server\_gain user manual

Contents

[1 About server 2](#_Toc77718474)

[2 Interacting with the server 2](#_Toc77718475)

[3 Setting up server 5](#_Toc77718476)

[3.1 Setting up Python and PIP 5](#_Toc77718477)

[3.2 Setting up database 6](#_Toc77718478)

[3.3 Setting up server 7](#_Toc77718479)

# About server

This is TCP/UDP server that can be used to receive data from and send GPRS commands to the FMB devices. This server can support multiple simultaneous connections at the same time, as well as SSL communication channels and automatic GPRS command sending via both TCP and UDP.

Up to version 1.2 this server is written to be similar to *server\_main* except it has *Graphical User Interface*, can handle multiple connections and manually disconnect clients. Unlike *server\_main*, this server doesn’t have connection timeout (*server\_main* closes connection with the device after 300 seconds of inactivity) and the connection with the device is active indefinitely unless the device itself closes it (configurable connection timeout should be implemented in future releases).

From version 2.3 this server is redesigned to perform automatic beacon testing. For this separate *Postgres* database is needed which is used to store data received from the devices.

If testing mode is enabled, the server will request information every configured time period from the database, calculate beacon statistics and out put it to the text browser as well as to .csv files.

There are 2 .csv files used during the test. One is *test\_<date>.csv* the other is *missing\_<date>.csv*. Test file contains periodic and full test (at the time) data while missing file contains all the times the beacon when missing (didn’t appear in the record received).

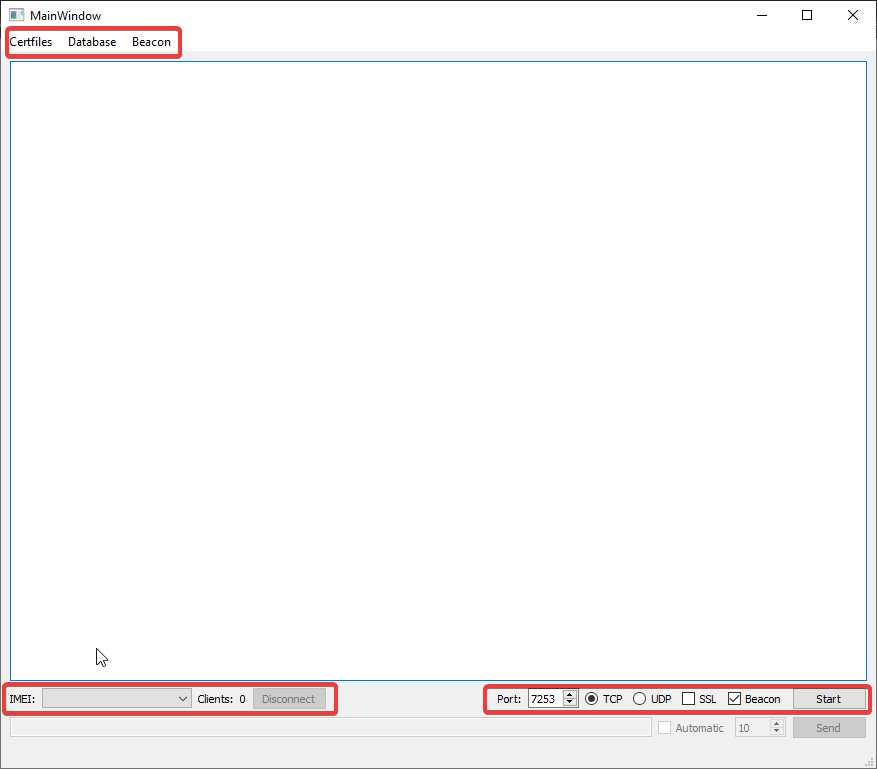
When application is started *backup.sql* file is created if it wasn’t present at the time. This file is used to store SQL queries when database is down or can’t be reached. This way no or minimal amount of test data is lost. When database is up again, data from backup file is transferred to the database and file is cleared. If database must be fixed manually, file can be used to transfer data via SQL commands. If *backup.sql* file gets corrupted in some way, data is moved to the *backup\_err.sql* file and *backup.sql* file is cleared.

