OPEN CONNECTIVITY FOUNDATION™

AUTOMATED DEVELOPMENT FOR CROSS-PLATFORM INTERNET OF THINGS

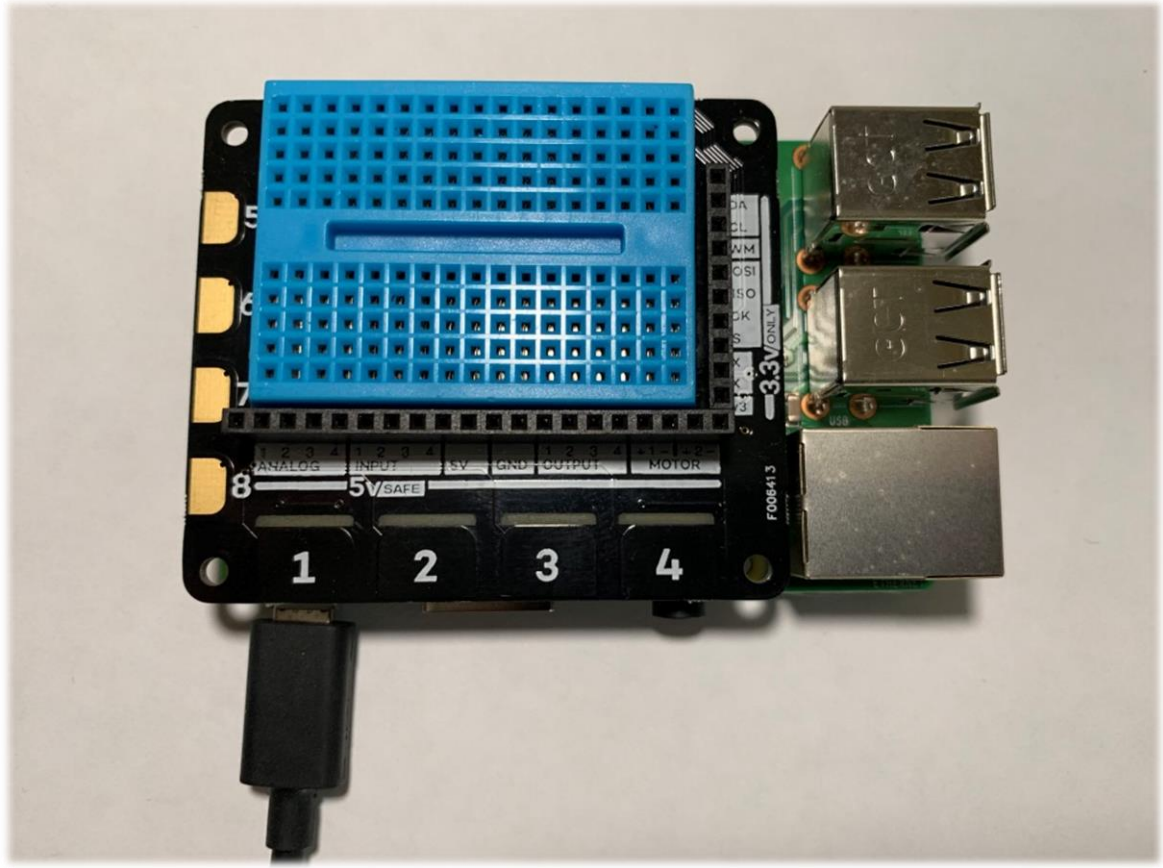
Develop a secure, certified hardware prototype in 15 minutes

Things you'll need:

- Raspberry Pi board (included in kit)
- Explorer Hat board (included in kit)
- Power supply (included in kit)
- USB thumb drive with instructions (included in kit)
- Micro SD card (included in kit)
- Keyboard and monitor (not included)
  - Or, use SSH on a separate computer to connect and control the Raspberry Pi, instead of a keyboard and monitor attached to the Raspberry Pi… instructions for both to follow

