OPEN CONNECTIVITY FOUNDATION®

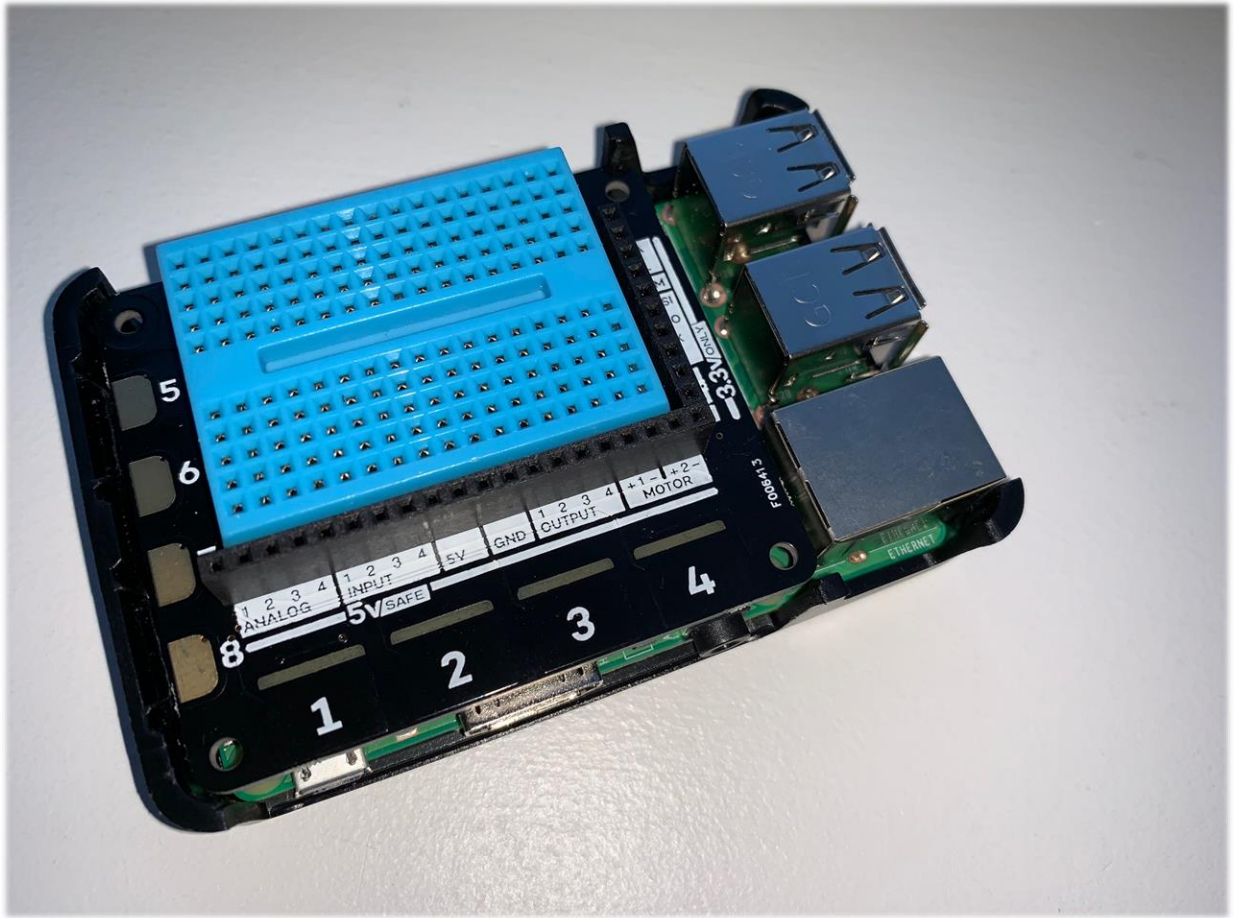
Build an OCF Device on Your Hardware

Develop a secure, certified OCF prototype on your own hardware in minutes

