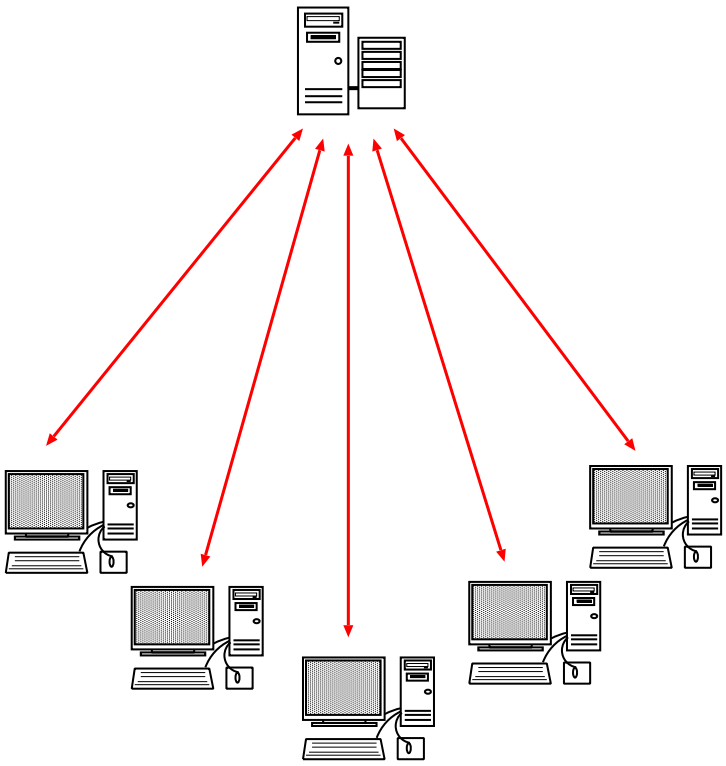
Wakasa Database Administrator's Guide

Arto Teräs ⟨teras@wis.ec.t.kanazawa-u.ac.jp⟩

January 6, 2004



Contents

1 Introduction

This manual documents the tools to insert data into and remove data from the Wakasa database, which is used for storing weather related observation data, especially snowfall data. Other administrative tasks such as taking backups are covered briefly.

This guide is intended for the database administrator. For using the plotting tools, refer to the separate User's Guide. There is also a Developer's Guide which explains the structure and source code of the current tools on a technical level and gives advice on further development, for example adding new instruments and developing analysis programs which access data in the base directly.

2 System requirements

2.1 Hardware

The system should work on basically any hardware which runs a Unix-like operating system. A standard PC is probably the most economical in most cases. The exact hardware requirements depend heavily on amount of data, number of users, usage patterns and required speed. Also, it is possible to install the database engine and the plotting tools either on the same computer or on separate computers. The database performance depends mainly on the amount of RAM and hard disk speed (especially in the case of many simultaneous users), the plotting tools are more CPU intensive. An example configuration for moderate use is shown below.

- 1 GHz processor or better
- 512 MB RAM
- 40 GB hard disk
- 100 Mbit/s network card

2.2 Required software

The system has been tested only on the GNU/Linux operating system (Debian GNU/Linux and Vine Linux distributions) but should work on any Unix-like operating system. The necessary applications are also available for Windows, but running on Windows would probably require some minor modifications to the code. The required applications are listed below.

- PostgreSQL database engine, version 7.2.1 or newer.
  Home page: `http://www.postgresql.org`
- Python programming language, version 2.1 or newer.
  Home page: `http://www.python.org`
- The following additional Python modules:
  - pyPgSQL module version 2.3 or newer.
    Home page: `http://pypgsql.sourceforge.net/`

  - Gnuplot module version 1.5 or newer.
    Home page: `http://gnuplot-py.sourceforge.net/`
  - Mxdatetime module version 2.0.3 or newer.
    Home page: `http://www.egenix.com/files/python/mxDateTime.html`
  - Numeric module version 21.0 or newer, including support for masked arrays[1].
    Home page: `http://www.pfdubois.com/numpy/`

- Gnuplot 3.8i or newer[2].
  Home page: `http://www.gnuplot.info`

Python and Gnuplot should be installed on the same computer, the database engine can be installed on a separate computer if desired. All the components are open source software.

2.3 Optional software

Optional software includes various graphical database browsers and management tools. The following tools were used and found useful during the development of the system:

- Pgaccess, version 0.98.8 or newer recommended.
  Home page: `http://www.pgaccess.org/`
- Phppgadmin, version 2.4 or newer.
  Home page: `http://phppgadmin.sourceforge.net/`
- PostgreSQL AutoDoc, a tool to generate documentation automatically from the database structure and comments.
  Home page: `http://www.rbt.ca/autodoc/`

The administrator might also install additional programming languages (including database connectivity modules) and analysis tools which the scientists prefer to use, but that doesn't belong to the scope of this manual.