Set up the Hardware (Explorer Hat)

- Install the ExplorerHat board on the GPIO header on the Raspberry Pi board
- Insert the microSD card
- Connect the network (automatic)
- Plug in the power
- Connect keyboard and monitor (or SSH)



Boot and Connect to the Raspberry Pi

- Get a terminal view of the Raspberry Pi
  - Connect a monitor, keyboard and (optional) mouse to the Raspberry Pi, or
  - Connect from your computer using SSH
    - You can ping the Raspberry Pi (ping <your host name>.local) to verify that the Raspberry Pi is available
      - If it is taking a long time, try "sudo killall –HUP mDNSResponder" on MacOS
    - MacOS
      - Use the terminal program and "New Remote Connection
    - Windows
      - Use Putty on (or similar)
    - SSH to the Raspberry Pi using "ssh://pi@<your host name>.local"
- Default login: user: pi, password: raspberry

Set up the development environment (Network)

- This method will work you are on a network
- Install the development environment
  - Install all of this from the home directory: cd ~
  - IoTivity-lite development: curl https://openconnectivity.github.io/IOTivity-Lite-setup/install.sh | bash
  - Project scripts: curl https://openconnectivity.github.io/Project-Scripts/install.sh | bash
    - In case of an error, please rerun this command
  - Raspberry Pi examples (answer "y" to all the prompts): curl https://openconnectivity.github.io/Sample-Raspberry-Pi-Code/pi-boards/install.sh | bash
  - Make sure the PATH is set: reboot or source ~/.bashrc

Let's build a device (page 1)

- Create a directory for development (this can be anywhere, but we'll use the following):
  - cd ~
  - mkdir workspace
  - cd workspace
  - Create an OCF project (can be named anything, but we'll use the following):
    - create_project.sh myexample
    - cd myexample
  - We'll use a pre-built sample to start:
    - cp ~/Sample-Raspberry-Pi-Code/IoTivity-lite/explorer-hat-pro/setup.sh ./
    - ./setup

Let's build a device (page 2)

- Automatically generate the code, introspection file and security files:
  - gen.sh
- Build the project executiable:
  - build.sh
- Set the security to "ready for owner transfer method" (RFOTM):
  - reset.sh
- Run the server code on the Raspberry Pi:
  - run.sh
- You have successfully build an OCF device and it is ready to onboard!

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 get it from the OCF USB stick)
  - 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
  - Use the UI to turn on and off any of the lights on the ExplorerHat board
  - Use the AI to turn on "observe" on any of the switches, then watch the terminal as you press the button on the ExplorerHat board
- <u>You have successfully onboarded and controlled your OCF Device!</u>
  - In case of any issues onboarding the OCF Device, please make sure it is in the Ready For Ownership Transfer Method (RFOTM) provisioning state by using the reset.sh script

What we did

- Started with an OCF configuration file:
  - edit_config.sh
- Created an OCF input file:
  - edit_input.sh
- Created the server code, introspection file and sercurity files:
  - edit_code.sh
- Generated the source code
  - gen.sh
- Built the application
  - build.sh
- Reset the security status (to unowned)
  - reset.sh
- Run the server device
  - run.sh
- Used OTGC to discover, onboard and control the device
  - OTGC

Here's how you can do it on your own (page 1)