Clarke Stevens
Shaw Communications
clarke.stevens@sjrb.ca

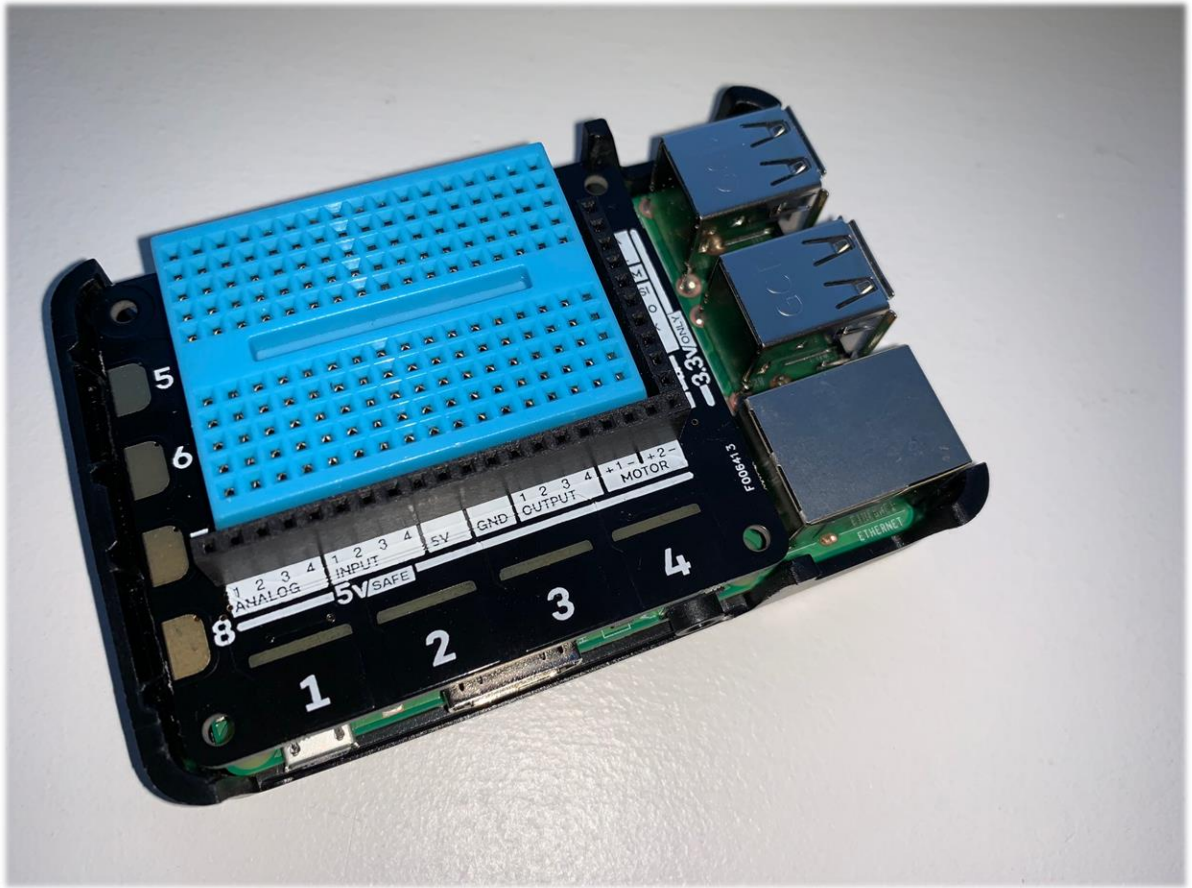
Shaw

Create your hardware platform that works with OCF



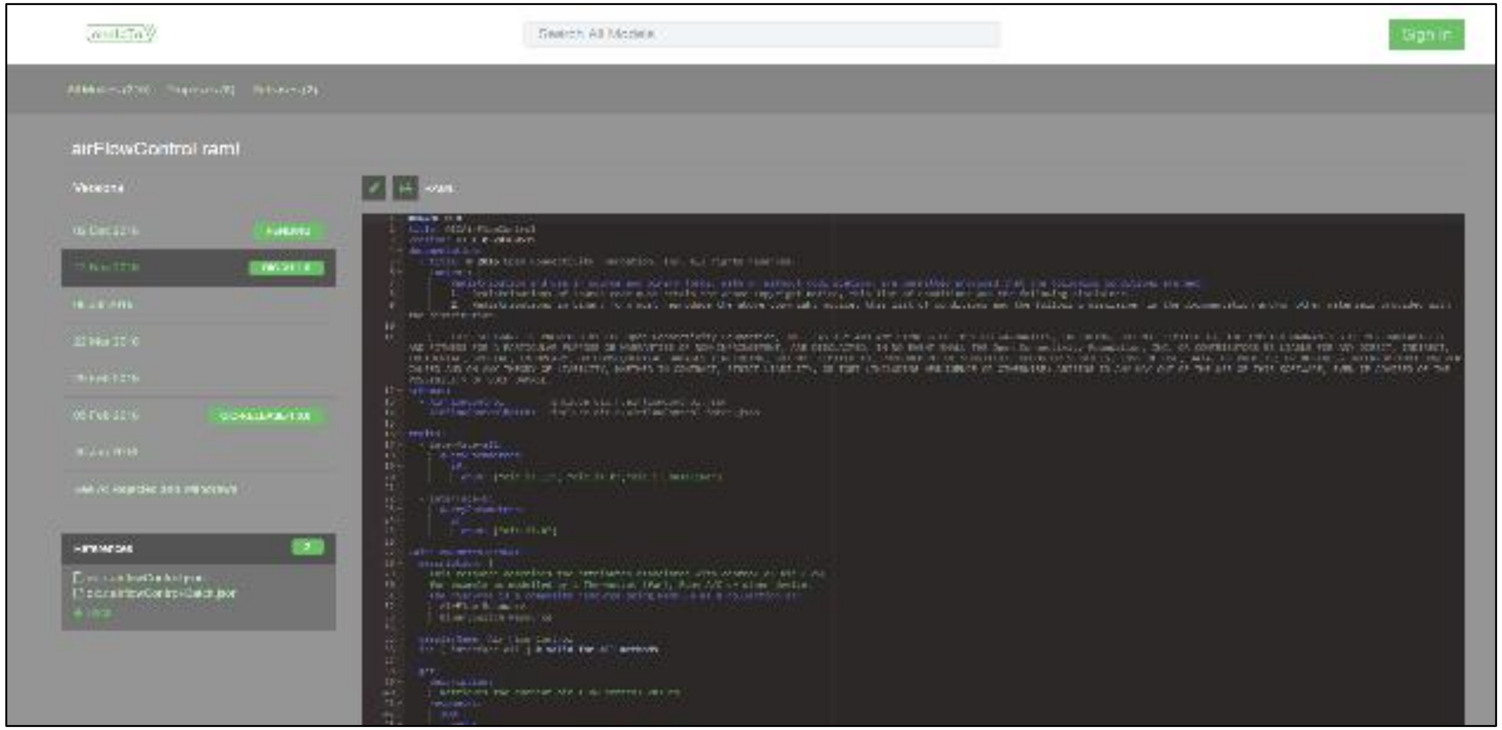
Determine your resources

- Sensors
  - Analog inputs (x4)
  - Digital inputs (x4)
  - Touch sensors (x8)
- Actuators
  - Digital outputs (x4)
  - Motor outputs (x2)

