER JOIN site ON signal.site_id = site.site_id
  INNER JOIN parameter ON signal.parameter_id = parameter.parameter_id
  INNER JOIN source ON signal.source_id = source.source_id
  INNER JOIN source_type ON source.source_type_id = source_type.source_type_id
WHERE site.name = site_A AND
  ?tmin::timestamp <= "2017-01-01 00:12:00"::timestamp AND
  signal.timestamp <= "2017-01-01 00:18:00"::timestamp;
```

CHAPTER 2

Instructions for data provider

Data provider is usually the scientist performing measurements in the field.

2.1 System Architecture

Data flow model `specification`.

The data provider must make sure that i) the raw data arrive in the landing-zone at the right place, ii) a conversions script exist to standardize the raw data, and iii) meta-data are provided.

2.2 General workflow

Working on the real landing-zone would be dangerous. Therefore, all development and testing is done on a copy of the landing-zone. The datapool provides a command to create development landing-zones. A development landing-zone can have any names, but let's call it `dlz` for now:

```
$ pool start-develop dlz
```

This creates a folder (a copy of the real landing-zone) called `dlz` in the home directory. You can see how the created landing zone looks like with `ls -l dlz`.

The datapool provides various checks to ensure that the provided conversion scripts and meta-data are consistent. The checks are ran by:

```
$ pool check dlz
```

If everything is fine, modify the develop landing-zone (e.g. add a new sensor) according to the instructions given below. After the modifications run the checks again.

```
$ pool check dlz
```

It is recommended to execute this checks after any small changes. If this succeeds, update the operational landing zone:

```
$ pool update-operational dlz
```

All future raw data should be delivered directly into the operational database.

In the following sections, the different types of modifications/additions are explained.

2.3 Add Raw Data to Existing Source

Raw data files are written to the respective `data/` folders in the operational landing zone as follows:

1. A new file, for example `data.incomplete`, is created and data are written to this file.
2. Once the file content is complete and the corresponding file handle is closed, the file is renamed to `data-TIMESTAMP.raw`.

Note, the file must end with ‘‘.raw‘‘! The actual format of `TIMESTAMP` is not fixed but must be unique string, that starts with a dash `-`, and can be temporarily ordered. Encoding a full date and time string will help the users and developers to inspect and find files, especially if present in the backup zone.

This procedure is called *write-rename pattern* and avoids conversion of incomplete data files. The risk for such a race condition depends on the size of the incoming data files and other factors and is probably very low. But running a data pool over a longer time span increases this risk and could result in missing data in the data base.

2.4 Add Site

In order to add a new measuring site, the information about this site have to be provided as a `site.yaml` file in a new folder for the site, within the `sites` folder of the landingzone. The information to be specified are:

- Name: Name of the site
- Description: Free Text describing the particularities of the site
- Street, City and Coordinates (CH1903/LV03): Specifying where the site is located
- Pictures (optional): Pictures relating to the site can be specified. Pictures are normally stored in the `images` folder of the specific site.

The structure of the file has to be the same as in the example below:

```
name: industry
description: Site at pump house at industry. Installed nearly on top of the  pump.␣
↪Detection of pump possible! When a pump must be removed the sensor has to be taken␣
↪out and installed after again.
street: Undermülistrasse
city: 8320 Fehraltorf, Switzerland
coordinates:
    x: 47.394973
    y: 8.733811
    z:

# pictures are optional:
```

(continues on next page)

(continued from previous page)

```
pictures:
    -
      path: images/installation.png
      description: Installation of Lora Ultrasonic Sensor 00769
      # date is optional:
      date: 2016/08/22 12:00:00
    -
      path: images/impact.jpg
      description: Impact Zone of Sensor 00769
```

2.5 Add or Modify Parameters

The file `parameters.yaml` is stored in the `data` folder and contains all the parameters. New parameters can be added here. The information to be included are:

- Name: Name of the Parameter
- Unit: Specifies the unit of the Parameter
- Description: Additional description of the parameter. In case there is no description required, the field can remain empty.

The syntax has to match the following example (note the dash in the first line):