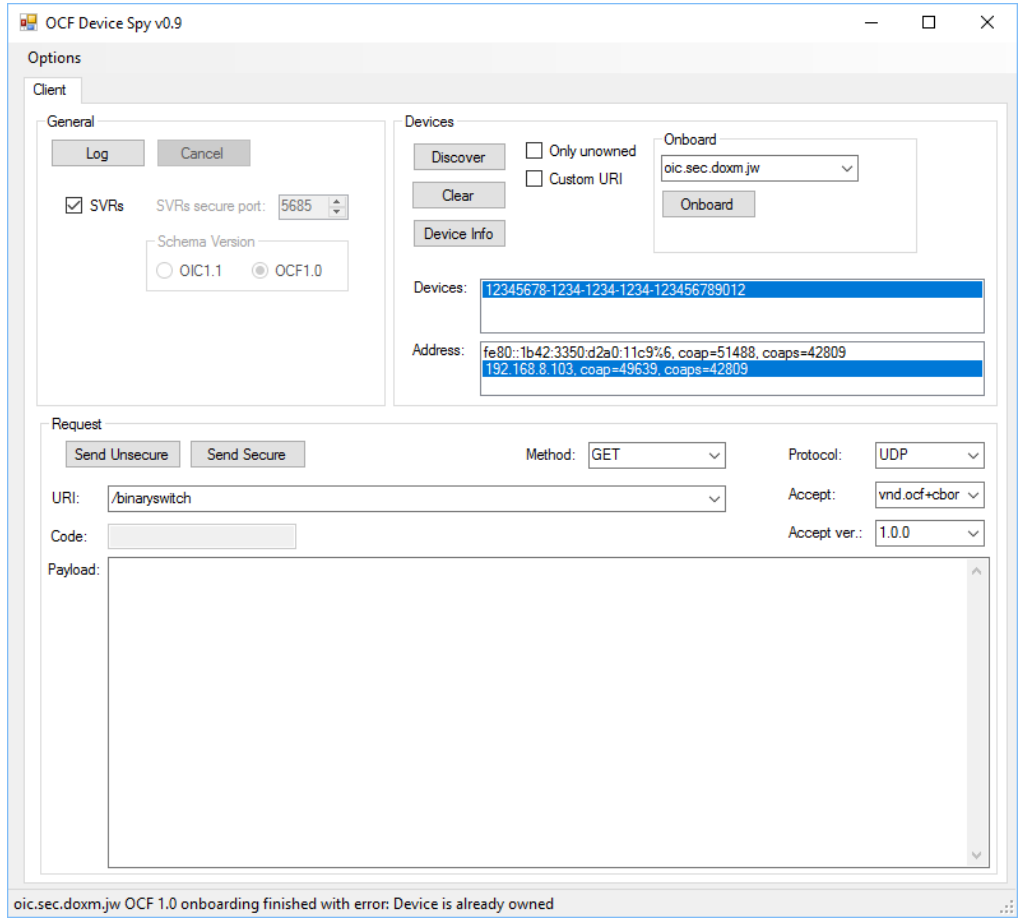
- Create a new OCF project (can be named anything, but we'll use the following):
  - cd ~/workspace
  - create_project.sh mytestproject
  - cd mytestproject
- Edit the configuration file for your specific device:
  - edit_config.sh
- Automatically generate the code, introspection file and security files (this usually only needs to be done once unless the config file is changed):
  - gen.sh

Here's how you can do it on your own (page 2)

- Use the same development loop as before, but edit the code as necessary:
  - Edit the code:
    - edit_code.sh
  - Build the project executiable:
    - build.sh
  - Set the security to "ready for owner transfer method" (RFOTM):
    - reset.sh
  - Run the server code on the Raspberry Pi:
    - run.sh
  - Run OTGC and control the server on the Raspberry Pi

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
    - In case of any issues onboarding the OCF Device, please make sure it is in the RFOTM provisioning state by using the reset.sh script.
  - 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