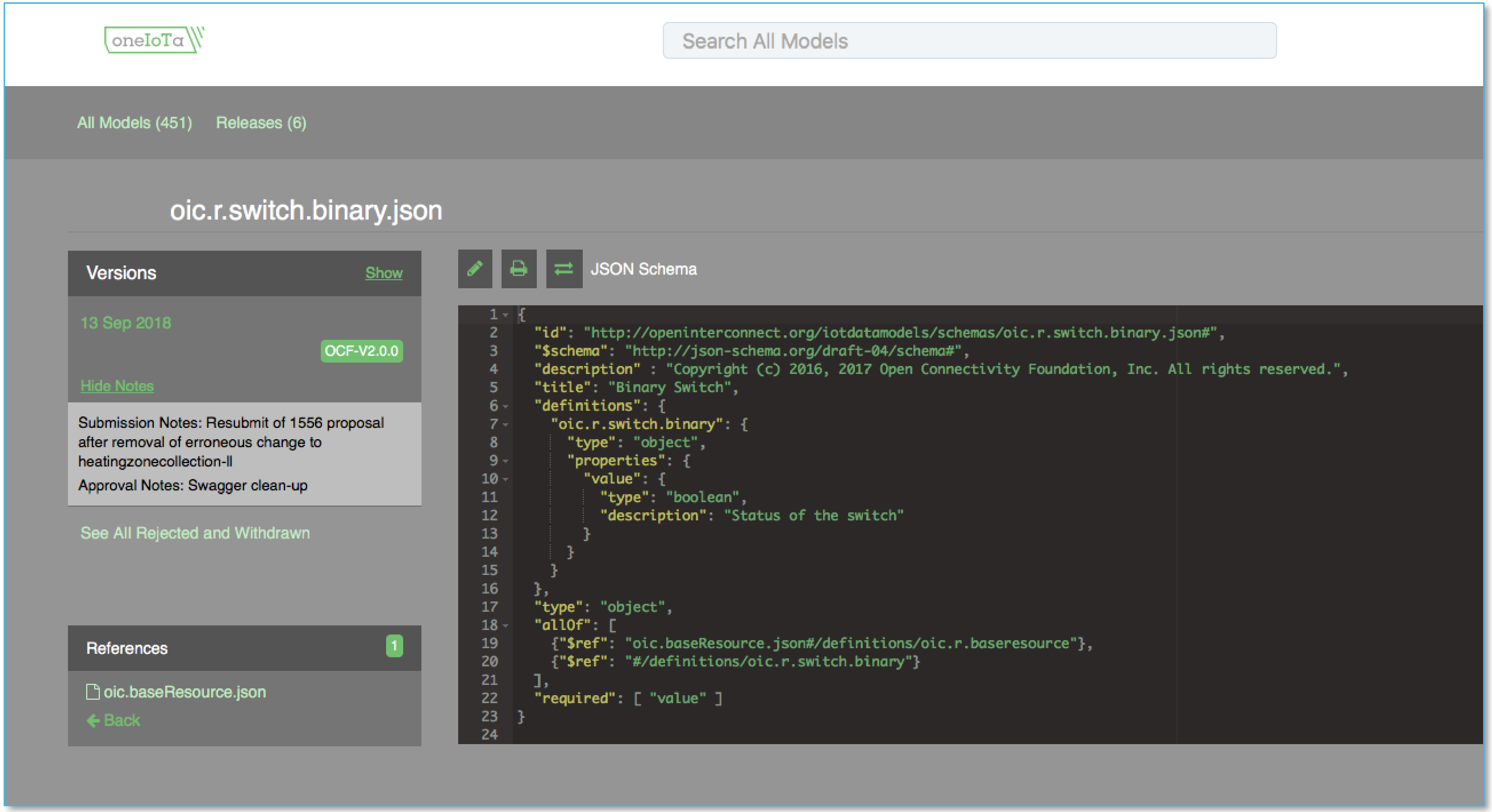


Which resources are in oneIoTa (oneiota.org)?

- Sensors
  - Analog inputs (x4)
    - oic.r.energy.electrical
  - Digital inputs (x4)
    - oic.r.switch.binary
  - Touch sensors (x8)
    - oic.r.switch.binary
- Actuators
  - Digital outputs (x4)
    - oic.r.switch.binary
  - Motor outputs (x2)
    - Not in oneIoTa yet



Example: oic.r.switch.binary



Create the input file for DeviceBuilder (example abridged)