3 Database management

This manual does not aim to be a complete guide to PostgreSQL database administration. It only describes certain main points which must be considered during installation and Wakasa database specific issues. For more information, the reader is adviced to refer to the PostgreSQL documentation, which is also available online [1].

[1] The standard package of Numeric includes masked arrays (MA) support, but at least in Debian GNU/Linux it has been moved to a separate package called python-numeric-ext.

[2] At the time of writing, the latest stable release of Gnuplot is version 3.7, which does not work with all Wakasa plots. Version 3.8 is the development version.

3.1 Installation and configuration

The basic installation of the database engine is not covered by this manual, please follow the instructions provided by PostgreSQL or the operating system distributor when using their packages (for example rpm packages in most GNU/Linux distributions). However, the following things should be noted in the installation.

1. Using configuration file `postmaster.conf` (default location `/etc/postgres/postmaster.conf`) it is possible to adjust where the actual data files are stored. The Wakasa database can easily require several gigabytes of disk space, which must be taken into account when installing. The data location is also the most critical considering performance, so if the system contains several hard disks the fastest disk should be used for storing the data.

2. The default configuration of PostgreSQL does not allow connections from the network. To enable network connections, file `pg_hba.conf` (default location `/etc/postgres/pg_hba.conf`) should be changed accordingly. For example the following line enables password authenticated connections[3] via tcp/ip ports for the ip address range `192.168.0.1 - 192.168.0.254`:

   ```
   host all 192.168.0.0 255.255.255.0 md5
   ```

   Refer to PostgreSQL documentation for details. Note that if both the database and the plotting tools are installed on the same computer and the users log in for example using ssh, network connections to the database are not necessary[4].

3.2 User accounts

Database user accounts are separate from local Unix user accounts on the database server. Having a Unix user account doesn't automatically give permission to use the database, on the other hand it is perfectly possible (and a very common configuration) to grant access to the database for people who don't have a local user account on the server. There may be also several logical databases on one server running only one database engine, for example one database for snowfall measurements and another for personnel management.

It is possible to adjust the permissions in detail on a per table and per user basis. For example, it would be possible to open access to some measurements and restrict access to others. This is usually not needed in a scientific environment. However, the permissions can be set so that only administrators have write permission and other users can only read the data, to prevent accidentally modifying or deleting values. Permissions are given using the SQL command GRANT and revoked using the command REVOKE. For detail, refer to the PostgreSQL documentation.

3.3 Creating initial tables

Before creating the tables, a database must be created. For example, to create a database called `wakasa` you can type "`createdb wakasa`" from the command line or use the `psql` tool and SQL language

[3] This configuration uses passwords with md5 encryption, they are never passed as plain text through the network.

[4] The plotting tools developed in the Wakasa project can connect via a local Unix socket and do not need a tcp/ip port when used on the same computer than the database. However, some graphical management tools such as Phppgadmin connect using tcp/ip even when used locally.

command "`CREATE DATABASE wakasa`"[5].

To create the table structure, two scripts are provided. There are two versions of the database, standard version and array version. The array version uses a SQL extension of PostgreSQL database engine called arrays, which allows storing multiple values in one table cell as an array but still selecting individual elements or parts of the array in retrieval. Differences between the two versions are described in more detail in the Wakasa Database User's Guide. Script `create-tables.postgresql` creates the table structure for the standard (no arrays) version, `create-tables-with-arrays.postgresql` creates the structure for the array version. The array version is recommended because of better performance. See the User's Guide for more detailed description about the differences between the two versions.

The usage of both scripts is similar and relies on the `psql` tool. The user giving the command must be a database superuser, normally the `postgres` user. For example, to create the array version table structure in database `wakasa`, give the following command:

```
psql -d wakasa -f create-tables-with-arrays.postgresql
```

After the tables are created, by default the only user who has permission to access them is the user who created them. Permissions have to be granted to other users who want to access the tables. A simple configuration is to grant read-only access to all users who have an user account in the database. For example, to grant read only access for everybody to table `balance`, connect to the database using `psql` and give the following SQL command:

```
GRANT SELECT on balance to PUBLIC;
```

Unfortunately PostgreSQL doesn't seem to have a simple method for granting access rights to all tables using one command. However, it can be done using a command line trick [2]:

```
psql -t -A -c "SELECT tablename FROM pg_tables WHERE tablename NOT LIKE 'pg_%'"
<database name> | xargs -i psql -c "<grant command>" -d <database name>
```

For example, to grant read-only access to all users for database `wakasa`, the command line would be (everything on one line):

```
psql -t -A -c "SELECT tablename FROM pg_tables WHERE tablename NOT LIKE 'pg_%'"
wakasa | xargs -i psql -c "GRANT SELECT on {} to PUBLIC" -d wakasa
```