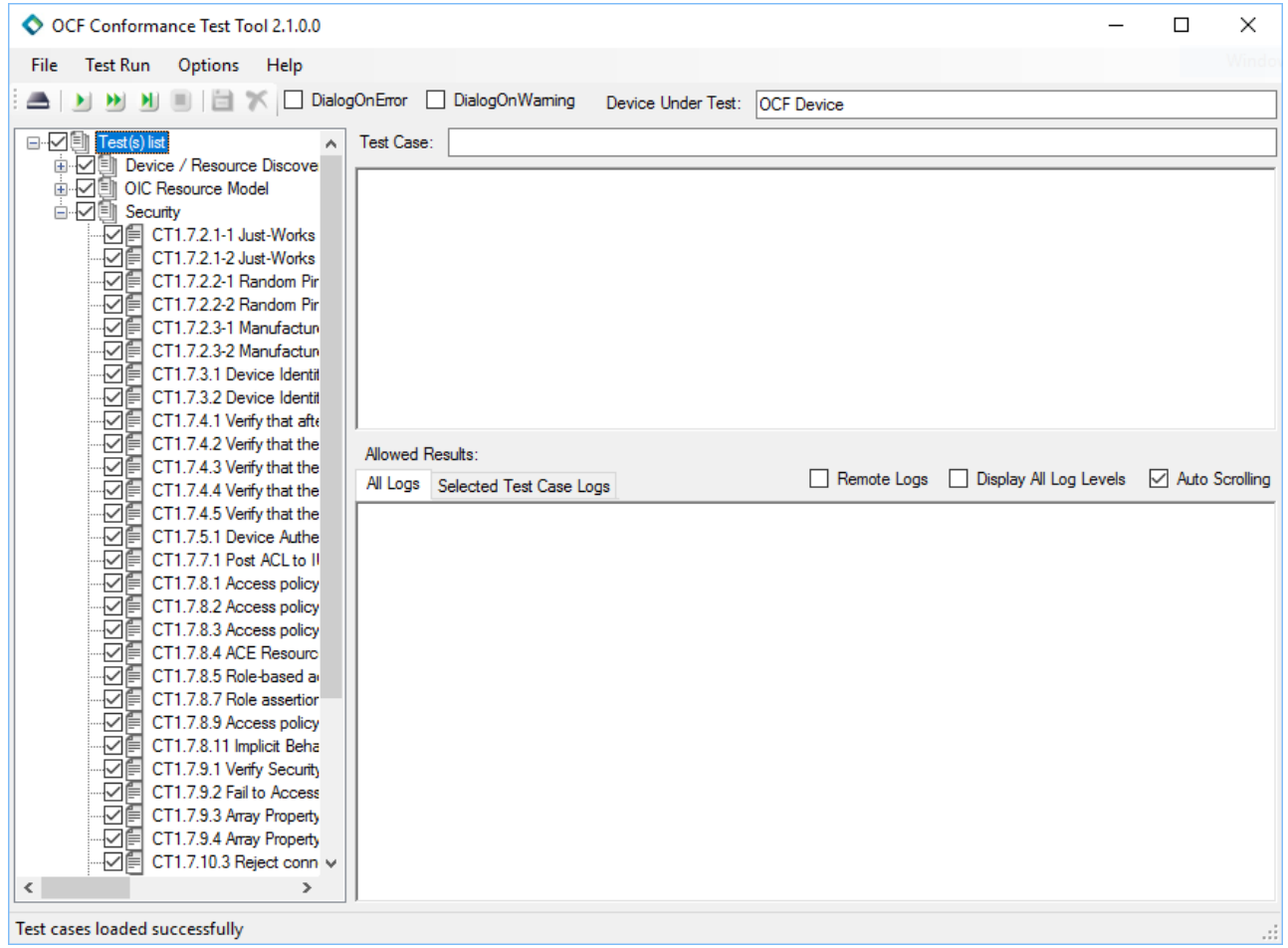
Simple device description input file

```
[
   {
     "path" : "/binaryswitch",
     "rt"   : [ "oic.r.switch.binary" ],
     "if"   : ["oic.if.a", "oic.if.baseline" ],
     "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 Certification Test Tool

- The Certification Test Tool is and automated test too that devices must pass to get OCF certification
  - Run the CTT
  - Select the PICS file generated by gen.sh
  - Select the discovered device and the interface to use
  - Click the play button to start the tests
  - Go get coffee
  - But don't take too long because you will need to change the onboarding state to RFOTM a few times
  - Pass the CTT and work with a certified lab to get official certification.

Available to OCF members