```
-
   name: Absorbance 202.50_nm
   unit: m-1
   description: absorbance at 202.50 nm wavelength
```

2.6 Add a new Source Type

TODOXS

2.7 Add Source Instance

todo

2.7.1 Conversion of raw data

The files arriving in the landing zone are called *raw data*. Every raw data file must be converted into a so called *standardized file* by a conversion script. The format of the standardized files is defined below. Typically, every source instance needs an individually adapted conversion script.

2.7.2 Standardized file format

The standardized file format for the input data is a `csv` file with either six or four columns. It must adhere the following standards:

- File format: `csv` file with semicolon delimited (`;`)

- Data format: `yyyy-mm-dd hh:mm:ss`
- Column names: The first row contains the column names. The first three are always: `timestamp`, `parameter`, `value`. Next either the three columns `x`, `y`, `z`, or a single column `site` must be given. The parameter must exisit in the `parameters.yaml` and have the exactly same name (see above).
- `value` column: Must contain only numerical values. Missing values (`NULL`, `NA`, or similar) are not allowed.
- The z-coordinate columns may be empty.

2.7.3 Example standardized file format with coordinates

timestamp	parameter	value	x	y	z
2013-11-13 10:06:00	Water Level	148.02	682558	239404	
2013-11-13 10:08:00	Water Level	146.28	682558	239404	
2013-11-13 10:08:00	Average Flow Velocity	0.64	682558	239404	412
...	...	...	...	...	

2.7.4 Example standardized file format with site

timestamp	parameter	value	site
2013-11-13 10:06:00	Water Level	148.02	zurich
2013-11-13 10:08:00	Water Level	146.28	zurich
2013-11-13 10:08:00	Average Flow Velocity	0.64	zurich
...	...	...	...

2.7.5 Conversion script

The conversion script must define a *function* which reads raw data and write an output file (a standardized file). The first argument if this function is the path to the input raw data, the second argument the path to the resulting file.

The follwing points should be considered when writing an conversion script:

- Indicate corrupt input data by throwing an exception within a conversion script. A informative error message is helpful and will be logged.
- If a converson script writes to `stdout` (i.e. normal `print()` commands) this may not appear in the datapool log file and thus might be overseen.
- All required third party modules, packages, or libraries must be installed globally. Do not try to install them within a script.

The following code snippets show how a conversion script could look like for different languages.

2.7.6 R

- The file must be named `conversion.r`.
- The function must be named `convert`.

```r
# Example R conversion script
# September  27, 2016 -- Alex Hunziker

library(reshape2)

convert <- function(raw_file, output_file){

   data.raw <- utils::read.table(raw_file, sep="\t", skip=1, header=F)
   names(data.raw) <- c("Date Time", "Water Level", "Average Flow Velocity", "Flow",
                        "Temperature", "Surface Flow Velocity", "Distance",
                        "Distance Reading Count",  "Surcharge Level", "Peak to Mean␣
↪Ratio",
                        "Number of Samples", "Battery Voltage")

   if(ncol(data.raw) !=12 )
       stop(paste("Error: Input File has", ncol(data.raw),
                  "columns, instead of the expected 12 columns."))

   if(!all(sapply(data.raw[2:ncol(data.raw)], is.numeric)==TRUE))
       stop("Error: Non-numeric input where numeric values were expected.")

   # define coordinate
   xcoor <- 682558
   ycoor <- 239404
   zcoor <- ""

   ## reformat data

   time <- strptime(data.raw$"Date Time", "%d.%m.%Y %H:%M")
   data.raw$"Date Time" <- format(time, "%Y-%m-%d %H:%M:%S")

   data.form <- reshape2::melt(data.raw, id.vars = c("Date Time"))

   colnames(data.form) <- c("timestamp", "parameter", "value")
   data.form$X <- xcoor
   data.form$Y <- ycoor
   data.form$Z <- zcoor

   # remove NA values
   data.form <- stats::na.omit(data.form)

   utils::write.table(data.form, file=output_file, row.names=FALSE, col.names=TRUE,
                      quote=FALSE, sep=";")

}
```

2.7.7 Julia

- The function must be named `convert`.
- The name of the julia file and the declared module must be the same (up to the `.jl` file extension). So the file containing the module `conversion_lake_zurich` must be saved as `conversion_lake_zurich.jl`.
- Further the module and file name must be unique within the landing zone.

