whdp Documentation

Release 0.1

Uwe Schmitt

Jul 24, 2019

Contents

As always, RTFM!

If you want to work with data stored in the WaterHubDataPool (WHDP), these instruction are essential. If you want to add data to the WHDP, read the *Instructions for data provider*. If you want to load data from the WHDP, read the *Instructions for data users*.

Content:

CHAPTER 1

Instructions for data users

A typical data user is an Eawag research staff member or student.

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 the WHDP

First-time use ~

1. Contact the WaterHub project coordinator to request access
2. Obtain the password via the WaterHub project coordinator
3. **Take note of the following information** Host: eng-whdp1.eawag.wroot.emp-eaw.ch Port: 22 Name of the database: whdp Default user name: whdp_user
4. Continue with steps as outlined below

1.1.1 Connect with terminal

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:

1.1.2 Connect within 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.

1.1.3 Connect within Python

Different options exist, psycopg2 is widely 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:

1. For online sensor data:
   a. The raw data arrive in the landing-zone at the right place,
   b. A conversion script is provided to interpret the raw data, and
   c. Meta-data are provided.
2. For offline laboratory data:
   a. The raw data is copied to the whdp server
   b. A conversion script is available and running to process the data into the lab_results.csv file
   c. 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 WHDP provides a command to create development landing-zones. A development landing-zone can have any names, but let's call it `dlz` for now:

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

```
$ whdp 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.

```
$ whdp check dlz
```

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

```
$ whdp 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 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 site
- Unit: A unique name for the room/space (e.g., BU_B09, FC_C24, NE_A017)
- Area: The section of the room/space floor plan (e.g., GW).
- Setup: The experimental setup (e.g., GW).
- Component: A part of the experimental setup (e.g., MBR-Tank1).
- Status: The current state of operation of the site (e.g., operational).
- 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: NE_WaterHub_GW_GW_MBR-Tank1
description: MBR Tank 1 - Buffer tank
unit: NE_A17
area: GW
setup: GW
component: MBR-Tank1
status: operational

# pictures are optional:
pictures:
    -
      path: images/installation.png
      description: Installation of Lora Ultrasonic Sensor 00769
```

(continues on next page)

(continued from previous page)

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

Adding a site if fairly trivial by starting an interactive input program. Typing a dot (.) at any time will abort the entry:

```
$ whdp add dlz site
```

2.3.1 Semi-automatic site updates

The WHDP-provider-tool-kit (https://gitlab.com/krisvillez/whdp-provider-tool-kit) includes a script dlz_site.py to automatically generate all sites in the dlz. This script reads an Excel file (whdp_table_site.xlsx) to obtain the list of all sites.

2.3.2 Note on sampling for laboratory data analysis

- If the added site is a location where laboratory samples can be taken, it is likely necessary to modify the laboratory workflow accordingly (E.g. programmable drop-down menus in Excel sheets).
- The laboratory data conversion script is programmed so that adding a site does not require modification of this script.

2.4 Add or modify variables

The file `variable.yaml` is stored in the `data` folder and contains all variables recognized within the WHDP. This file is modified to add new variables. The information to be included are:

- Name: Shorthand name of the variable (e.g., sensor_bp)
- Description: Additional description of the variable. In case there is no description required, the field can remain empty.
- Unit: Specifies the measurement unit of the variable

Within `variable.yaml`, each variable is described by 4 text lines with the following structure (note the dash in the first line):

```
-
  name: sensor_bp
  unit: mbar
  description: pressure
```

To modify `variable.yaml` with the nano editor, type the following in the command line:

```
$ nano dlz/data/variables.yaml
```

2.4.1 Semi-automatic variable updates

The WHDP-provider-tool-kit (https://gitlab.com/krisvillez/whdp-provider-tool-kit) includes a script dlz_variable.py to automatically generate all variables in the dlz. This script reads an Excel file (whdp_table_variable.xlsx) to obtain the list of all variables.

2.4.2 Note on name convention

In conventional WHDP practice, each variable name is composed of the following parts separated by an underscore (_):

- An actuator/sensor/lab specification (e.g., 'actuator' for valve state; 'sensor' for pressure measurement; 'lab' for laboratory measurement)
- A variable description, according to the EN_81346 norm (see https://de.wikipedia.org/wiki/EN_81346), e.g. sensor_bp for pressure measurement
- [optional] An additional variable descriptor when the EN_81346 norm is too vague, e.g.: sensor_bq_doconc for dissolved oxygen concentration

2.4.3 Note on laboratory data

- If the added variable is a new laboratory measurement, it is likely necessary to modify the laboratory workflow accordingly (E.g. programmable drop-down menus in Excel sheets).
- Adding a variable requires modification of the laboratory data conversion script .

2.5 Add or modify persons (offline laboratory data)

The file `persons.yaml` is stored in the `/dlz/lab_data` folder and contains all current and previous staff members recognized within the WHDP. The information to be included are:

- Abbrev: Unique shorthand identifier of the person (e.g., villezkr)
- Name: Full name of the staff member (e.g., Kris Villez)
- Department: Affiliation of the staff member (e.g., eawag-eng)

Within `persons.yaml`, each variable is described by 4 text lines with the following structure (note the dash in the first line):

```
-
    abbrev: villezkr
    name: Kris Villez
    department: eawag-eng
```

To modify `persons.yaml` with the nano editor, type the following in the command line:

```
$ nano dlz/lab_data/persons.yaml
```

2.5.1 Semi-automatic person updates

The WHDP-provider-tool-kit (https://gitlab.com/krisvillez/whdp-provider-tool-kit) includes a script dlz_person.py to automatically generate all persons in the dlz. This script reads the sheet *person* in an Excel file (last known: LabSamples2019.xlsx) to obtain the list of all persons.

2.6 Conversion of raw data (offline laboratory data)

2.6.1 Standardized file format

The data arriving in the landing zone are called *raw data*. The raw data must be converted into a so called *standardized file* by a conversion script. The format of the standardized files is defined below. In the WHDP current practice, a single Python script is used to collect all laboratory data and process it into a single file 'lab_results.csv'. This file is overwritten every day. The WHDP takes this file in and also overwrites existing entries when results are changed. In normal operation, data entries in the WHDP can be overwritten but cannot removed.

2.6.2 Standardized file format for lab_results.csv

The standardized file format for the input data is a `csv` file with 4 columns. It must adhere the following standards:

- File format: `csv` file with semicolon as delimiter (`;`)
- Columns: The first row contains the column names. The columns are always:
  - lab_identifier
  - sample_identifier: must be unique
  - variable: must exist in `variables.yaml` and have the exact same name (see above)
  - filter_lab
  - dilution_lab [can be empty]
  - method_lab
  - value_lab: must contain only numerical values. Missing values (`NULL`, `NA`, or similar) are not allowed.
  - description_lab [can be empty]
  - person_abbrev_lab
  - timestamp_start_lab [can be empty]: format `yyyy-mm-dd hh:mm:ss`
  - timestamp_end_lab: format `yyyy-mm-dd hh:mm:ss`
  - site: must be defined as such and have the exact same name (see above).
  - person_abbrev_sample: must be defined in `persons.yaml` and have the exact same value for abbrev (see above).
  - filter_sample
  - dilution_sample [can be empty]
  - timestamp_sample: format `yyyy-mm-dd hh:mm:ss`
  - description_sample [can be empty]
  - method_sample [can be empty]

Example:

lab_identifier	sample_identifier	variable	filter_lab	dilution_lab	method_lab	value_lab	description_lab	person_abbrev_lab	timestamp_start_lab	timestamp_end_lab	site	person_abbrev_sample	filter_sample	dilution_sample	timestamp_sample	description_sample	manual
NE19063	NE19063_lab_1	TOC	None		Shimadzu_TOC-L	1.39		villezkr	2019-02-06 23:59:59		NE_WaterHub-Test	hessange	GF/PET 0.45/25		2019-02-06_17:15:00	Batch-D	manual
NE19054	NE19054_lab_1	NO3N	None		IC_Anionen	14.9		villezkr	2019-02-07 23:59:59		NE_WaterHub_GW_ML2	maranfab	None		2019-02-07_10:00:00		manual
NE19055	NE19055_lab_1	NO3N	None		IC_Anionen	10.1		villezkr	2019-02-07 23:59:59		NE_WaterHub_GW_ML3	maranfab	None		2019-02-07_10:00:00		manual
NE19054	NE19054_lab_1	PO4P	None		IC_Anionen	1.5		villezkr	2019-02-07 23:59:59		NE_WaterHub_GW_ML2	maranfab	None		2019-02-07_10:00:00		manual

2.6.3 Conversion script

Unlike the case for online sensor data, the conversion script is not executed automatically by the WHDP software. Instead the data provider must execute this conversion script manually or set up a time-based execution schedule (with *cron*).

2.6.4 Note on automated data transfer and processing

- To automate this process, the following scripts are currently in use
  - /home/whdp-provider/laboratory/t_lab_central.py (copy *.xlsx files from Q drive)
  - /home/whdp-provider/laboratory/t_lab_project.py (copy *.xlsx files from Q drive)
  - /home/whdp-provider/laboratory/c_lab.py (convert *.xlsx files to *.csv format)
- The above scripts are executed once per day by means of *cron*, a time-based job scheduler

2.7 Add a new source_type (online sensor data)

In order to add a new signal source_type, the information about this source_type have to be provided as a `source_type.yaml` file in a new folder for the source_type within the `dlz/data` folder of the landingzone. The information to be specified are:cd ..

- Name: unique shorthand name, e.g. obtained by combining the product name and manufacturer (e.g. concube_scan)
- Description: Detailed description

Adding a source_type if fairly trivial by starting an interactive input program. Typing a dot (.) at any time will abort the entry:

