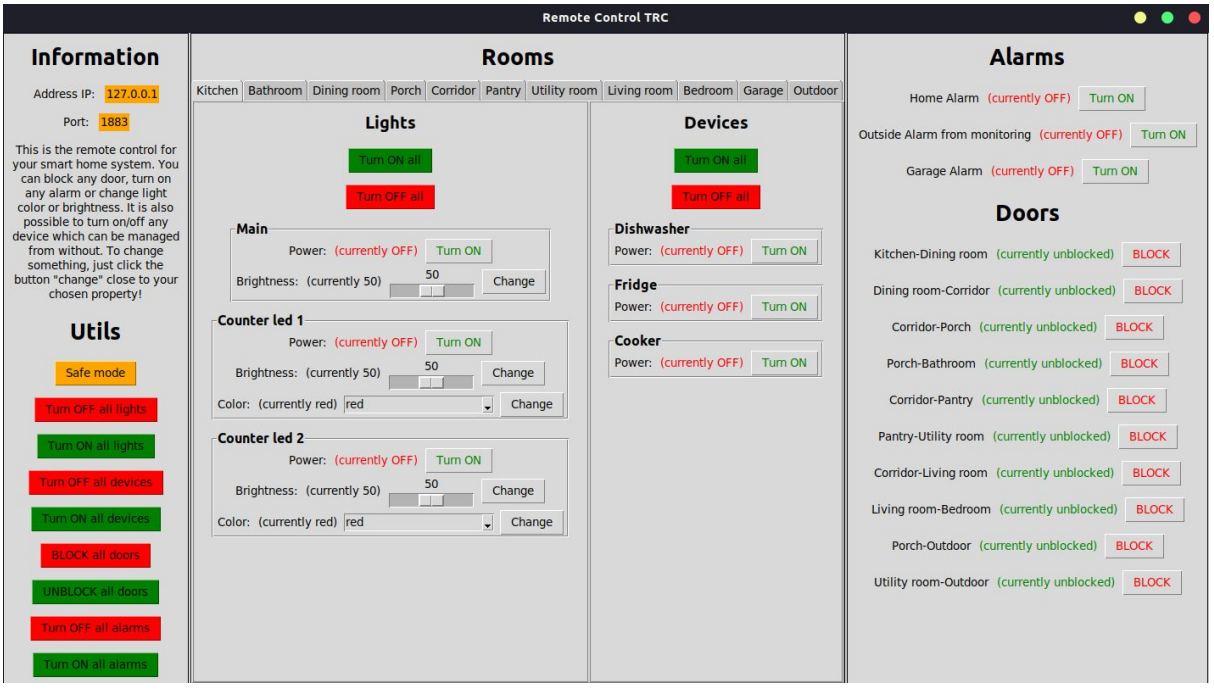
Smart Home Remote Control - Documentation

Introduction

Smart Home Remote Control is an application used as remote control to smart home. It allows you to connect to your home system and send commands, which can activate alarm or turn off the light at home. You can do it locally by connecting to the server through the router at home and entering local IP server address and port or by connecting to the server through the internet and entering public external IP server address and port. Remote control doesn't force you to use one fixed system configuration - you can change JSON files as you want to and you smart home system looks like. All this features make this app flexible and comfortable in usage.

Instruction

To run this app, you need to have python and mosquitto (MQTT) installed. After downloading all the package, in main directory you have to enter "python main.py" command in console. It will trigger connect window where you need to enter correct IP server address and port. After clicking "start" button, you will be able to see a main window of remote control where there is a possibility to execute specific commands.



1. Screenshot of remote control panel

The application is divided into some sections: (from the left) information about connection and functional buttons (for example all doors blockade or all alarms activation). Close to this section, there is another one with all rooms, where you can choose a specific room and turn on or turn off lights or devices. Additionally, you can control light brightness or color if your specific light allows you to do it (you can specify colors in JSON files). In every room you can find another functional buttons which allows you to do specific action on all lights or devices. The last section has alarms and doors - here you can change alarm activation or door blockade. Anything you want to do, must be finished by clicking the submit button (“change” or “turn on/off”) - it will send a specific command to the server. There is only one small difference in functional buttons - if you click them, you send a queue of commands changing object status to the given one, resulting from the button (for example “block all doors” will send a queue of commands to all unblocked doors and force them to be blocked).