```
# Example Julia conversion script
# September  27, 2016 -- Alex Hunziker

module conversion_FloDar_Fehraltorf_2

# ---> 1.) load required package (optional)

using DataFrames

function convert(raw_file, output_file)

  # ---> 2.) read file

  if(!isfile(raw_file))
    error("Error: raw_file does not exist.")
  end

  # the header line contains non-utf8 encoded characters, so we skip this:
  dataraw = DataFrame(readtable(raw_file, separator = '\t', skipstart=1,␣
↪header=false))

  names!(dataraw, map(symbol, ["Date Time", "Water Level", "Average Flow Velocity",
↪"Flow",
                               "Temperature", "Surface Flow Velocity", "Distance",
                               "Distance Reading Count", "Surcharge Level",
                               "Peak to Mean Ratio", "Number of Samples", "Battery␣
↪Voltage"]))

  ## ---> 3.) test properties

  if(size(dataraw, 2) != 12)
    error("Imput File has wrong number of columns.")
  end

  ## ---> 4.) add additional information (optional)

  #Define coordinate
  xcoor = 682558
  ycoor = 239404
  zcoor = ""

  ## ---> 5.) reformate data

  selCol = symbol("Date Time")
  time = Dates.DateTime(dataraw[selCol], "dd.mm.yyyy HH:MM")
  dataraw[selCol] = Dates.format(time, "yyyy-mm-dd HH:MM")

  dataForm = stack(dataraw, [2:12], selCol)
  dataForm = dataForm[:, [selCol, :variable, :value]]
  dataForm[4] = xcoor
  dataForm[5] = ycoor
  dataForm[6] = zcoor
  names!(dataForm, [:timestamp, :parameter, :value, :x, :y, :z])

  deleterows!(dataForm, find(isna(dataForm[:, symbol("value")])))

  ## ---> 6.) write file
```

(continues on next page)

(continued from previous page)

```
  writetable(output_file, dataForm, separator = ';')

end

end
```

2.7.8 Python

```
# Example Python conversion script
# September  27, 2016 -- Alex Hunziker

# ---> 1.) load required packages (optional)
import os.path
import pandas

def convert(raw_file, output_file):

    # ---> 2.) read file

    if not os.path.isfile(raw_file):
        raise ValueError('Error: Input File does not exist.')

    raw_data = pandas.read_csv(raw_file, sep='\t', encoding="latin-1")
    colNames = ("Date Time", "Water Level", "Average Flow Velocity", "Flow",
↪"Temperature",
                "Surface Flow Velocity", "Distance", "Distance Reading Count",
                "Surcharge Level", "Peak to Mean Ratio", "Number of Samples",
                "Battery Voltage")
    raw_data.columns = colNames

    # ---> 3.) test properties

    if len(raw_data.columns) != 12:
        raise ValueError('Error: Input File has wrong number of columns.')

    # ---> 4.) add additional information (optional)

    # Define coordinate
    xcoor = 682558
    ycoor = 239404
    zcoor = ""

    # ---> 5.) reformat data

    time = pandas.to_datetime(raw_data['Date Time'], format="%d.%m.%Y %H:%M")
    raw_data['Date Time'] = time.apply(lambda x: x.strftime('%Y-%m-%d %H:%M'))

    data = pandas.melt(raw_data, id_vars=['Date Time'],
                       value_vars=list(raw_data.columns[1:12]))

    data.columns = ['Date Time', 'parameter', 'value']

    data = data.dropna()
```

(continues on next page)

(continued from previous page)

```
    data['x'] = xcoor
    data['y'] = ycoor
    data['z'] = zcoor

    ## ---> 6.) write file

    data.to_csv(output_file, sep=";", index=False)
```

2.7.9 Matlab

- The function must be named `convert`.
- The file name must be named `convert.m`.