When the server is run, it has 2 windows – a GUI window and console window. Console window contains all of the logging data of the server. Same data is saved in *application\_events.log­* file.

Server can be run using compiled binary file or using source code, since Python is interpreted programming language. More on how to run the server in *Setting up server* paragraph.

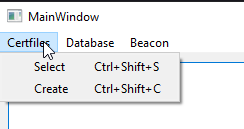
# Interacting with the server

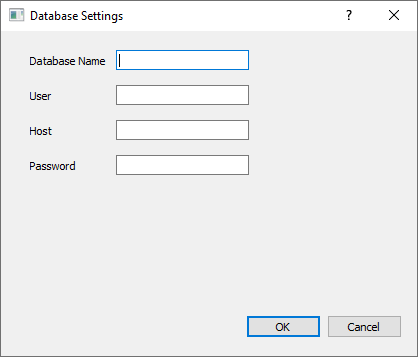
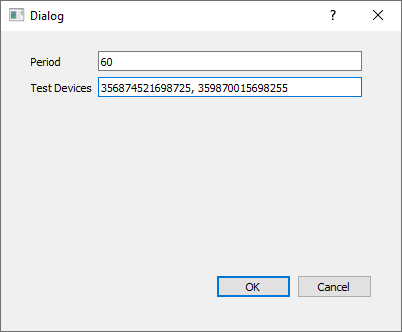
Server has multiple GUI widgets that help to configure its parameters. At different states of the server (such as server is off or server is running) different widgets can be enabled, disabled or change.



Most of the widgets and their actions are self-explanatory, however, some may need further explanation. Here are some of them:

* *Certfiles* – located at the left corner of the toolbar. Menu that allows for selection of certificate files, used when connection is wrapped around Secure Socket Layer (SSL). There is additional *Create* menu option which would allow the quick creation of certificates but this feature is not implemented as of day of writing.



* *Database* – opens database settings widow where user can configure parameters required for successful transactions between server and database. As of day of writing, only used when Beacon test is enabled.  
    
  
* *Beacon* – opens beacon test parameters window where user can configure parameters of beacon test. *Period* – determines how often (in seconds) server queries the database for periodic test results. *Test Devices* – determines which devices participate in a beacon test. Server will only gather data from database from the devices that participate in the test. This parameter should be specified in this format to avoid errors: *<imei>, <imei>, <imei>,…  
    
  *
* *Beacon checkbox* – located in lower right corner, between “START” button and “SSL” checkbox. If checked, parsing of beacon data, database connection and beacon test all are ON.   
    
  
* *Automatic checkbox* – toggles on and off automatic GPRS CMD sending. Command should be specified in line edit widget on the left and send period should be specified in spin box to the right.  
    
  

Once server is started it can be stopped by pressing the „STOP“ button. By doing so, all of the connections are terminated including the connection with the database.

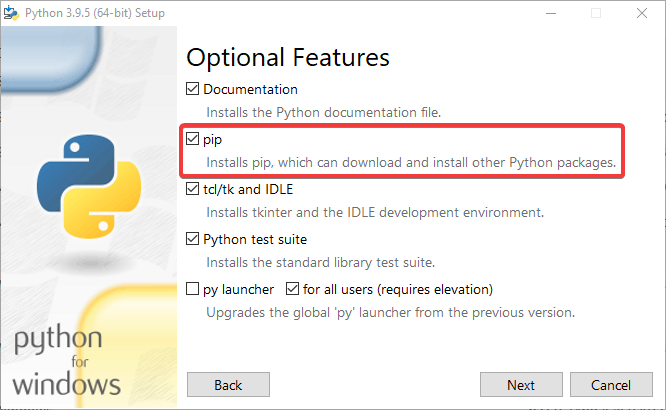
# Setting up server

## Setting up Python and PIP

Server uses Python interpreter thus Python 3 (preferably the latest stable version) is required to run.

