After I wrote my Alexa app to flush my toilet with a raspberry pi (which can be seen here:), the next logical step is multi-platform support. Google also offers ways to integrate its Google Home with other smart-home devices. Unfortunately for me, the process is a bit bulky and rather complicated. That is, I don’t want to spend all my time spinning up and securing websites and databases for users who will never come. For more information on setting up your own smart-home device in a production environment you can look at google’s documentation [here](https://developers.google.com/actions/smarthome/?utm_campaign=iot_update_actionsongoogle_020718&utm_source=gdev&utm_medium=yt-desc).

Figure 1. Yellow is infrastructure I'd have to support.

Like my Alexa app, I wanted to seek a more simple approach, to get the proof of concept out there and to get my toilet up and running! With that in mind, I followed along with [this](https://codelabs.developers.google.com/codelabs/smarthome-washer/index.html#0) Google walkthrough and added my own twist. Without further ado, lets get started!

The first thing I did was complete the guide mentioned earlier up to and including step 6. If you’re interested in these devices and more in-depth implementations, I encourage you to complete the guide as it offers a more in-depth exploration of the stack. That being said, let’s move on.

In the guide, many features are added to the functions/index.js file. Looking back at Figure 1, the whole firebase project in the guide *more or less* exists as the big yellow “Partner Cloud Services” block which is what we will leverage to communicate with our device. So functions/index.js is where we will modify the code. The first thing we’ll do is get the toilet up and running.

Previously, I’ve set up my raspberry pi to host an endpoint, listening for API requests. When it receives a request, it will move the servo motor which in turn flushes the toilet. Our goal, is to call this API endpoint from our “Partner Cloud Service”. In order to do that, we’ll call the service from our index.js file. Change the “onExecute” method to reflect the following code being sure to change <your-device-ip> to your flushing device’s IP address, e.g. 192.168.0.2

app.onExecute((body) => {

const {requestId} = body;

const payload = {

commands: [{

ids: [],

status: 'SUCCESS',

states: {

online: true,

},

}],

};

for (const input of body.inputs) {

for (const command of input.payload.commands) {

for (const device of command.devices) {

const deviceId = device.id;

payload.commands[0].ids.push(deviceId);

for (const execution of command.execution) {

const execCommand = execution.command;

const {params} = execution;

switch (execCommand) {

case 'action.devices.commands.OnOff':

firebaseRef.child(deviceId).child('OnOff').update({

on: params.on,

});

payload.commands[0].states.on = params.on;

//https://stackoverflow.com/questions/32604460/xmlhttprequest-module-not-defined-found

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function () {

if (this.readyState == 4 && this.status == 200) {

console.log("FLUSH SUCCESS!");

}

};

xhttp.open("POST", "http://<your-device-ip>:8614/flush", true);

xhttp.send();

break;

case 'action.devices.commands.StartStop':

firebaseRef.child(deviceId).child('StartStop').update({

isRunning: params.start,

});

payload.commands[0].states.isRunning = params.start;

//https://stackoverflow.com/questions/32604460/xmlhttprequest-module-not-defined-found

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function () {

if (this.readyState == 4 && this.status == 200) {

console.log("FLUSH SUCCESS!");

}

};

xhttp.open("POST", "http://<your-device-ip>:8614/flush", true);

xhttp.send();

break;

case 'action.devices.commands.PauseUnpause':

firebaseRef.child(deviceId).child('StartStop').update({

isPaused: params.pause,

});

payload.commands[0].states.isPaused = params.pause;

break;

}

}

}

}

}

return {

requestId: requestId,

payload: payload,

};

});

Also, be sure to declare your XMLHttpRequest variable at the top of the file

var XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;

Once we’ve done that, we can run the following command

$>npm install xmlhttprequest

Running this command in the functions directory will install the newly required dependencies. Why am I using xmlhttprequest? I don’t have a good reason other than its consistent with other files in the project. Finally, on the free tier of firebase, Google does not support making external requests. We’ll have to enable billing for this project. It probably won’t be a problem if you don’t intend to flush your toilet too many times. Go to the Google Cloud Platform console to enable billing for your project. After that we can firebase deploy to get our changes.

You can stop here if you don’t mind your toilet being called washer, but if you want a slightly better experience you can implement the following in the onSync method in functions/index.js

app.onSync(body => {

return {

requestId: 'ff36a3cc-ec34-11e6-b1a0-64510650abcf',

payload: {

agentUserId: '123',

devices: [{

id: 'toilet',

type: 'action.devices.types.SWITCH',

traits: [

'action.devices.traits.OnOff',

],

name: {

defaultNames: ['My Toilet'],

name: 'Toilet',

nicknames: ['Toilet']

},

deviceInfo: {

manufacturer: 'Acme Co',

model: 'acme-washer',

hwVersion: '1.0',

swVersion: '1.0.1'

},

attributes: {

pausable: false

}

}]

}

};

});

Unfortunately, at this point in time Google does not have support for smart toilets, so I chose the device type SWITCH since like a toilet, it only has two states. After implementing this, you should be able to say ‘turn on toilet’ to flush the toilet!