digitalSTROM Smartphone Apps

digitalSTROM

Version: master-branch*

June 23, 2014

*Revision: 6b45505f8633508e7540608978707e05d4eba319

digitalSTROM Alliance
Brandstrasse 33
CH-8952 Schlieren-Zürich
Switzerland

Contents

1 Prerequisites

In order to develop smartphone and other external applications for digitalSTROM, it is essential to first understand the system and its concepts. So please read the following documents before proceeding:

- digitalSTROM Basic Concepts
- digitalSTROM System Interfaces

Communication with the digitalSTROM Server is only available through the JSON api, so a JSON handler is needed. Using an asynchronous networking client is also encouraged.

2 The digitalSTROM JSON API

External applications communicate with the digitalSTROM Server through the JSON API. See the JSON documentation for a description of all the available JSON functions and parameters. Access is granted with a token system described in 3. By default port 8080 should be used, but since the user might have the port forwarded, it should be configurable.

JSON requests to the API are built up like this:
https://<server ip>:<port>/json/<class>/<function>?<paramter>&<parameter>

For example this function force calls scene 5 on light devices in zone 1307:
https://10.0.0.2:8080/json/zone/callScene?id=1307&groupID=1&sceneNumber=5&force=true&token=xxxxxxxx
Not all JSON functions take parameters, for example *json/apartment/getConsumption*.

The JSON formatted reply returns true for "ok" if the digitalSTROM Server successfully processed the query[1], and JSON objects/arrays if data was requested.

Example reply (json/apartment/getConsumption):

```
{
    "ok": true,
    "result": {
        "consumption": 74
    }
}
```

Tip: Install a JSON plugin in your browser and test the JSON commands you want to use beforehand, to see the exact formatting of the reply.

> **Notice** The digitalSTROM Server uses a self signed certificate, so in order to connect the user should accept that the certificate is not signed by a known authority. This can also be solved by simply accepting any certificate in your network client.

[1] This does not guarantee execution on the digitalSTROM Meter or digitalSTROM Device though.

3 Handling login and tokens

Access to the JSON API is granted through a token-based login method. An application token is obtained from the digitalSTROM Server and then subsequently used to request session tokens, which provide temporary access.

3.1 First Login

Connecting for the first time, a new application token is requested: [2]
/json/system/requestApplicationToken?applicationName=<Your App Name>

The application token returned is a string that must be persisted in the application. The token needs to be enabled before it is active. To do so, ask the user for username and password for the digitalSTROM Server, and use these to perform a login:
/json/system/login?user=<username>&password=<password>

This returns a temporary session token, which can then be used to enable the application token:
/json/system/enableToken?applicationToken=<application token>
&token=<session token>

> **Notice** Do not persist the username and password in your application.

3.2 Subsequent Logins

For subsequent logins, start with requesting a session token using a valid application token:
/json/system/loginApplication?loginToken=<application token>

This returns a session token that will stay valid for 60 seconds. This time period is reset every time the token is touched. The token is applied to all the JSON commands e.g.:
/json/apartment/getConsumption&token=<session token>

[2] This function won't work if requested from a logged in session.

4 Test Server

If you don't have access to a digitalSTROM setup, you can remotely test your application and JSON commands against a setup available at **https://testrack2.aizo.com**. To access port 8080 of the digitalSTROM Server use port 58080, and for the configurator use port 50443. Username and password is *dssadmin*.

A web cam is monitoring the test setup. The real time video can be seen on the right side of the configurator page. Under the video feed is a description of the placement of the devices.