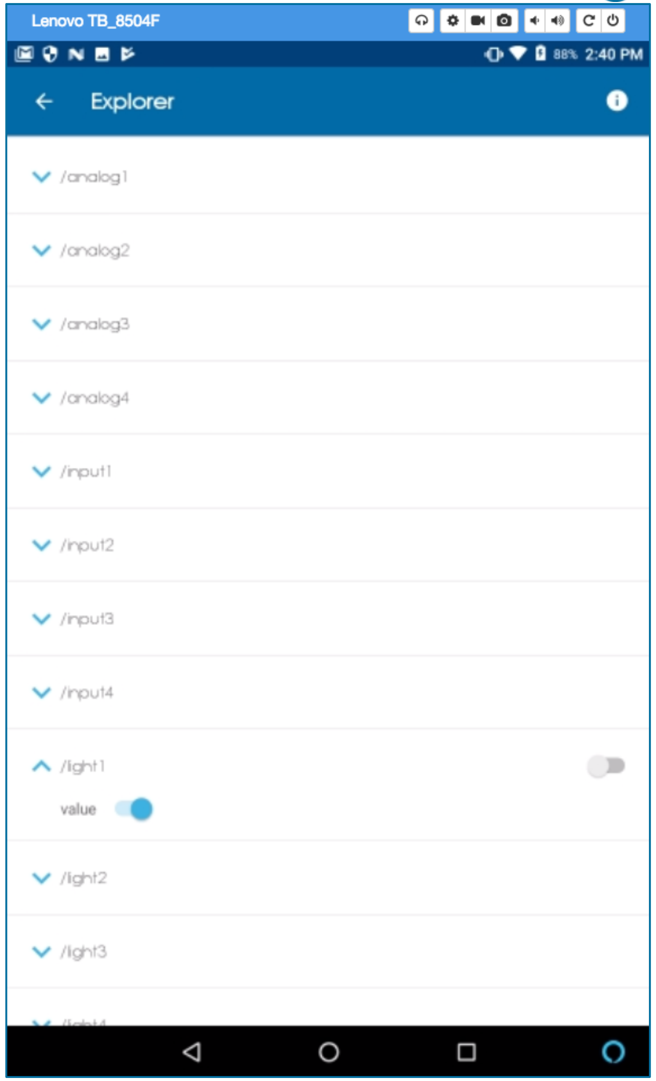
```
[
  {
    "path" : "/touch1",
    "rt"   : [ "oic.r.sensor.touch" ],
    "if"   : ["oic.if.baseline", "oic.if.a" ],
    "remove_properties" : [ "range", "step" , "id", "precision" ],
    "remove_methods" : ["post"]
  },
 {
    "path" : "/analog1",
    "rt"   : [ "oic.r.energy.electrical" ],
    "if"   : ["oic.if.baseline", "oic.if.s" ],
    "remove_properties" : [ "range", "step", "value" , "id" , "precision"],
    "remove_methods" : ["post"]
  },
  {
    "path" : "/light1",
    "rt"   : [ "oic.r.switch.binary" ],
    "if"   : ["oic.if.baseline", "oic.if.a" ],
    "remove_properties" : [ "range", "step" , "id", "precision" ]
  },
  {
    "path" : "/oic/p",
    "rt"   : [ "oic.wk.p" ],
    "if"   : ["oic.if.baseline", "oic.if.r" ],
    "remove_properties" : [ "n", "range", "value", "step", "precision", "vid"  ]
  }
]
```

Run the tool chain

- Automatically generate the code stubs, introspection file and security files (NOTE: a code stub will be built for each resource in input file):
  - gen.sh
- Build the project executable:
  - build.sh
- Set the security to "ready for owner transfer method" (RFOTM):
  - reset.sh
- Run the server code on the Raspberry Pi:
  - run.sh

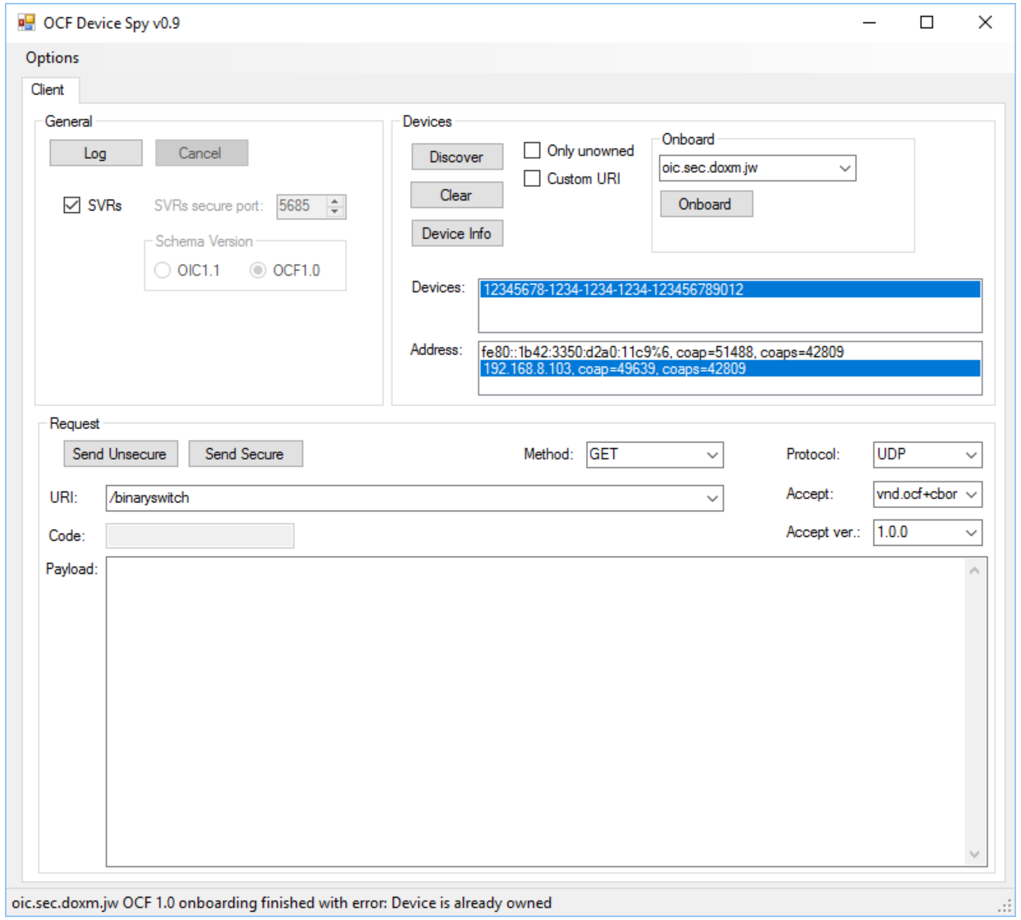
Onboard and control the server with OTGC