```
$ whdp add dlz source_type
```

2.7.1 Note on programmable logic controllers (PLCs)

In conventional WHDP practice, each programmable logic controller (PLC) is considered a source_type of its own for the following reasons:

- The conversion scripts for devices (source) of the same type (source_type) are expected to be similar. Because every PLC is programmed very differently, one cannot expect the conversion scripts to be very similar, at least not to the same degree that one expects for stand-alone sensors.
- The use of a source_type to describe a single PLC device allows to provide a conversion script specific to the PLC which can be used to assign the data automatically to their sources (actuators/sensors) associated with the given PLC.

2.8 Add a source (online sensor data, instance of source_type)

In order to add a new signal source, the information about this source have to be provided as a `source.yaml` file in a new folder for the source_type within the `dlz/data` folder of the landingzone. The information to be specified are:

- Name: unique shorthand name for the source (e.g., concube_grey1, prgB615)
- Description: detailed description
- Serial: [optional] serial number
- Manufacturer: [optional] the manufacturer of the device

Adding a source_type if fairly trivial by starting an interactive input program. Typing a dot (.) at any time will abort the entry:

```
$ whdp add dlz source
```

2.8.1 Semi-automatic source updates

To avoid errors, the WHDP-provider-tool-kit (https://gitlab.com/krisvillez/whdp-provider-tool-kit) includes scripts to automatically generate all sources of a certain source_type in the dlz. This script reads the data from an Excel file which lists the available source_types. Currently, the following scripts and associated Excel sheets are available:

script	file
dlz_sources_memographgrey.py	memographgrey_config.xlsx
dlz_sources_plcgrey.py	plcgrey_config.xlsx

2.8.2 Note on programmable logic controllers (PLCs)

- In conventional WHDP practice, each actuator and sensor associated with the same programmable logic controller (PLC) is considered a separate source. A single source can be associated with multiple variables however. E.g., continuous-scale sensors (e.g., pH) typically have a status variable (manual on/manual off/auto on/auto off) and an actual measured value.

- In conventional WHDP practice, the name for a source representing a PLC-connected actuator or sensor is equal to name of the PLC program that registers the state (and value) of the actuator (sensor). E.g.: prgB615

2.9 Add raw data to existing source (online sensor data)

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 rare 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.

2.9.1 Note on automated data transfers

- To automate this process, the following scripts are currently in use
  - t_plcgrey.py (data transfer from plcgrey PLC)
- The above script are executed once per day by means of *cron*, a time-based job scheduler
- The conversion script for data transfered from plcgrey PLC in use is located here:
  - /home/whdp-provider/dlz/data/plcgreyconversion.py

2.10 Conversion of raw data (online sensor 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. Most typically, every `source` needs an individually adapted conversion script. As an alternative, the WHDP also allows an individually adapted conversion script for a `source_type`

2.10.1 Standardized file format for association with a source

The standardized file format for the input data is a `csv` file with 4 columns. It must adhere the following standards:

- File format: `csv` file with semicolon as delimiter (`;`)
- Timestamp format: `yyyy-mm-dd hh:mm:ss`
- Column names: The first row contains the column names. The columns are always: `timestamp`, `variable`, `value`, and `site`. The variable must exist in `variables.yaml` and have the exact same name (see above). The `site` must be defined as such and have the exact same name (see above).
- `value` column: Must contain only numerical values. Missing values (`NULL`, `NA`, or similar) are not allowed.

Example:

timestamp	variable	value	site
2019-01-29 12:29:29	sensor_bp	18.605044	NE_WaterHub_GW_GW_M1
2019-01-29 12:29:30	sensor_bp	19.225162	NE_WaterHub_GW_GW_M1
2019-01-29 12:29:31	sensor_bp	19.535282	NE_WaterHub_GW_GW_M1
...	...	...	...

2.10.2 Standardized file format for association with a source_type