The abovementioned trick does not give permissions to number sequences which are also part of the database (used for instance to generate the parameter set id numbers). They are not needed for reading existing data, but required when inserting new data. The sequence names can be queried from the system view `pg_statio_user_sequences`. To grant enough permissions for user `assistant` to insert new data into database `wakasa`, use the following command lines:

```
psql -t -A -c "SELECT tablename FROM pg_tables WHERE tablename NOT LIKE
'pg_%'" wakasa | xargs -i psql -c "GRANT SELECT,INSERT,UPDATE,DELETE on {}
to assistant" -d wakasa
```

```
psql -t -A -c "SELECT tablename FROM pg_statio_user_sequences" wakasa | xargs
-i psql -c "GRANT SELECT,INSERT,UPDATE,DELETE on {} to assistant" -d wakasa
```

[5]The `psql` tool requires a valid database name when connecting to the PostgreSQL engine. In an empty installation, two special databases exist: `template0` and `template1`. Use one of these for the first connection (type "`psql -d template0`"). After the `wakasa` database is created that can naturally be used.

Another possibility is to take a look at small helper scripts `grant-readonly-noarrays.postgresql` and `grant-readonly-arrays.postgresql`. They can be used to grant read permissions (including the sequences) and customized to help in other permission commands. Of course it is always also possible to write all the necessary table and sequence names by hand when using the GRANT command. The current permissions for all tables and sequences can be listed by connecting to the database using the `psql` tool and typing `\dp`.

3.4 Backups

Taking backups from data stored in a database needs some special attention. The database engine uses internal buffering of data into memory and keeps a log of transactions which are written to the disk later. Therefore simply copying the database engine data directory is not guaranteed to give a working backup of the base, if the engine is running.

It is possible to get a backup by first shutting down the database engine process and then copying the directory. However, practically all database engines provide another method which can be used also when the database is running, called SQL dump. The dumps can not only be used to restore a backup but also to transfer the data to another computer, to a newer version of the database engine or possibly even to a database engine from a different manufacturer.

The tool to take database dumps in PostgreSQL is called `pg_dump`. There are many options which might be worth exploring, only one solution is presented below. A dump of database wakasa to file `/backups/wakasa-dump.tar` in tar format can be created using the following command line:

```
pg_dump -Ft -b wakasa >/backups/wakasa-dump.tar
```

The table structure, permission information and the data are all included in the dump. The tool to restore the data from the dump is called `pg_restore`. Before calling `pg_restore`, it is necessary to create the database using `psql` or `createdb` command[6]. However, it is not necessary to create the table structure, it will be restored from the dump file. Restoring the data must be done as a superuser, usually user `postgres`.

The command to restore the dump is

```
pg_restore -d wakasa -Ft /backups/wakasa-dump.tar
```

This will restore all information, including the access permissions. In particular, it requires that all the user names who have been granted permissions exist in the target system. Therefore this is useful mainly when using the dump as a backup, for example reloading the data after a hard disk failure. When moving the data to another computer (for example when sharing a copy with another laboratory), it is usually better to ignore the access permission data. In this case, use the following command line:

```
pg_restore -d wakasa -Ft -O --no-privileges /backups/wakasa-dump.tar
```

After restoring the tables and data without access privileges, it it necessary to grant access to desired users as described in section 3.2.

[6]Normally, the easiest way is to create the database by typing "`createdb wakasa`". This worked fine on the database server running PostgreSQL version 7.2.3, however, on another computer running version 7.2.1 restoring the data failed. The solution was to first connect to the database engine using `psql` (for example using command "`psql -d template0`") and then create the database using the SQL command "`CREATE DATABASE wakasa WITH TEMPLATE = template0;`" before giving the restore command.

4 Inserting data into the database

One of the main parts of the administration is to insert new observation data in the database. It can be mostly automatized but requires some manual input to get time stamps and instrument parameters saved correctly. Therefore it is important to understand how the data is stored in the base and be able to verify that the data has been inserted correctly.

4.1 Understanding the table structure

The table structure is explained in the Wakasa Database User's Guide, which also contains a reference section at the end explaining the purpose of all the fields in every table. From the point of view of the administrator, the most important is to understand the division between tables which contain measurement data and tables which contain instrument parameters. Figure 1 gives an example how data is stored.

Weatherstation_parameters

paramset_id	name	instrument_description	location_latitude	other columns
1	Kanazawa-jwa	JWA Mamedas KADE C-WT	NULL	. . .
2	Fukui2003-aws	AWS with rain gauge	36.14	. . .
3	Fukui2003-air	Airport station, on the roof	36.14	. . .

Weatherstation

time_utc	paramset	temperature	humidity	other columns
2003-01-28 03:00:30	1	4.55	66.1	. . .
2003-01-28 03:01:00	1	4.55	NULL	. . .
2003-01-28 03:01:00	2	4.45	66.7	. . .

Figure 1: Weather station data in the database

When inserting new data in the database, measurement data is read from the text files produced by the instruments. On the other hand, parameter set data is filled in manually by the database administrator. This is done by creating a device parameter file which contains the necessary information. Device parameter files are described in section 4.2.