What to do next

- Define your own input file
- Run the tooling!
- Build it… and see if it works
- Change the code, to what you want it to do

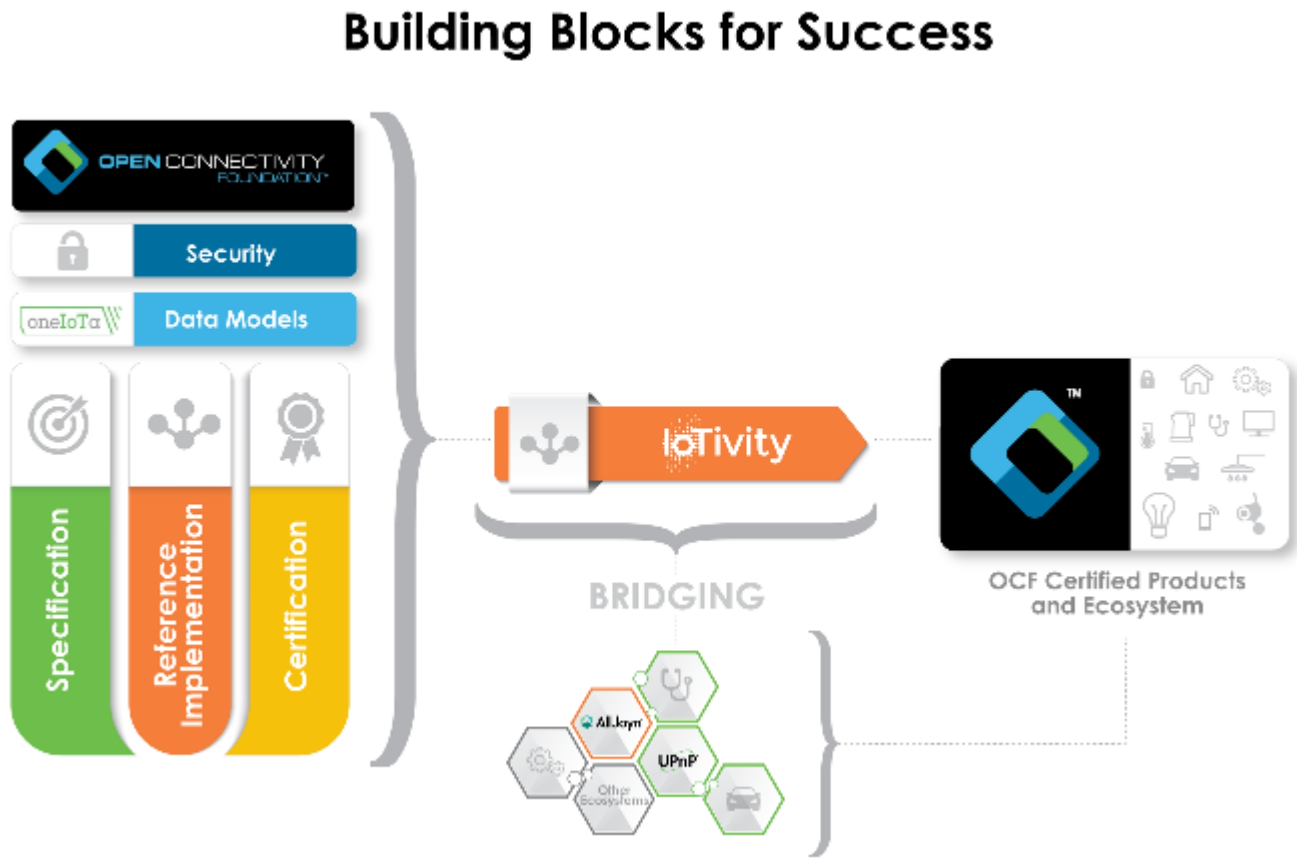
What else?

- The DeviceBuilder can be modified for any programming language using templates
- The OTGC is available on Android, iOS, Linux and Windows soon
- The OTGC is available as open source, so you can use a product that is already OCF certified as the basis for your own client tool on multiple operating systems

Interested to know more about OCF, IoTivity and oneIoTa? Read on…

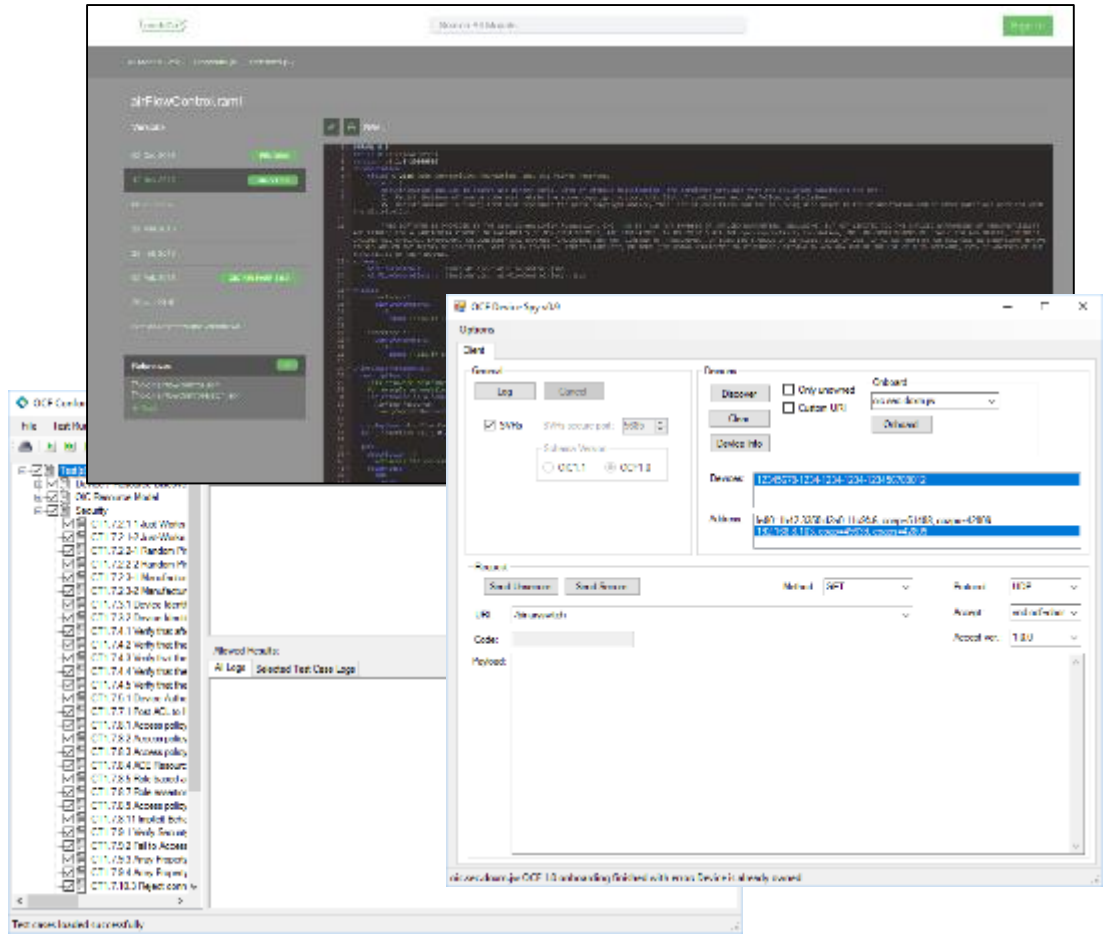
What is the Open Connectivity Foundation?

- Standards body for interoperable IoT
  - Strong security built in from the start
  - Works interoperably with existing ecosystems