- Install OTGC on an Android device (make sure you're on the right LAN):
  - (download and run the APK or install procedure for your platform)
  - Launch the OTGC application
- Click the discover button to search for OCF devices on the LAN
  - Arrow in circle icon
- Onboard the discovered server
  - "+" icon associated with the server device
- Get the UI to control the Raspberry Pi server from the Android OTGC
  - Gear icon



Testing with Device Spy (as an alternative to OTGC)

- Device Spy is a lower level client that allows you to construct the actual payloads that are sent. It is only available on Windows.
  - Discover the device by clicking the [Discover] button
  - Onboard the device by clicking SVRs, selecting an address and clicking the [Onboard] button
  - Inspect the resource payload by setting the resource URI, selecting the GET method and clicking the [Send Secure] button
  - Change the value of the resource (e.g. false to true) by selecting the POST message and clicking the [Send Secure] button
  - The state of the light should change accordingly



Add your source code to integrate with hardware

```
static void
get_analog1(oc_request_t *request, oc_interface_mask_t interfaces, void *user_data)
{
        (void)user_data;  // not used

        // TODO: SENSOR add here the code to talk to the HW if one implements a sensor.
        // the call to the HW needs to fill in the global variable before it returns to this function here.
        // alternative is to have a callback from the hardware that sets the global variables.
        myParamArgs[0] = 1;
        CallPythonFunction((char *)"explorer-hat-pro", (char *)"readAnalog", 1, myParamArgs);
        g_analog1_voltage = returnDouble;

        // The implementation always return everything that belongs to the resource.
        // this implementation is not optimal, but is functionally correct and will pass CTT1.2.2

        PRINT("get_analog1: interface %d\n", interfaces);
        oc_rep_start_root_object();
        switch (interfaces) {
      .
      .
      .
```

Add your source code to integrate with hardware

```
static void
post_output2(oc_request_t *request, oc_interface_mask_t interfaces, void *user_data)
{
        (void)interfaces;
        (void)user_data;
        bool error_state = false;
        PRINT("post_output2:\n");
        oc_rep_t *rep = request->request_payload;
        .
        .
        .
        // TODO: ACTUATOR add here the code to talk to the HW if one implements an actuator.
        // one can use the global variables as input to those calls
        // the global values have been updated already with the data from the request
        myParamArgs[0] = 2;
        myParamArgs[1] = g_output2_value ? 1 : 0;
        CallPythonFunction((char *)"explorer-hat-pro", (char *)"writeLight", 2, myParamArgs);

        oc_send_response(request, OC_STATUS_CHANGED);
        .
        .
        .
```

Other tasks for a real product

- Debug server to control your hardware and use DeviceSpy to verify messages between client and server
- Test with Onboarding Tool and Generic Client (OTGC)
- Use the open source code from OTGC to build your own client or work with a productized client from another vendor
- Test your product (in-house) using the Compliance Test Tool (CTT)
- Certify your product with the CTT at an Authorized Test Lab (ATL)
  - This will get you the OCF certification logo and validate that you are compliant with the specifications

OPEN CONNECTIVITY FOUNDATION®