In the example of figure 1, there are three weather stations, one located in Kanazawa and two in Fukui, represented by parameter set id's 1, 2 and 3, respectively. For each parameter set, there is also a more descriptive name: `Kanazawa-jwa`, `Fukui2003-aws` and `Fukui2003-air`. These names have been chosen by the administrator when feeding the data in. The `location_latitude` field for the `Kanazawa-jwa` parameter set is empty, because the data was not available when the parameters were written to the database. In fact, most of the fields are allowed to be empty when the data is inserted, and the values can be manually added later by using a database editor.

However, in a few cases there are obligatory parameters which cannot be left empty when inserting the data. For example, the `box_area` parameter for the electric balance is required for calculating the

precipitation rate. These parameter values also should not be edited later or then also the data should be edited accordingly.

4.2 Device parameter files

The device parameter files are used by the `insertdata.py` data insertion script, described in section 4.4. They are normal text files and can be created and edited using any text editor.

The syntax of parameter files is similar to the Windows `.ini` file syntax. Section names are in brackets, and each section contains name-value pairs, where the name and value are separated using either a colon or an equal sign. Lines beginning with a hash are treated as comments. There are two types of parameter files divided by the section structure, called "simple type" and "complex type" in this manual.

4.2.1 Parameter files, simple type

In a normal simple parameter file, there is only one section. The name of the section is taken as the name of the parameter set when storing the data into the database parameters table. The name-value pairs must correspond to the parameter table column names in the database. If no value is provided for a column, the default value (usually NULL) is used. An example parameter file for the POSS bistatic radar (`poss.ini`) is given below:

```
# Parameter file for POSS data from Fukui Jan-Feb 2003
#
[wakasa2003]
location_latitude: 36.14
location_longitude: 136.22
location_utc_offset: +9
time_resolution: 60
```

4.2.2 Parameter files, complex type

For some devices, one parameter set is not sufficient when reading a data file. Currently the only such device is the MRR radar. MRR can be programmed to automatically do measurements using several height resolutions. For example, it may be set to measure 30 seconds using 120 m height resolution, then the following 30 seconds using 60 m height resolution and following 30 seconds using 30 m height resolution, and then starting the cycle from beginning. Each of these height resolutions has it's own calibration spectra and other parameters, so they are treated as separate parameter sets in the program. Therefore a parameter file of the simple type is not sufficient.

In the "complex type" parameter files, there is an extra section called `Main`. The `Main` section contains a field `paramsets`, which lists the parameter sets currently in use. Note that it is possible to have more alternative parameter sets stored in the same file, and specify which ones are used by the `paramsets` field in the `Main` section. In the `Main` section, it is also possible to specify a default value for the reliability field, and maybe some additional values in the future. An example parameter file for the MRR radar (`mrr.ini`) is shown below.

```
# Parameter file for MRR data from Fukui Jan-Feb 2003
#
[Main]
paramsets: wakasa2003_60m, wakasa2003_120m, wakasa2003_200m
reliability:

[wakasa2003_60m]
location_latitude: 36.14
location_longitude: 136.22
location_elevation: 6
location_utc_offset: +9
time_resolution: 90
height_resolution: 60
height_steps: 30
calibration_spectra:
instrument_description: Metek MRR-2

[wakasa2003_120m]
location_latitude: 36.14
location_longitude: 136.22
location_elevation: 6
location_utc_offset: +9
time_resolution: 90
height_resolution: 120
height_steps: 30
calibration_spectra:
instrument_description: Metek MRR-2

[wakasa2003_200m]
location_latitude: 36.14
location_longitude: 136.22
location_elevation: 6
location_utc_offset: +9
time_resolution: 90
height_resolution: 200
height_steps: 30
calibration_spectra:
instrument_description: Metek MRR-2
```

It is possible to use this more complicated parameter file format for any device, but in most cases the simple type is enough.

In the case of MRR, the parser in `insertdata.py` automatically assigns measurements to correct parameters sets based on the height resolution information when reading the data file.

4.3 Configuration file insertdata.conf

The configuration file `insertdata.conf` is used to store common values such as the database location and the database user name which is used when inserting data. An example configuration file is given below.

```
# Configuration file for the insertdata.py script. Syntax similar as
# in Windows INI files, parsable using ConfigParser class in Python.
# Note that the entries are case sensitive.

[Database]
host: localhost
port: 5432
name: wakasa
username: teras
passwd:

[Postgresql]
use_arrays: 0
```

The configuration file can be modified using a normal text editor. There are two sections enclosed in brackets — `Database` and `Postgresql` — and a number of key-value pairs under the sections. Lines beginning with the '`#`' character are treated as comments. The key-value pairs are described in the following table.

Table 1: Configuration file entries

Database section	
host	Host name or IP address of the database server.
port	TCP/IP port number on the database server accepting connections from clients.
name	Name of the database to connect to. One database server can host multiple databases.
username	User name to use for the database connection. Note that this user must have write permission to the wakasa tables to be able to insert data.
password	Password for the database user account.
Postgresql section	
use_arrays	Specifies the Wakasa database version in use: 0 for standard version, 1 for the array version. This setting must match the table layout in the database, or inserting data will fail.