The standardized file format for the input data is a `csv` file with either 5 columns. It must adhere the following standards:

- File format: `csv` file with semicolon as delimiter (`;`)
- Timestamp format: `yyyy-mm-dd hh:mm:ss`
- Column names: The first row contains the column names. The columns are always: `timestamp`, `variable`, `value`, `site`, and `source`. The variable must exist in `variables.yaml` and have the exact same name (see above). The `site` must be defined as such and have the exact same name (see above). The `source` must be defined as an instance of the considered `source_type` and have the exact same name (see above).
- `value` column: Must contain only numerical values. Missing values (`NULL`, `NA`, or similar) are not allowed.

Example:

timestamp	variable	value	site	source
2019-01-29 12:29:29	sensor_bp	18.605044	NE_WaterHub_GW_GW_M1	prgB615
2019-01-29 12:29:30	sensor_bp	19.225162	NE_WaterHub_GW_GW_M1	prgB615
2019-01-29 12:29:31	sensor_bp	19.535282	NE_WaterHub_GW_GW_M1	prgB615
...	...	...	...	...

2.10.3 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 following points should be considered when writing an conversion script:

- Indicate corrupt input data by throwing an exception within a conversion script. An informative error message is helpful and will be logged.
- If a conversion script writes to `stdout` (i.e. normal `print()` commands) this may not appear in the WHDP 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.

2.10.4 R

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

```
# 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=";")

}
```

2.10.5 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
```

2.10.6 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)
```

2.10.7 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 permissions 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 WHDP

```
$ 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. Initialize 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 WHDP:

```
$ mkdir -p /nfsmount/landing_zone
$ whdp 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 selected 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:

```
$ whdp 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.

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 Design principles

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 description 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

4.2 signal

This is the central table holding the measurements generated by online sensors. Each row represents a *value* of a measured variable at a given *time* and location (*site*).

Note: In the original UWO datapool, it is so that *the coordinates of the signal may not correlate to entries in the site table*. In the WHDP this is not a concern. For this reason, it is sufficient to link the online sensor measurement to an instance of site without additional coordinates.

field	datatype	description
signal_id	integer (pk)	
value	float	the actual measured value of the variable
timestamp	date_time	time when the value was measured.
source_id	integer (fk)	source
site_id	integer (fk)	site
variable_id	integer (fk)	variable

4.3 lab_results

This is the central table holding the measurements generated by offline laboratory measurements. Each row represents a *value* of a measured variable corresponding to a given sampling time (*timestamp_sample*) and location (*site*).

field	datatype	description
lab_result_id	integer (pk)	automatically generated during upload
lab_identifier	string (uq)	string identifying the measurement
sample_identifier	string	string identifying the sample
variable_id	integer (fk)	unique id of variable description
filter_lab	string	applied filter during analysis
dilution_lab	float	applied dilution during analysis (>= 1)
method_lab	string	analytic method
value_lab	float	obtained measurement
description_lab	string	comment about the analysis
person_id_lab	integer (fk)	unique id of person executing the analysis
timestamp_start_lab	date_time	time of first analysis of this sample
timestamp_end_lab	date_time	time of last analysis of this sample
site_id	integer (fk)	unique id of site where sample was taken
person_id_sample	integer (fk)	unique id of person taking the sample
filter_sample	string	applied filter during sampling
dilution_sample	string	applied dilution during sampling (>=1)
timestamp_sample	date_time	time of sampling
description_sample	string	comment about the sample