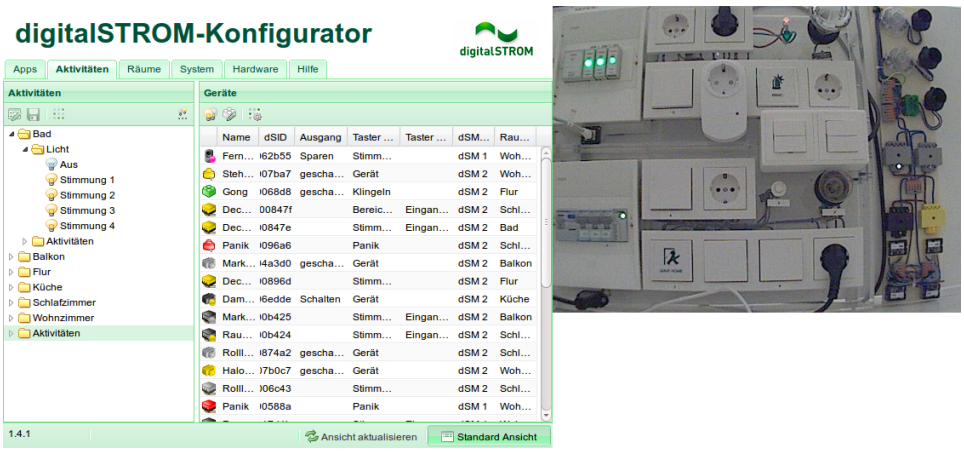

Figure 1: Screenshot showing the testrack2.aizo.com configurator with the embedded webcam view

5 Hello digitalSTROM apps

5.1 Android

The networking client used is the callback based *Android Asynchronous Http Client*[3] by loopj/James Smith. All of the digitalSTROM JSON calls are collected in the *DsJSONService.java* class, and the application runtime data are kept in the *App.java* extension class. The only data persisted past application sessions are server ip, port and application token. These are stored using *Shared Preferences*.

All the functions in DsJSONService.java are built up like this:

1. Use passed arguments to build the URL string.
2. Execute HTTP GET with the URL string.
3. Create new JsonHttpResponseHandler().
4. Set onSuccess callback (executed when GET is successful)
   (a) Checks if the "ok" boolean is true.
   (b) Process the rest of the digitalSTROM Server JSON reply.
   (c) Broadcasts completion if needed.
5. Set onFailure callback
   (a) If the GET fails, it is most likely because of an invalid session token.
   (b) Call getSessionToken() passing a copy of the failed function to be retried if successful.

When logging in for the first time, the enableApplicationToken() function is called. This performs the JSON function *json/system/login*, and on success the function *json/system/enableToken* ending out in a success (or failure) broadcast.

[3] Find it here http://loopj.com/android-async-http/

5.2 iPhone

The networking client used is the block based *AFNetworking*[4] by Mattt Thompson. All of the digitalSTROM JSON calls are collected in the singleton class *DigitalstromJSONservice*, with the digitalSTROM related runtime data being kept in this class as well. Communication throughout the app is accomplished using the observer pattern, where other classes can observe digitalSTROM related data in the *DigitalstromJSONservice* class.

All the JSON related functions in DigitalstromJSONservice class are built up like this:

1. Use passed arguments to build the URL string.
2. Setup a AFJSONRequestOperation using the URL string.
3. Setup the success block (executed when request is successful)
   (a) Checks if the "ok" boolean is true.
   (b) Process the rest of the digitalSTROM Server JSON reply.
4. Set the failure block
   (a) If the request fails, it is most likely because of an invalid session token.
   (b) Call updateSessionToken passing a block with the failed function to be retried if successful.
5. Start the the request operation.

[4] Find it here http://afnetworking.com

6 Tips and Tricks

Most of the digitalSTROM Server data is exposed in the property tree[5] so you will quickly find the */json/property/query* JSON call very useful. Here is an example query that retrieves the names and ids of User Defined Events on the digitalSTROM Server:

/json/property/query?query=/usr/events/*(name,id)

The property tree can be browsed from the digitalSTROM Configurator, by selecting "Advanced View" --> "System" tab --> "Property Tree".

Scenes that have not been renamed, or otherwise modified, will not have a node in the property tree. They can still be called though. Your application have to take this into account, usually by creating your scene objects with default names, and then update their names according to the values returned from the digitalSTROM Server.

[5] See the the *Property Tree* chapter in the digitalSTROM System Interfaces document.

7 Further references and reading