The `host` and `port` fields may also be empty. In that case, a local socket connection is used instead of a TCP connection. The local socket connection can normally be done without a password.

4.4 Data insertion script insertdata.py

The script `insertdata.py` can be used to insert data in the database. The syntax of the script is:

```
insertdata.py [-h] [-c conffile] [-d device] [-p paramfile]
              [-s starttime] [-e endtime] [-r reliability]
              [-u utc-offset] [-v] [--no-questions] inputfiles
```

The `inputfiles` argument is obligatory, at least one input file must be given. These are the files which contain the measurement data stored by instruments. Other command line parameters are described in the following table.

Table 2: Command line parameters of `insertdata.py`

Parameter		Description
`-c`	`--conffile=STRING`	Name of the file containing configuration parameters for the program, as described in section 4.3. If this parameter is not specified, the default filename `insertdata.conf` is used. Example: `-c insertdata-arrays.conf`
`-d`	`--device=STRING`	Type of the device which produced the input files given as arguments. This is currently an obligatory parameter, there is no automatic detection. See table 3 for the list of supported devices. Example: `-d poss`
`-e`	`--endtime=TIME`	Insert only data before the given time. If the input file has data with a later timestamp, that part of data is discarded. Syntax: YYYY-MM-DD HH:MM:SSZZZ, where ZZZ is the time zone information (ISO 8601 notation). Example: `-e "2003-01-27 21-02-50+09"`
`-h`	`--help`	Show the help screen.
	`--no-questions`	Skip all questions, even in verbose mode. If used in verbose mode, the `insertdata.py` scripts sometimes asks for a manual confirmation from the user before proceeding. Combined with this option, all the extra information of verbose mode is shown, but default answers are used for all questions.
`-p`	`--paramfile=STRING`	Name of a file containing configuration parameters for the device whose data is being inserted. See section 4.2 for the parameter file syntax. If this parameter is not given, the script will prompt the user to specify a parameter file before inserting the data. Example: `-p poss-fukui.ini`

(Continued on next page)

(Continued from previous page)

Parameter		Description
`-r`	`--reliability=VALUE`	Number describing the reliability of the data. This value will be stored in the reliability column of the measurement data table. Consult the local policy of the site about which values should be used. If the parameter is not given, NULL is inserted. Example: `-r 1`
`-s`	`--starttime`	Insert only data after the given time. Syntax: as endtime. Example: `-s "2003-01-27 21-02-50+09"`
`-u`	`--utc-offset=VALUE`	UTC offset of the timestamps in inputfiles. Source files of many devices don't contain this information explicitly, so it must be given manually by the administrator. Syntax: +XX or -XX, where XX is the offset in hours. If the offset is negative, please place the value in quotes to prevent it being interpreted as another command line parameter. Examples: `-u +09` (Japanese time), `--utc-offset="-05"`
`-v`	`--verbose`	Verbose output. This option can be given several times to reach the desired verbosity level. By default, the level is 0 and only errors are printed during processing. Level 1 prints some status information, shows the parameter set read from the device parameter file and asks for a confirmation before proceeding. Level 2 is useful for development and debugging, as all SQL queries and other detailed information is printed. Examples: `-v` (verbosity level 1), `-v -v` (verbosity level 2)

Instruments supported by `insertdata.py` are listed in the following table. The string in "Device name" column should be given to the script using the `-d` (`--device`) parameter.

Table 3: Instruments supported by `insertdata.py`

Instrument	Device name	Description
Electronic balance	balance	Precipitation rate measured using an electronic balance, data in the format produced by the logger of WIS lab [3].
Optical lidar	ceilo	Vaisala Ceilometer CT25K optical lidar, data in the format produced by the logger of WIS lab.
Heat sensor	heatsensor_yamada	Yamada heat capacity sensor, data in csv format as produced by the device.
MRR-2 radar (processed data)	mrr	Metek MRR-2 radar, processed data in the format produced by the device.

(Continued on next page)

(Continued from previous page)