```
%
% SWW-DWH: Example MatLab conversion script
%
% 19/12/2016 - Frank Blumensaat
% Example: conversion('raw_data\data-001.raw','out.dat');
% ------------------------------------------------------

function conversion(fNameIn,fNameOut)

% read full content of the file into 'data'
fid = fopen(fullfile(fNameIn), 'r');
dataRaw = textscan(fid, '%s %f %f %f %f %f %f %f %f %f %f %f %f', Inf, 'Delimiter','\t',
↪'TreatAsEmpty',...
    {'NA'},'HeaderLines',1);
fclose(fid);

% possible to include check if 12 columns and numeric val's in col2 - col12

fid = fopen(fullfile(fNameIn), 'r');
names = textscan(fid, '%s %s %s %s %s %s %s %s %s %s %s %s %s', 1,'Delimiter','\t',
↪'HeaderLines',0);
fclose(fid);

% % parse string of TRANSFER time (time stamp) into ML number
datTime = datenum(dataRaw{1,1}(:),'DD.mm.YYYY hh:MM');

% define coordinates
xcoor = ones(length(dataRaw{1}),1).*682558;
ycoor = ones(length(dataRaw{1}),1).*239404;
zcoor = zeros(length(dataRaw{1}),1);

% split data matrix acc. to parameter and remove NaNs
for j = 2:size(dataRaw,2)
    dataSplit(j-1).var = excise([datTime dataRaw{1,j} xcoor ycoor zcoor]);
end

% some parameter names are not conforming to parameters.yaml:
parametersRaw =  {'Level', 'Velocity', 'Surface Velocity', 'PMR', 'NOS', 'Power Supply
↪'};
parametersUniform = {'Water Level', 'Average Flow Velocity', 'Surface Flow Velocity',.
↪..
```

(continues on next page)

(continued from previous page)

```
                    'Peak to Mean Ratio', 'Number of Samples', 'Battery Voltage'};

fixNames = containers.Map(parametersRaw,parametersUniform);

% write processed data to a cell array
celldata = {};
clear celldataTemp
for k = 1:length(dataSplit)
    for i = 1:length(dataSplit(k).var)
        celldataTemp{i,1} = datestr(dataSplit(k).var(i,1),'yyyy-mm-dd HH:MM:SS'); %␣
↪following the ISO 8601 data standard
        name = char(names{k+1});
        % our parameters.yaml does not have the units in (..), so we remove them:
        name = regexprep(name, '\(.*\)', '');
        % correct some names:
        if isKey(fixNames, name)
            name = fixNames(name);
        end
        celldataTemp{i,2} = name;
        celldataTemp{i,3} = dataSplit(k).var(i,2);
        celldataTemp{i,4} = dataSplit(k).var(i,3);
        celldataTemp{i,5} = dataSplit(k).var(i,4);
        celldataTemp{i,6} = '';
    end
    celldata = vertcat(celldata,celldataTemp);
    clear celldataTemp
end

%% write selected data to TXT file
fid = fopen(fullfile(fNameOut),'w');
fprintf(fid, '%s; %s; %s; %s; %s; %s \n', 'timestamp', 'parameter', 'value', 'x', 'y',
↪ 'z');
[nrows] = size(celldata);
for row = 1:nrows
    fprintf(fid,'%s; %s; %f; %d; %d; %d \n',celldata{row,:});
end
fclose(fid);
end

%% function to remove NaN values
function X = excise(X)
X(any(isnan(X)'),:) = [];
end
```

CHAPTER 3

Instructions for Admin

3.1 Installation on Ubuntu 18.04. LTS

Run the following instructions as `root` (this means, type `sudo` in front of every command) in order to have the root rights. If a file needs to be opened/modified, use the editor "nano" (type `nano` in front of the filename).

1. Ubuntu packages

```
$ apt install git r-base postgresql python3-pip python3-psycopg2
```

2. Install julia 1.0.1 and create a symbolic link: (Ubuntu offers julia 0.4.x only):

```
$ wget https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.1-
↪linux-x86_64.tar.gz
$ tar xzf julia-1.0.1-linux-x86_64.tar.gz
$ mv julia-1.0.1 /opt
$ ln -s /opt/julia-1.0.1/bin/julia /usr/local/bin/
$ julia --version
```

3. Create data base users and database

   We create a user `whdp` who owns the `whdp` database and thus has all permissions to modify tables and data in this database. In addition to that we create a `whdp_reader` user which only has read access.

```
$ sudo -u postgres createuser -P whdp
$ sudo -u postgres createuser -P whdp_reader
```

   Note down the choosen passwords for both users.

   Then create the database and restrict the pemissions of `whdp_reader`.