  - Three pillar alignment
    - International specification
    - Open source implementation
    - Automated certification tool and international authorized test labs
  - Flexible, RESTful, data-model-based architecture



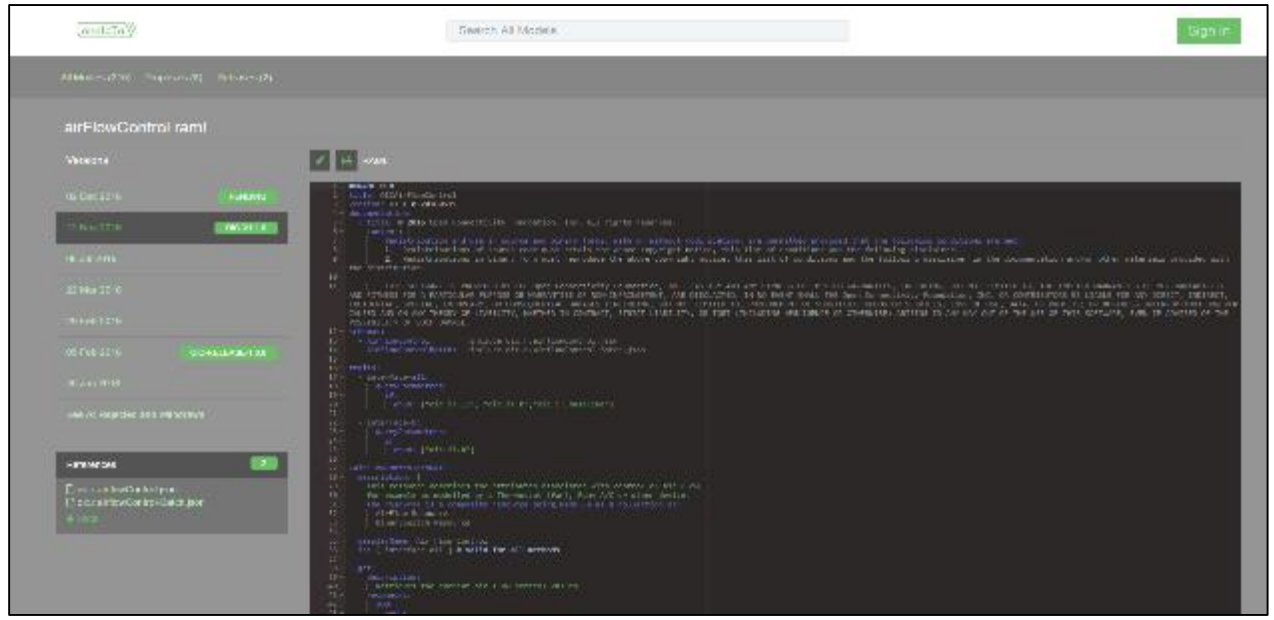
Complete suite of development tools

- Online tool for crowd-sourcing data models and building ecosystem interoperability
- Data models drive everything
- Automated tool chain creates
  - Server source code
  - Introspection file
  - Automated test case file
  - Secure onboarding file
  - Client tool user interface
- Several test and development tools



oneIoTa (oneiota.org)

- oneIoTa is an IDE and process management tool for the resource data models at the center of OCF.
  - OCF atomic resource models are entered in the editor windows in oneIoTa
  - More complex resources can be composed of atomic resources
  - Completed resources are submitted to an approval process for OCF (or partner organizations
  - Other organizations use the same approval process in oneIoTa
  - Mappings between OCF and other ecosystems are also entered in oneIoTa
  - Resource models are used to build specifications, devices, source code, GUIs, PICS files and bridges



oneIoTa is the source of all resources you will need

- Go to http://www.oneiota.org
- Search for any resources you will need
- Use these descriptions when defining your device description file
- All devices consist of a list of resources
- All tools will consider oneIoTa as the authoritative source of resources

OPEN CONNECTIVITY FOUNDATION™