Instrument	Device name	Description
MRR-2 radar (raw data)	mrrraw	Metek MRR-2 radar, raw data in the format produced by the device
POSS radar	poss	Andrew POSS bistatic radar, data in the format produced by the logger of WIS lab. The program accepts the `psXXXXX` files as input (files without `.fft` ending).
Radiometer WVR	radiometrics_wvr	Radiometrics WVR1100 radiometer, data in the `.los` files produced by the device.
Radiometer RPG	rpg_brt	Radiometer Physics RPG-LWP and RPG-TEMP90 radiometers, data in the binary format produced by the devices. The parser accepts the `.BRT`, `.MET` and `.OLC` files[7].
Vaisala soundings	radiosonde_vaisala	Vaisala radio sounding data, in the `.AED` and `.APA` files produced by the device. Both files are read by the program and must be in the same directory. However, it is sufficient to give only one of the files as an argument, other filenames are generated automatically.
Video based observation system	video	Video camera based observation system data, in the text format produced by Toru Shiina's program. The program processes three types of files, those ending in `.dsd`, `.dsf` and `vyd` which must all be in the same directory. However, it is sufficient to give only one of the files as an argument, other filenames are generated automatically.
Weatherstation AWS	weatherstation_aws	Data produced by the "AWS" automatic weather station, data in comma separated format (`.csv` files). The AWS can be installed in several configurations, this only accepts files which have data in the following order: temperature, humidity, wind speed, wind direction, precipitation gauge pulse.
Weatherstation JWA	weatherstation_jwa	Data produced by the "JWA" automatic weather station, in the format produced by the logger of WIS lab.

Usually, the user must specify at least the type of the device (`-d` parameter) and a parameter file (`-p` parameter). Often, it is also necessary to manually specify the time zone information (`-u` parameter). Two examples are given below.

[7]The parser for the RPG radiometer files is very slow in the current version. It will probably need to be rewritten if large amounts of RPG data will be handled.

Insert electronic balance data which has timestamps in Japanese time, using the parameter file `balance-kanazawa2004.ini`. Insert data from all files in the `/data/balance` directory:

```
./insertdata.py -d balance -p balance-kanazawa2004.ini -u +09 -v /data/balance/*
```

Insert WVR radiometer data which has timestamps in UTC time. Insert only data after January 25, 2003 at 15:00:00. Use the parameter file `rpg-fukui-2003.ini` and an alternative configuration file. Use verbose mode but don't ask any questions:

```
./insertdata.py -c insertdata-arrays.conf -d rpg_brt -p rpg-fukui-2003.ini -u +00
-s "2003-02-25 15:00:00+00" -v --no-questions /data/wvr/*.los
```

Please be especially careful with the time zone information. Even if the instrument is installed in Japan, the timestamps in the data files might be in UTC time, for example if the logger computer clock was set in UTC. In such case, the `location_utc_offset` in parameter file (the `.ini` file) should be +09 to describe the location of the device, but the value of the `-u` command line parameter should be `+00`!

The `-s` (`--starttime`) and `-e` (`--endtime`) parameters can be used to insert only a part of the data from a file. In most cases all the data should be inserted and these parameters are not necessary. They can be useful especially in connection with the `-r` (`--reliability`) parameter, which allows to specify which reliability value is given to the data. The reliability is an integer field in the database, but a policy on how to use it has not been decided yet.

The `-v` (`--verbose`) mode can be used to control how much information is presented to the user. By default, the script is relatively quiet and only error messages and critical questions are displayed. If the verbosity level is higher, more information is shown and the user is asked for a confirmation in additional cases. To raise verbosity, specify the `-v` one or several times. Currently, level 1 is recommended when the script is used interactively. Level 2 (`-v -v`) gives extra verbose output, for example shows all SQL commands sent to the database. It is mainly useful only for debugging, because it produces a huge amount of output. An example of output using different verbosity levels is shown below.

Verbosity level 0:

```
[teras@pyxis23] ./insertdata.py -d poss -p poss.ini -u "+09"
/data/wakasa/poss/ps030120
[teras@pyxis23]
```

Verbosity level 1:

```
[teras@pyxis23] ./insertdata.py -d poss -p poss.ini -u "+09" -v
/data/wakasa/poss/ps030121
Connecting to database wakasa as user teras
Connected ... Now creating a cursor
getParamsets: Parameter set with name wakasa2003 already found in the
database, checking for conflicts and merging data.
Using the following values for parameter set wakasa2003:
paramset_id : 2
location_longitude : 136.22
creation_time : 2003-06-23 16:50:07.076821
```

```
location_latitude : 36.14
time_resolution : 60.0
location_utc_offset : 9
instrument_description : Andrew POSS
Proceed (y/n)? y
Writing parameter set wakasa2003 to the database.
Processing file /data/wakasa/poss/ps030121...
done.
[teras@pyxis23]
```

