User Guide

for the Accessory-Server system

1. Introduction

This guide explains how to install, configure and use the Accessory-Server.

The Accessory-Server consists of different software projects which are combined to provide a nice user experience. The main part is the server which publishes HomeKit-accessories in the local network and uses the information provided by the previous configuration to execute Commands. This Commands can be executed either through the Remote-App or through HomeKit-Accessories.

Commands can be:

* Recorded infrared signals from an infrared remote
* Recorded 433MHz radio signals from a remote control
* Arbitrary shell commands

In order to use the infrared and radio commands you need special hardware to record and send them. The hardware part is the hardest part of the whole installation, but if you know a little bit about electronics than this should not be a problem at all.

In contrast to commands, it is also possible to collect sensor data from digital or analog sensors. In order to use this feature, you need to write a little Arduino program which provides the sensor data. In the most cases this programs consist of about 10 lines of code.

There are some prerequisites to use the features of the system:

* Required: Raspberry Pi with the latest Raspbian Jessie (Lite)
* Required: iPhone (+ Apple Watch) with the latest iOS Version
* Optional: Hardware for sending and recording infrared signals
* Optional: Hardware for sending and receiving 433MHz radio signals
* Optional: Arduino Uno R3 board for providing sensor data

With the minimum configuration you would be able to send arbitrary shell commands, but in order to use all features you should have all parts. If something is explained and you don’t have the necessary hardware, just skip it.

1. Installation – Software

To install all required components for the server, just copy the 'AccessoryServer' directory to the Raspberry and run the 'install.sh' script without 'sudo'.

After the reboot, you can start the server with the 'start.sh' script (Use the '-d' option to get debug info on the console instead of the server.log file, and append the '-t' option to optionally recompile the sources).

You don't have to compile anything or do any configuration (the install script does all the work).

If you want to use shell commands, you need to install the Accessory-Hub as well.   
Just Copy the 'AccessoryHub.jar' file of the 'AccessoryHub/Example' directory to the Raspberry and create a 'commands.txt' file at the same location.

Start the application with 'java –jar AccessoryHub.jar X.X.X.X:8888' (IP of the Rasperry Pi and the port which is defined in the 'AccessoryServer/AppConfig.ts file' (default is 8888).

1. Installation – Hardware

The hardware part can be tricky and you should understand the basics of electronics.

For the infrared signals you need a transistor in order to switch the high current of 2A.

The high current must be provided by an external power supply. So pick some emitters, a 38kHz receiver, some resistors and a transistor and do the math behind it. In the end there should be one cable to send and one to receive signals.

For the radio signals just get any 433MHz receiver. Note, that these modules need a voltage of 12V in most cases. So if you are planning to use the same external power supply for infrared and radio components, you should pick a 12V/2A power supply.

You can play with the values in order to send the strongest signals. In my home, I used 2A for 2 infrared emitters in order to get a bright light with a wide angle. But this depends on the location of your Raspberry Pi.

Before connecting anything to the Raspberry Pi, make sure to read about the risks of damaging it!

1. Configuration of Remotes

Remotes are configured with the Remote-App. At the first start of the app you have no other choice than adding a new Remote. Later you will be able to add additional Remotes by clicking the add button in the remotes overview. This action starts an assistant which guides you through the process.

A Remote is only a collection of commands. After the configuration of the remote you can edit the commands, add new ones or add an empty compound command. Each command can can contain other commands to form a compound command. When a compound command is clicked, all commands in the list are sent sequentially. To prevent the system from sending the commands to fast, you can also add a delay.

If you want to use shell commands, you have to switch the assistant to the IP-mode. In this mode you can pick a shell command which needs to be defined previously in the commands.txt file.

You can use as many Accessory-Hubs as you want. Note, that a command is sent to all Accessory-Hubs on activation.

1. Configuration of Accessories

Accessories are, like Remotes, configured with the Remote-App. The configuration process is similar. The assistant guides, you through the process. You can use any command or compound command from any remote as action for the new accessory. This commands have to be configured in advance by configuring a remote.

There are different accessory types you can use:

* Always On
* On/Off
* On/Off + Control
* On/Off + Control + second On/Off

These types define how an accessory behaves in the Home-App and how the icon will be displayed.

The most powerful type is the last one, which utilizes two switches and one control ranging from 0 to 100 percent.

If you are using a type with a control utility, you can set a max value for the steps. The max value conforms to 100 percent.

When the configuration is finished, you can switch to the Home-App and add the new accessory. The code is simply '000-00-000 for all accessories.

You can delete accessories again through the Remote-App, but you have to remove them manually from the Home-App.

1. Using Remotes

Remotes can be used in two ways. You can send commands by clicking them in the list, or by swiping in a direction at the touchpad. The touchpad will only appear in the remote if at least one command was configured for it in the assistant. You can edit all commands and remotes at any time including the touchpad configuration.

1. Adding Sensors

In order to use sensors, you should understand the basics of programming.

In the 'AccessoryServer/SensorsExample' you find an example for a digital and analog sensor. The only important thing is to connect the Arduino per USB to the Raspberry Pi.

The program can do whatever you want as long as it sends values in the right format to the Raspberry Pi.

As long as you keep the folder structure and the main file, you can upload the script to the Arduino by starting the 'upload.sh' script.

You can also activate more debug information by changing the appropriate value in the 'AccessoryServer/AppConfig.ts' file.

During the Assistant in the Remote-App you need to provide the right pin number of the Arduino.

In the case of analog sensors, it is also possible to set an analog reference value. So you don’t have to care about the reference in the Arduino program.

1. Troubleshooting

* The App records an infrared signal but sending it does not work:

In some cases, it simply doesn’t work. The workaround for this problem is to look for a LIRC configuration file for your device. Then add the whole file to the beginning of the '/etc/lirc/lird.conf' file. You can also try to create a file with the 'irrecord' command.

Then you can add the 'irsend' command for each signal to the 'commands.txt' of a local running Accessory-Hub.

* The Home-App doesn’t find any accessories anymore:

This are bad news in most cases. You can try to reboot your iPhone and starting the server once with './start.sh –d –t - r' (But then you have to add all accessories again in the Home-App.)

* The commands of the Accessory-Hub are not shown during the configuration:

You have to start the java executable with the IP and the port of the server.

If this does not help you could check the server.log for 'client connected' messages.

* The accessories work on iPhone but not on iPad:

You need to log in with the same iCloud accounts on both devices. If this does not help reboot both devices.

* Server crashed:

Open an issue on GitHub with the server.log file containing the crash message.