Python 3 can be downloaded and installed [here](https://www.python.org/downloads/).

While installing Python 3 it is highly recommended to also install *pip* which is a module manager that allows installation of third-party modules with a single command, simplifying the process.



When Python and pip are installed, some third-party modules must be installed. Without these modules server will fail to run and *ModuleNotFoundError* will be raised. To install third party modules open Power Shell window and type the command:

*pip install <module\_name>*

As of today, the list of used modules in all of the scripts defined in this documentation, consist of the following:

* [PyQt5](https://pypi.org/project/PyQt5/)
* [psycopg2](https://pypi.org/project/psycopg2/)
* [libscrc](https://pypi.org/project/libscrc/)

User’s installation of Python may lack some other modules. All modules can be installed using pip with the command described above.

## Setting up database

For beacon testing Postgres database is needed. Easiest way to set up Postgres database is to use Docker and Docker image of it.

To install Docker on your machine, follow the instructions [HERE](https://docs.docker.com/get-docker/).

Once Docker is installed, now it’s time to pull Postgres image and set up database.

In detail description how to do it can be found [HERE](https://hub.docker.com/_/postgres), but bellow are few commands that should do:

*docker pull postgres*

*docker run –name <name of container> -e POSTGRES\_PASSWORD=<database\_password> -p 5432:<your\_port> -d postgres*

Example of the second command could be:

*docker run –name postgres\_beacon -e POSTGRES\_PASSWORD=’password’ -p 5432:7000 -d postgres*

Now that the Postgres image is up and running, the database itself can be setup. This is done in 2 steps – first step is to create a database manualy by entering Docker image a

*docker exec -it <docker\_conainer\_name> /bin/bash*

This command will enter into the BASH shell of Postgres container. You should see *bash-5.1#* or something similar on a left side. Now enter Postgres database itself using:

*psql -U postgres*

By default the user name is *postgres*. Now you’re in Postgres database shell. You can also access the same shell remotely if you have [psql](https://blog.timescale.com/tutorials/how-to-install-psql-on-mac-ubuntu-debian-windows/) installed. In this case, type in the remote machine:

*psql -h <ip\_address> -U postgres -p <database\_port>*

If everything’s alright, you should be prompted for password and should see the same shell as if you were in Docker container.

Now you should create database. As of time of writing this document, the database name in server application is hardcoded to “beacon\_test”. Unless you know what you’re doing, uses this name for a database:

*CREATE DATABASE beacon\_test;*

**Don’t forget the semicolon at the end of the command**. Semicolon indicates that the command is finished and should be executed.

Now the database tables and relations between them must be initialized. This can be done manually from Postgres shell (after connecting to database using *\c beacon\_test*) or by using a file – *init\_db.sql*

If you decided to enter commands manualy, make sure that you’re connected to the database (lshell tag on the left should be *beacon\_test=#)* and enter commands line by line.

Alternatively, you can transfer file to Docker container and execute it from there. Get back to the shell of your machine and use:

*docker cp init\_db.sql <name\_of\_container>:/init\_db.sql*

*docker exec -it <name\_of\_container> psql -U postgres -d test -f /init\_db.sql*

If no errors occurred, the database is finally initialized.

## Setting up server

Server can be run as a compiled binary or run from code. If don’t have a compiled binary of the server, make sure that you have all Python dependencies met listed above (if they’re not met, use *pip* to install missing modules). Then install [pyinstaller](https://pypi.org/project/pyinstaller/) which is used for compiling an executable binary from Python code. Navigate to the server folder and run this command:

*pyinstaller –onefile main.py*

If everything is alright and no errors occurred, the compiled binary should be in newly created *dist* folder.

Alternatively, you can run server using Python interpreter only. Navigate to the server folder and type:

*py -3 main.py*

Server may take some time to start if it’s started for the first time.