Verbosity level 2:

```
[teras@pyxis23] ./insertdata.py -d poss -p poss.ini -u "+09" -v -v
/data/wakasa/poss/ps030122
Verbosity level is 2
Connecting to database wakasa as user teras
Connected ... Now creating a cursor
Executing query SELECT paramset_id, name, creation_time,
location_latitude, location_longitude, location_elevation,
location_utc_offset, instrument_description, time_resolution FROM
poss_parameters WHERE name = 'wakasa2003';
getParamsets: Parameter set with name wakasa2003 already found in the
database, checking for conflicts and merging data.
Using the following values for parameter set wakasa2003:
paramset_id : 2
location_longitude : 136.22
creation_time : 2003-06-23 16:50:07.076821
location_latitude : 36.14
time_resolution : 60.0
location_utc_offset : 9
instrument_description : Andrew POSS
Proceed (y/n)? y
Writing parameter set wakasa2003 to the database.
Executing query UPDATE poss_parameters SET name = 'wakasa2003',
creation_time = '2003-06-23 16:50:07.076821', location_latitude =
36.140000000000001, location_longitude = 136.22, location_elevation =
NULL, location_utc_offset = 9, instrument_description = 'Andrew POSS',
time_resolution = 60.0 WHERE paramset_id = '2';
Committing changes to database
Processing file /data/wakasa/poss/ps030122...
Executing query INSERT INTO poss(time_utc, paramset, reliability,
mean_freq, freq_stdev, mode_freq, mode_power, total_power,
temperature, above_noise_percentage, precip_type,
precip_intensity_code, precip_accum, precip_rate, error_code, fft)
VALUES ('2003-01-21 15:00:48', '2', NULL, '455.100000', '291.500000',
'48.000000', '0.100000', '3.000000', '2.000000', '48', 'N', '0',
'18.500000', '0.000000', '0000', NULL);
```

```
Committing changes to database
Executing query INSERT INTO poss(time_utc, paramset, reliability,
mean_freq, freq_stdev, mode_freq, mode_power, total_power,
temperature, above_noise_percentage, precip_type,
precip_intensity_code, precip_accum, precip_rate, error_code, fft)
VALUES ('2003-01-21 15:01:48', '2', NULL, '456.600000', '291.000000',
'48.000000', '0.000000', '3.000000', '2.000000', '48', 'N', '0',
'18.500000', '0.000000', '0000', NULL);
...
[Large amount of output removed]
...
done.
[teras@pyxis23]
```

4.5 Parameter set consistency check

If the parameter set with the same name is already found in the database when inserting data, `insertdata.py` does a simple consistency check. This is implemented to detect common mistakes, for example creating a new parameter file with new values for a new experiment, but forgetting to change the parameter set name.

Basically, the check ensures that there are no conflicting values in the sets, but additional values are allowed. This allows for example to create a simple parameter file first, and later add the `location_latitude` and `location_longitude` information. The new information is then updated in the database. On the other hand, if the location information is different than that already stored in the database, an error is signaled.

4.6 Trying to insert duplicate data

Probably the most common error when using `insertdata.py` will be to process data which has already been inserted to the database. In this case the database will produce an error message, refusing to insert lines which would result in duplicate keys in a table. Key is the column or combination of columns which is used to identify rows and the combination of values which forms the key must therefore be unique on every row. An example:

```
[teras@pyxis23] ./insertdata.py -d poss -p poss-kanazawa.ini -u "+09"
/data/sft/Kan03/poss/ps030217
Error writing a block of data (timestamp 2003-02-16 15:00:27 UTC) in
the database. Probably due to duplicate data (trying to insert data
which has been inserted earlier).
Aborting, see message from the database below:
ERROR:  Cannot insert a duplicate key into unique index poss_pkey
```

(The database connection library does not return separate error codes for different types of database errors, and therefore the script reports the source of error as “probably”).

There is no "overwrite" option in the script, so if the user wants to reinsert the data (for example after manually fixing some errors in the data files) existing data must first be deleted from the database. Deleting data using the `deletedata.py` script is described in section 5 of this manual.

Note that if a new parameter set is given, the insert will not result in duplicate keys even if the device type and timestamp would be the same. Even an identical parameter set can be used by just changing the name. This can be useful if several similar devices are used simultaneously, for example if there are three electronic balances to measure snowfall rate. Data from all balances can be inserted to the same database by just using differently named parameter sets, and the measurement results thus easily compared.

4.7 Verifying data

After inserting data into the database, it is important to verify that it has been inserted correctly. The `insertdata.py` script and parsers should be able to handle most common error cases such as incomplete data lines in source files, but it is always possible that a bug is encountered and the insertion fails. Also, it is possible that the user mistakenly specifies a wrong UTC offset or a wrong parameter set name, resulting in erroneus data in the system.

The PostgreSQL command line tool `psql` can be used to view and edit the database contents by using SQL commands. There are also several graphical tools, for example `pgaccess` [4] is a relatively good free (open source) graphical database browser. Errors can often also be detected by visualizing the data using the plotting tools as described in the Wakasa Database User's Guide.

5 Deleting data from the database

Sometimes there's a need for deleting data from the database. It may be because of errors encountered when inserting it, or some data may not be needed any more. Also, if a data file has been inserted partially, the duplicate data problem may be encountered as described in section 4.6. Then it may be easiest to first delete the previously inserted part and re-insert the whole data file.

Any data from the database can be deleted using the standard SQL command DELETE, specifying the necessary conditions such as timestamp restrictions. However, the data is often distributed to more than one table (depending on the device), which makes the process more complicated. Therefore a helper script called `deletedata.py` has been written to enable simple deletion of data based on the device, parameter set and time range.