```
$ sudo -u postgres createdb -O whdp whdp
$ sudo -u postgres psql -c "REVOKE ALL PRIVILEGES ON ALL TABLES IN␣
↪SCHEMA public FROM whdp_reader;"
$ sudo -u postgres psql -c "GRANT SELECT ON ALL TABLES IN SCHEMA␣
↪public TO whdp_reader;"
```

Now check:

```
$ psql -U whdp -h 127.0.0.1 whdp
$ ^D    (control-D to exit postgress shell)
```

Next, the database must be configured to allow for remote access.

In the file `/etc/postgresql/10/main/pg_hba.conf` edit the line `host all all 127.0.0.1/32 md5` to `host all all 0.0.0.0/0 md5`.

In the file `/etc/postgresql/10/main/postgresql.conf` change `# listen_addresses = 'localhost'` to `listen_addresses = '*'`.

Restart the data base:

```
$ service postgresql restart
```

4. Install datapoool

```
$ cd /opt
$ git clone https://sissource.ethz.ch/sispub/whdp.git
$ apt install python3-pip
$ pip3 install -e whdp
```

Check installation:

```
$ whdp --help
```

Install needed packages for demo scripts:

```
$ /opt/whdp/scripts/setup_julia_et_al.sh
```

5. Create user account for data provider:

```
$ addgroup whdp
$ useradd -m -G whdp,systemd-journal -s /bin/bash whdp-provider
```

Assign password:

```
$ passwd whdp-provider
```

6. Initalize whdp configuration and setup landing zone:

   We assume that the landing zone will be located on a shared drive mounted at `/nfsmount`, but you are free to choose any other folder.

   Create the landing zone and link it to the data pool:

```
$ mkdir -p /nfsmount/landing_zone
$ pool init-config /nfsmount/landing_zone
```

Set the correct permissions:

```
$ chgrp -R whdp /nfsmount/landing_zone
$ chmod -R g+w /nfsmount/landing_zone
```

7. Adapt configuration:

```
$ /etc/whdp/whdp.ini
```

   Add the database user and password. Replace `DB_USER` and `DB_PASSWORD` with the one selsceted in step 3.

```
...
[db]
connection_string = postgresql://DB_USER:DB_PASSWORD@127.0.0.1:5432/whdp
```

   If necessary adapt also the path to the landing zone, define a backup landingzone, or change software versions.

   Then check:

```
$ pool check-config
```

8. Create the central management tool service for controlling the init system:

```
$ ln -s /opt/whdp/scripts/whdp.service /etc/systemd/system
$ systemctl daemon-reload
```

9. Start service:

```
$ systemctl start whdp.service
$ systemctl status whdp.service
```

10. Observe running service:

    can be stopped with ^C), can be used without `-f`:

```
$ journalctl -u whdp -f
```

    Keep this terminal window open if you want observe the whdp activities.

11. Install julia packages:

    Login as user `whdp-provider` first.

    Install needed Julia packages (these are installed per user) to be able to run the test scripts:

```
$ /opt/whdp/scripts/setup_julia.sh
```

CHAPTER 4

Database layout

A formal description of the data base layout used by the whdp datapool.

Legend:

- `pk` = Primary Key
- `fk` = Foreign Key
- `uq` = Unique within table
- A `field` in **bold letters** indicates a field which cannot be NULL

4.1 signal

This is the central table holding the measurements. Each row represents a *value* of a *parameter* measured at a given *time* and location (*site*). The coordinates of the signal may not correlate to entries in the *site* table.

field	datatype	description
signal_id	integer (pk)	
value	float	the actual measured value of the parameter
timestamp	date_time	time when the value was measured.
parameter_id	integer (fk)	parameter
source_id	integer (fk)	source
site_id	integer (fk)	site
coord_x	string	at a given site, a signal may origin from a specific place
coord_y	string	the type of coordinate system is CH1903/LV03
coord_z	string	elevation

4.2 site

A site is a location where measurements are made. At a given site, several measuring equipments (source) can be found. The location of the site is also described by its coordinates.

field	datatype	description
site_id	integer (pk)	
name	string (uq)	Name of that site
description	string	
street	string	street and number
postcode	string	postal code
city	string	name of the city/village
coord_x	string	coordinates of a given site
coord_y	string	the type of coordinate system is CH1903/LV03
coord_z	string	elevation

4.3 picture

Every site may contain a number of pictures. Filenames for each site must be unique. The filetype (e.g. png, jpg, tiff) is determined by the filename extenion of the `filename` field.

field	datatype	description
picture_id	integer (pk)	
site_id	integer (fk)	referring to the site
filename	string	
description	string	additional information about the picture
data	bytea	contains the (binary) content of the file
timestamp	date_time	creation date of the picture

4.4 source

A (data-) source is a specific measuring equipment. Every measurement (**signal**) origins from a specific source. Sources are categorized into **source_types**. The name of a source must be unique.

field	datatype	description
source_id	integer (pk)	
source_type_id	integer (fk)	source category
site_id	integer (fk)	optional reference to a site (may be NULL)
name	string (uq)	Name of that source. Usually is a combination of source_type and site name.
description	string	
serial	string	serial number. Is unique, if available
manufacturer	string	company which produced that equipment
manufacturing_date	date	date when the equipment was manufactured

4.5 source_type

Categorization of a given source.

field	datatype	description
source_type_id	integer (pk)	pk
name	string (uq)	Name of that source
description	string	

4.6 special_value_definition

Certain source types produce categorical data, such as «dry», «wet», «n/a» and so on. This table is used *to correlate categorical data and numeric values* for a given source type. For example the numerical value `1` might encode the state «dry».

field	datatype	description
special_value_definition_id	integer (pk)	
source_type_id	integer (fk)	source_type
description	string	
categorical_value	string	the catecorical value
numerical_value	float	the numeric value it is mapped to.

4.7 parameter

Every value in the **signal** table is connected to a specific parameter which describes and defines its unit.

field	datatype	description
parameter_id	integer (pk)	
name	string (uq)	e.g. “rain intensity”, “absorbance 200.00”, etc.
description	string	
unit	string	the physical unit, e.g. “mm/h”, “m-1”

4.8 parameter_averaging

Sometimes, data coming from a *source* comes already processed. For example, the windspeed is the average speed during a certain time period. These kind of information is parameter- and source-specific.

field	datatype	description
parameter_averaging_id	integer (pk)	not really necessary, since parameter_id and source_id together are unique
parameter_id	integer (fk)	parameter
source_id	integer (fk)	source
integration_length_x	float	data integration in meters
integration_length_y	float	data integration in meters
integration_angle	float	data integration in degrees
integration_time	float	data integration time in seconds

4.9 comment

There are two types of signal annotations: comments and quality. A comment is an arbitrary text, where as quality annotations have a controlled vocabulary. A signal may contain more than one comment.

field	datatype	description
comment_id	interger (pk)	
signal_id	integer (fk)	
text	string	the comment itself
timestamp	date_time	the time the comment was added
author	string	the username of the author who added the comment

4.10 signal_quality

A signal may contain more than one quality flag (but not the same quality flag twice). The combination of signal_id and quality_id must be unique.

field	datatype	description
signal_quality_id	integer (pk)	
signal_id	integer (fk)	
quality_id	integer (fk)	
timestamp	date_time	date when annotation was added
author	string	username of the author who added the annotation

4.11 quality

Measuring the environment is always error prone. This table holds the controlled vacabulary mentioned above. As some quality flags may be assigned programatically the *method* field indicates the origin of such an quality entry.

field	datatype	description
quality_id	integer (pk)	
flag	string (uq)	a textual description of *quality_id*
method	string	a description how the quality flag is generated.

4.12 Design priniciples

The design of the database follows the https://en.wikipedia.org/wiki/Star_schema to model multidimensional data with a https://en.wikipedia.org/wiki/Data_warehouse.

You find a graphical descripton of the star schema `here`.

We follow these principles to assure a consistent layout of the underlying tables:

- primary keys of a table are called `tablename\_id` instead of `id`
- table names are in singular
- the star schema avoids too much normalization

- a table should not contain too abstract information

CHAPTER 5

Command references

```
Usage: whdp init-config [OPTIONS] LANDING_ZONE_FOLDER

  initializes /etc/whdp/ folder with config files.

  landing_zone_folder must be a non-existing folder on the current machine.