4.4 site

A *site* is a specific location where measurements are made or samples are taken. At a given site, several measuring devices (*source*) can be found. The location of the site is described by a 4-level hierarchy including *unit* (=room, e.g. NE_A17), *area* (e.g., GW), *setup* (e.g., GW), and *component* (e.g., MBR-Tank1). Together, these four elements are combined to provide a unique *name* (e.g., NE_WaterHub_GW_GW_MBR-Tank1) for the *site*.

field	datatype	description
site_id	integer (pk)	
name	string (uq)	name of that site
description	string	
unit	string	room
area	string	room section
setup	string	experimental system (same controller)
component	string	subsystem of the experimental system
status	string	status of the subsystem

4.5 picture

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

4.6 source

Description of an online data-generating device. A (data-) source is a specific measuring equipment, i.e. an instance of the *source_type* class. Every measurement (signal) is produced by a source. The name of a source must be unique.

Note: For devices that change location frequently, it may be best to not list the *site_id* in this table and only specify *site_id* in the signal table.

field	datatype	description
source_id	integer (pk)	
source_type_id	integer (fk)	source category
name	string (uq)	short name for device, e.g. plc_grey_v1.0, concube_grey1)
description	string	description of device (e.g., S::CAN con::cube)
serial	string	serial number (unique, if available)
manufacturer	string	company which produced that equipment

4.7 source_type

Categorization of a given source.

field	datatype	description
source_type_id	integer (pk)	
name	string (uq)	short name for device type (e.g., plc_grey)
description	string	device type description (e.g., WAGO programmable logic controller)

4.8 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 categorical value
numerical_value	float	the numeric value it is mapped to

4.9 variables

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

field	datatype	description
variable_id	integer (pk)	
name	string (uq)	short name for variable (e.g. lab_cod, actuator_bf, sensor_bq_cond)
description	string	explanation of measurement; include reference to SOP(s) where available
unit	string	unit of measurement (e.g. "m3 h-1"); use *no unit* when the variable is not a continuous scale (e.g., pump on/off); use *1* for dimensionless variables (e.g., pH)

4.10 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.

Note: Note that current implementation uses an associative table to link each comment (*comment_id*) to a signal (*signal_id*). This allows many-to-one associations (multiple comments for same signal by different people) as well as one-to-many associations (one comment for multiple measurements).

field	datatype	description
comment_id	integer (pk)	
signal_id	integer (fk)	(via association table)
text	string	the comment itself
timestamp	date_time	the time the comment was added
person_id	integer (fk)	identified of the author who added the annotation

4.11 signal_quality

An online sensor measurement may contain more than one quality flag (but not the same quality flag twice). The combination of *signal_id*, *quality_id*, and *person_id* must be unique.

Note: Note that current implementation uses an associative table to link each label (*signal_quality_id*) to a signal (*signal_id*). This allows many-to-one associations (multiple labels for same signal by different people) as well as one-to-many associations (one label for multiple measurements).

field	datatype	description
signal_quality_id	integer (pk)	
quality_definition_id	integer (fk)	
signal_id	integer (fk)	(via association table)
timestamp	date_time	date when annotation was added

4.12 quality

Measurements contain errors. This table holds the controlled vocabulary mentioned above. As some quality flags may be assigned automatically, the *method* field indicates the origin of such a 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.
person_id	integer (fk)	author who added the annotation

4.13 persons

Information about the staff involved in sampling, laboratory analysis, data quality checks, and commenting.

field	datatype	description
person_id	integer (pk)	
abbrev	string (uq)	unique identifier for the staff member (alias)
name	string	name of staff member
department	string	department acronym

4.14 project

Information about projects.

Note: This is considered legacy code obtained from the original UWO datapool software. Since the entries in this table are not linked anywhere, filling this table has lowest priority.

field	datatype	description
project_id	integer (pk)	
abbrev	string (uq)	unique identifier for the project
title	string	project title
description	string	project goals

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 WHDP source code can be found here.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search