5.1 Data deletion script deletedata.py

The script `deletedata.py` can be used to delete data from the database. The syntax of the script is:

```
deletedata.py [-h] [-v] [-c conffile] [-d device] [-p paramset]
              [-s starttime] [-e endtime] [--no-questions]
```

The command line parameters accepted by `deletedata.py` are described in the following table. They are mainly the the same than in `insertdata.py`. The main difference is the `-p` (`--paramset`) option, which does not take a parameter file as argument but instead a parameter set name.

Table 4: Command line parameters of `deletedata.py`

Parameter		Description
`-c`	`--conffile=STRING`	Name of the file containing configuration parameters for the program. The format of the configuration files is exactly the same as for `insertdata.py`, described in section 4.3. If this parameter is not specified, the default filename `deletedata.conf` is used. Example: `-c deletedata-arrays.conf`
`-d`	`--device=STRING`	Type of the device whose data will be deleted. The supported devices are the same as those supported by `insertdata.py`, listed in table 3. Example: `-d poss`
`-e`	`--endtime=TIME`	Delete only data before the given time. Syntax: YYYY-MM-DD HH:MM:SSZZZ, where ZZZ is the time zone information (ISO 8601 notation). Example: `-e "2003-01-27 21-00-00+09"`
`-h`	`--help`	Show the help screen.
	`--no-questions`	Skip all questions, even in verbose mode. Without this option, the script asks for a confirmation before deleting data.
`-p`	`--paramset=NAME`	Delete only data associated with the given parameter set. Example: `-p kanazawa2003`
`-s`	`--starttime`	Delete only data after the given time. Syntax: as endtime. Example: `-s "2003-01-27 20-00-00+09"`
`-v`	`--verbose`	Verbose output. This option can be given several times to reach the desired verbosity level. Levels 0 and 1 are recommended for normal use. Level 2 is useful for development and debugging, as all SQL queries and other detailed information is printed. Examples: `-v` (verbosity level 1), `-v -v` (verbosity level 2)

The `-d` (`--device`) parameter is obligatory, all others optional. If only the device is specified, all data from the tables of that device and also parameter sets from the corresponding parameter table are deleted from the database. Also, if the parameter set is specified without a time range, all data associated to that parameter set and the parameter set entry itself are deleted. In the case that all data from all devices should be deleted, it is easier to delete the whole database using the `dropdb` command and recreate the table structure from scratch.

The following example deletes all POSS data with timestamps later than February 15, 2003 (midnight, Japanese time) which is associated to the parameter set `kanazawa2003`. In verbose mode, a confirmation is asked before proceeding.

```
[teras@pyxis23] ./deletedata.py -d poss -p kanazawa2003 -s "2003-02-15 00:00:00+09"
-v
```

```
Preparing to delete device poss data.
Parameter set: kanazawa2003
Starting time (UTC): 2003-02-14 15:00:00
Ending time (UTC): Not specified
Proceed (y/n)? y
Connecting to database wakasa as user teras
Connected ... Now creating a cursor
[teras@pyxis23]
```

6 Performance considerations

The database performance depends largely on the type of queries which are used to retrieve the data. Currently the base has not been optimized for any particular type of use. If a performance bottleneck is encountered in the future, it may be possible to improve the response time even by an order of magnitude without buying any new hardware. A few possible solutions are listed below.

1. SQL query reordering. In SELECT queries involving several tables, the performance may depend significantly of the order in which the tables are joined and data retrieval conditions written. For more information, see the PostgreSQL manual (especially the section titled "Performance hints") or any database book which contains a section about optimizing SQL queries.
2. Vacuuming. The Postgres VACUUM command should be run on a regular basis, especially if data is deleted from the database. See PostgreSQL manual section "Routine database maintenance tasks" for more information.
3. Creating indexes. Columns which are part of the key of the table are automatically indexed by the database using an efficient algorithm. However, if conditions are imposed on other columns (for example, retrieving ceilometer data with the condition `cb_height_1 < 1000`), the performance may improve significantly if an additional index created for those columns. For more information, see the section "Indexes" in the PostgreSQL manual.
4. Temporary tables. If a complex query takes a long time and the same values are needed very often, it can be useful to store the results of the query in a temporary table. Then the time consuming operation needs to be done only once, and subsequent queries can get the selected data quickly from the temporary table. For more information, see the documentation for `CREATE TABLE AS` command in the PostgreSQL manual (section "SQL commands").

References

[1] PostgreSQL database engine online documentation. `http://www.postgresql.org/docs/`

[2] PostgreSQL version 7.2 online documentation for the GRANT SQL command, with comments. `http://www.postgresql.org/docs/7.2/interactive/sql-grant.html`

[3] Image Information Science Laboratory, University of Kanazawa, Snowfall Team home page. `http://wis02.ec.t.kanazawa-u.ac.jp/research/muramoto/sf/`.

[4] PgAccess, PostgreSQL database administration tool, home page. `http://www.pgaccess.org/`