Technologies

Used technologies in this application:

- python - native language,
- tkinter - graphic interface of the application,
- MQTT - data transmission protocol based on publish/subscribe pattern. In this case, protocol is used to communication with smart home system - remote control publishes commands in specific topic and server sends them to all topic subscribers,
- Observer pattern - it is an intermediary between app engine and graphic interface

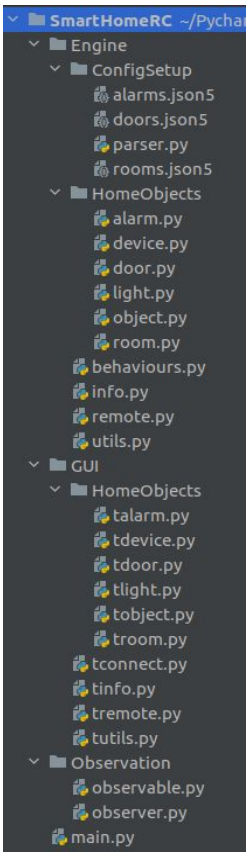
Construction

In main directory there is a main file named “main.py” which is used to start a remote control and there are also three other directories:

- Engine - an app engine,
- GUI - graphical interface for application,
- Observation - it connects engine with graphical interface.

The most important directory is Engine. Without Engine, remote control can’t run, but without GUI or Observer, it can still be run. If you want to run this app with Engine only, you have to implement integration between user with engine via console. Engine includes JSON files with all the configuration and parser.py file, which converts JSON objects into objects based on classes from HomeObjects directory. This directory has all classes definitions which represents objects to control. Object class in object.py is a root of any another class - Device, Light, Alarm inherit from Object. What more, Light class extends it additionally. Door and Room are classes which are not connected with Object (it is caused of data differentiation). Other files from engine directory provide grounds for remote control functionality. Info is the least important class - it informs only about connection (port and IP address) and show a short instruction. Utils class has functions which helps using the remote control (for example turn off all lights). Remote class is a root of this application - it connects with server, start your application and send specific commands to the server. Behaviours definitions are placed in Behaviours class. They describe what to do when you get the message from the server. This specific construction of application is a necessary

condition of remote control running and sufficient condition when you have an integration via console written. It is known that a typical user of this app don't have to be a software developer, that's why GUI is written here which is defined in GUI directory. It is responsible for displaying and executing engine functionality. HomeObject directory includes tkinter class versions with reference to class objects from HomeObjects in Engine directory. As in engine, TInfo is the least important class of he GUI. It displays everything what Info object contents. TConnectWindow (TConnect file) is the class which represents a start window where you need to enter correct server IP address and port. After clicking "start" button, you will be able to see a remote control panel which is defined in TRemote class. Functionality from Utils class is placed in GUI directory as TUtils class. Everything could be fine but there is no connection between engine and graphic interface - when something change in Engine, GUI doesn't know that it is changed. The solution for this problem is Observer Pattern included in Observation directory. Engine objects are Observable objects (inheritance) but GUI objects are Observer objects. When something changes in engine objects, all observer objects will be notified (in this case - GUI objects which will change their data on panel). Thanks to this pattern, I got not only efficient upgrade possibility but also engine and GUI severability. This is a good solution when you want to make a lot of complicated modifications - if engine had been written to cooperate with graphic interface, GUI technology replacement would have been impossible without engine changes.



Development

Thanks to construction, which is described above, this application is ready for changes. For example you can write GUI in another technology like PyQt, where you don't have to use Observer pattern because every event you can hold by signals and slots. The only thing you have to remember is to delete the observable heritance in Engine/HomeObjects classes and this is the only one thing from external directories and classes which is used in implementation of remote control engine. If you want to just extend this application (not modify), it will be possible thanks to the construction also. For example - to add a new functionality, you need to add a new parameter in JSON files (but it is not always necessary), set it in parser file, add a new attribute in Device class, add a new behaviour in case of message receiving (Behaviours class) and then, find out the way to set it in GUI (adding a new Frame with specific properties in TDevice class should be sufficient). If you want to add a new untypical functionality which could help you in remote control usage, you need to only define a new function in Utils class and add a new button with this function as a command (TUtils class). To sum app, application is written not only to enable modifications but also to allow you to extend it and develop it in your way.

Author:

Łukasz Wolski