Options:
  --verbose       dumps lots of output from interaction with db
  --use-sqlitedb  use sqlite db
  --force         use this twice to overwrite existing config files
  --help          Show this message and exit.
```

```
Usage: whdp check-config [OPTIONS]

  checks if config file(s) in /etc/whdp are valid.

Options:
  --verbose  dumps lots of output from interaction with db
  --help     Show this message and exit.
```

```
Usage: whdp init-db [OPTIONS]

  creates empty tables in operational database. Run check_config first to
  see if the configured data base settings are valid.

Options:
  --verbose  dumps lots of output from interaction with db
  --force    use this twice to overwrite existing db
  --help     Show this message and exit.
```

```
Usage: whdp check [OPTIONS] DEVELOPMENT_LANDING_ZONE_FOLDER

  checks scripts and produced results in given landing zone. does not write
```

(continues on next page)

(continued from previous page)

```
  to database.

Options:
  --result-folder TEXT  provide target for results
  --verbose             might dump lots of output
  --help                Show this message and exit.
```

```
Usage: whdp start-develop [OPTIONS] DEVELOPMENT_LANDING_ZONE_FOLDER

  setup local landing zone for adding new site / instrument / conversion
  script. this command will clone the operational landing zone (might be
  empty).

Options:
  --verbose  dumps lots of output from interaction with db
  --force    use this twice to overwrite existing db
  --help     Show this message and exit.
```

```
Usage: whdp update-operational [OPTIONS] DEVELOPMENT_LANDING_ZONE_FOLDER

  deploys local changes to operational landing zone.

Options:
  --verbose         might dump lots of output
  --force           use this twice to overwrite existing landing zone in case
                    of errors when checking
  --copy-raw-files  copy raw files also to operational landing zone
  --help            Show this message and exit.
```

CHAPTER 6

How to contribute to documentation

6.1 Initialization

To checkout the full repository, you need to configure `git` first.

```
$ git clone https://sissource.ethz.ch/sispub/whdp.git
$ cd whdp
$ git branch --track docs origin/docs
$ git checkout docs
```

Then install the packages needed to build the documentation:

```
$ cd docs
$ pip install -r requirements.txt
```

6.2 Typical workflow

6.2.1 Update your local repository

To fetch the recent changes from other contributors first update your local repository:

```
$ git pull origin docs
```

6.2.2 Edit or add files

If you now edit the files in the `sources` folder or add a new file you might want to include this into the table of contents. To do so you have to add the new file(s) without their file extension to `index.rst` in the section starting with `.. toctree::`.

After editing the files in `docs/sources` you can inspect the result of your changes: First `cd` to the `docs` folder and run:

```
$ make clean
$ make html
$ open build/html/index.html
```

Your browser should now show the current version of the documentation web site.

6.2.3 Publish your changes

To submit your changes first run `git status` to get an overview of changed and new files.

Then execute

```
$ git add PLACE_A_FILENAME_HERE
```

for all the files you added or changed. Then run

```
$ git commit -a -m "PLACE A MESSAGE HERE DESCRIBING YOUR CHANGES"
$ git push origin docs
```

After a few seconds you should see the changes published on https://whdp.readthedocs.io.

CHAPTER 7

Obsolete documentation

This material should be worked in to the other sections.

7.1 Run server in test mode

The following sequence initializes `whdp` and runs the server in single process mode.

```
$ rm -rf ./lz 2>/dev/null

$ export ETC=./etc
$ rm -rf $ETC 2>/dev/null

$ pool init-config --use-sqlitedb ./lz
$ pool init-db
$ pool check-config
$ pool run-simple-server
```

Usually `pool init-config` would write to `/etc/whdp` and thus the command requires `root` privileges. Setting the environment variable `ETC` allows overriding the `/etc` folder so we do not interfere with a global setup.

Further we use `--use-sqlitedb` so configuration and setup of a data base system as Postgres is not required. This flag is introduced for testing, in operational mode we recommond to avoid this flag and configer Postgres instead.

The last `run-simple-server` command will observe changes to the operational landing zone at *./lz* and report its operations. The command does not run in the background and thus will block the terminal until the user presses `CTRL-C` to enforce shutdown.

As a data provider we open another terminal window, setup a development landing zone and commit the defaults to the operational landing zone. You should then see some output from the `run-simple-server` command in the previous terminal window:

```
$ rm -rf ./dlz 2>/dev/null
$ export ETC=./etc
```

(continues on next page)

(continued from previous page)

```
$ pool start-develop dlz
$ pool check dlz
$ pool update-operational dlz
```

7.2 Workflow example

To initialize `whdp` configuration on the current server run the `init-config` subcommand, this might require admin permissions because the config file is stored in the `/etc/whdp` folder:

```
$ pool init-config ./lz

> init-config
- guess settings
  - 'matlab' not found on $PATH
- created config files at /etc/whdp
  please edit these files and adapt the data base configuration to your setup
+ initialized landing zone at ./lz
```

Then edit this file and run `pool check-config`:

```
$ pool check-config

> check-config
- check settings in config file /etc/whdp/whdp.ini
- try to connect to db
- could not connect to db postgresql://user:password@localhost:5432/whdp
- check R configuration + code execution
- matlab not configured, skip tests
- check julia configuration + code execution
- check julia version.
- check python configuration + code execution
+ all checks passed
```

To start development create a so called *development landing zone** which can be an arbitrary folder:

```
$ pool start-develop ./dlz

> start-develop
- setup development landing zone
- operational landing zone is empty. create development landing zone with example␣
↪files.
+ setup done
```

This copied some example `.yaml` files, conversion scripts and raw data files. To check the scripts run:

```
$ pool check-scripts ./dlz

> check-scripts
- check landing zone at ./dlz
- check ./dlz/data/sensor_from_company_xyz/sensor_instance_julia/conversion.jl
- wrote conversion result to /tmp/tmp9hcxslxv/sensor_instance_julia_0.csv
- wrote conversion result to /tmp/tmp9hcxslxv/sensor_instance_julia_0.txt
- check ./dlz/data/sensor_from_company_xyz/sensor_instance_python/conversion.py
- wrote conversion result to /tmp/tmp9hcxslxv/sensor_instance_python_0.csv
```

(continues on next page)

(continued from previous page)

```
- wrote conversion result to /tmp/tmp9hcxslxv/sensor_instance_python_0.txt
- check ./dlz/data/sensor_from_company_xyz/sensor_instance_r/conversion.r
- wrote conversion result to /tmp/tmp9hcxslxv/sensor_instance_r_0.csv
- wrote conversion result to /tmp/tmp9hcxslxv/sensor_instance_r_0.txt
+ congratulations: checks succeeded.
```

This checked the scripts and you can inspect the results files as displayed in the output.

To check the `.yaml` files:

```
$ pool check-yamls ./dlz/

> check-yamls
- check yamls in landing zone at ./dlz/
- setup fresh development db. productive does not exist or is empty.
- load and check 1 new yaml files:
- ./dlz/data/parameters.yaml
+ all yaml files checked
```

Now you can upload the changes from the development landing zone to the operational landing zone:

```
$ pool update-operational ./dlz

> update-operational
- check before copying files around.
- copied data/parameters.yaml
- copied data/sensor_from_company_xyz/sensor_instance_julia/conversion.jl
- copied data/sensor_from_company_xyz/sensor_instance_julia/raw_data/data-001.raw
- copied data/sensor_from_company_xyz/sensor_instance_matlab/raw_data/data-001.raw
- copied data/sensor_from_company_xyz/sensor_instance_python/conversion.py
- copied data/sensor_from_company_xyz/sensor_instance_python/raw_data/data-001.raw
- copied data/sensor_from_company_xyz/sensor_instance_r/conversion.r
- copied data/sensor_from_company_xyz/sensor_instance_r/raw_data/data-001.raw
- copied data/sensor_from_company_xyz/source_type.yaml
- copied sites/example_site/images/24G35_regenwetter.jpg
- copied sites/example_site/images/IMG_0312.JPG
- copied sites/example_site/images/IMG_0732.JPG
- copied sites/example_site/site.yaml
+ copied 13 files to ./lz
```

The datapool source code can be found here.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search