OPEN CONNECTIVITY FOUNDATION®

ToolChain

Code generation

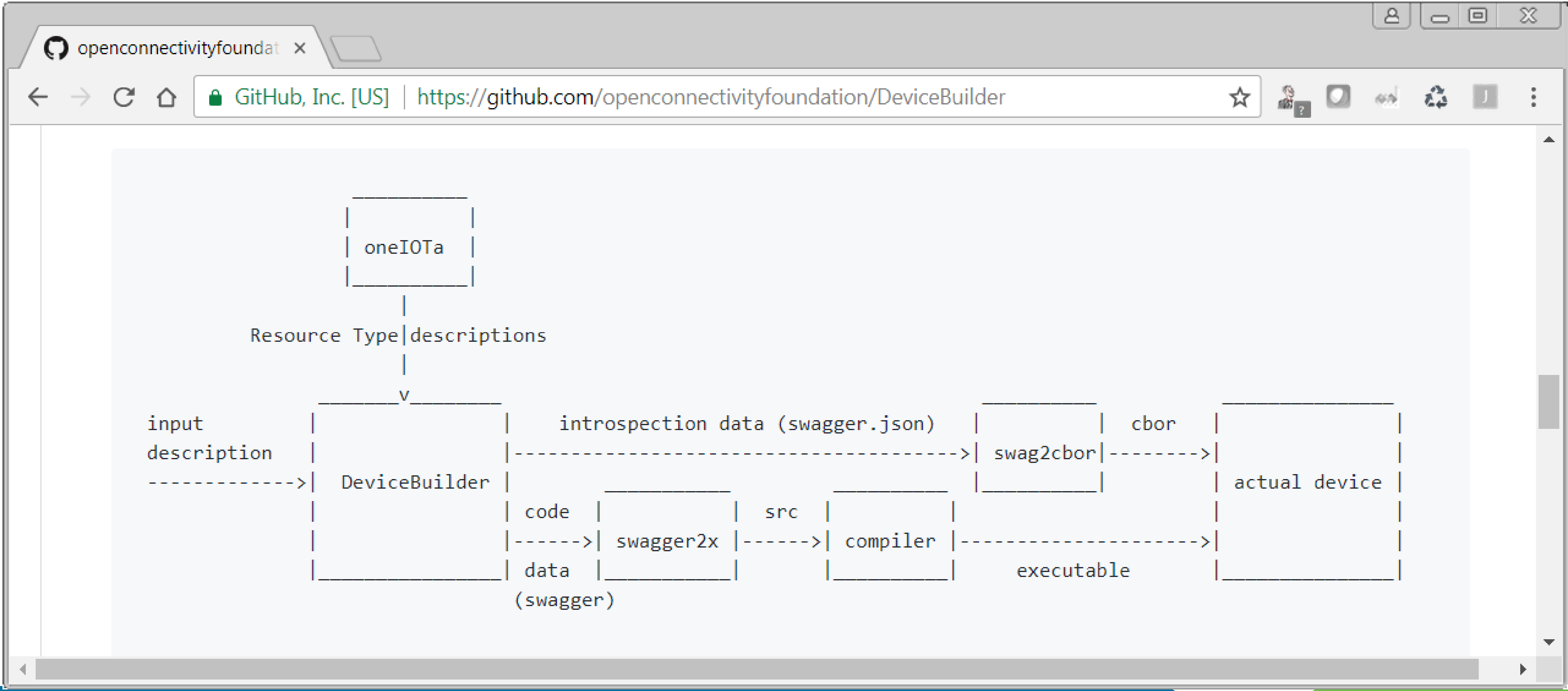
Code generation

Code generation is possible due to :

All Resources are defined in an machine readable format

- Using OpenAPI 2.0 definitions
- Defines resource by:
  - Which operations are supported (RETRIEVE, UPDATE, ..)
  - Schema definition of the payload per operation
  - An example of the payload
  - Define which query parameters are applicable

The DeviceBuilder tool chain



```
                  __________
                 |          |
                 | oneIOTa  |
                 |__________|
                      |
         Resource Type|descriptions
                      |
               _______v________                                          __________           _______________
input         |                |    introspection data (swagger.json)   |          |  cbor   |               |
description   |                |---------------------------------------->| swag2cbor|-------->|               |
------------->|  DeviceBuilder |        ___________         __________  |__________|         | actual device |
              |                | code  |           |  src  |          |                      |               |
              |                |------>| swagger2x |------>| compiler |--------------------->|               |
              |________________| data  |___________|       |__________|     executable       |_______________|
                                (swagger)
```

The DeviceBuilder toolchain

- The DeviceBuilder tool chain consist of a set of python scripts glued together with bash scripts.
- The script installs the necessary depended info it needs:
  - Git repos
  - Install python3 packages using pip3
- The chain works on Linux and on Window (using git bash).
- The chain will be used on a raspberry pi

DeviceBuilder

- Python script to merge the resources to be used into a single file.
- Takes an set of resources that will be used in the implementation
  - Combines swagger resource files into 1 large swagger file
    - Renames paths and removes properties while processing.
  - Does this for optional core resources and all resources in oneIOTa.
    - Mandatory resources (core and security) are not processed.

- Input: file with resources to be implemented
- Output: swagger file that list all “application level” resources
  - All resources that the application needs to implement to function correctly.

swagger2x

- Create code by means of template technology
  - Can used jinja2 template technology because OpenAPI 2.0 == JSON
  - Add a bit of glue so that JSON constructs can be converted into correct code
  - Jinja2 constructs allows to loop over the DOM of the swagger file
    - Convert by means of instructions in template all the info of the swagger file into code
    - The "clever bit" is in the template, it is an mix of the target code and template language
    - An template is targeted for an specific code base
      - Multiple templates can co-exist, hence the tool can support all languages/APIs
      - Target: C IOTivity-Lite API.

Introspection generation

json2cbor

- Tool to convert json to cbor (and visa versa)
- Used as first step to create cbor of the introspection file

cbor2include

- Utility to create include files from cbor
- Only used to convert the cbor introspection file into an include file

Note:

- Include file is updated when code generation is done.
  - Hence it will always reflect the implementation
  - It will be updated in the tree, hence each build will have an updated introspection device data.

Single script

Single bash script uses the python scripts to create code

Input:

- Input file with the wanted resources
- Output directory
- Which device type is being implemented

For example:

```
sh DeviceBuilder_IotivityLiteServer.sh ../input-lightdevice.json ../lightdevice "oic.d.light"
```

What do you get?

After running the DeviceBuilder script:

- Output Directory with:
  - Code currently based on C API of IOTivity-Lite 1.3.1
  - Introspection file that matches the implementation
    - Available in JSON and CBOR (converted from JSON)
  - Initial just works security file
    - Just a copy of an existing file

Problem: how to build this code

Build environment: IOTivity-Lite-setup

Github repo that uses IOTivity-Lite v1.3-rel as build environment and sets up the device builder tool chain to build an application

- This github repo sets up an "work area"
  - Installs tools/code/etc to build an IOTivity-Lite server application
    - Linux
    - raspberry pi
  - This includes installing a copy of the IOTivity-Lite code

Github to install the environment resides at:

- https://github.com/openconnectivity/IOTivity-Lite-setup

IOTivity-Lite-setup github repo

The repo has setup scripts for:

- Setting up IOTivity-Lite with code and build environment
  - IOTivity version: 1.3-rel
  - Sets it up in local folder
- Setting up DeviceBuilder
  - Relative to IOTivity-Lite folder
- Setting up MRAA to interact with the hardware

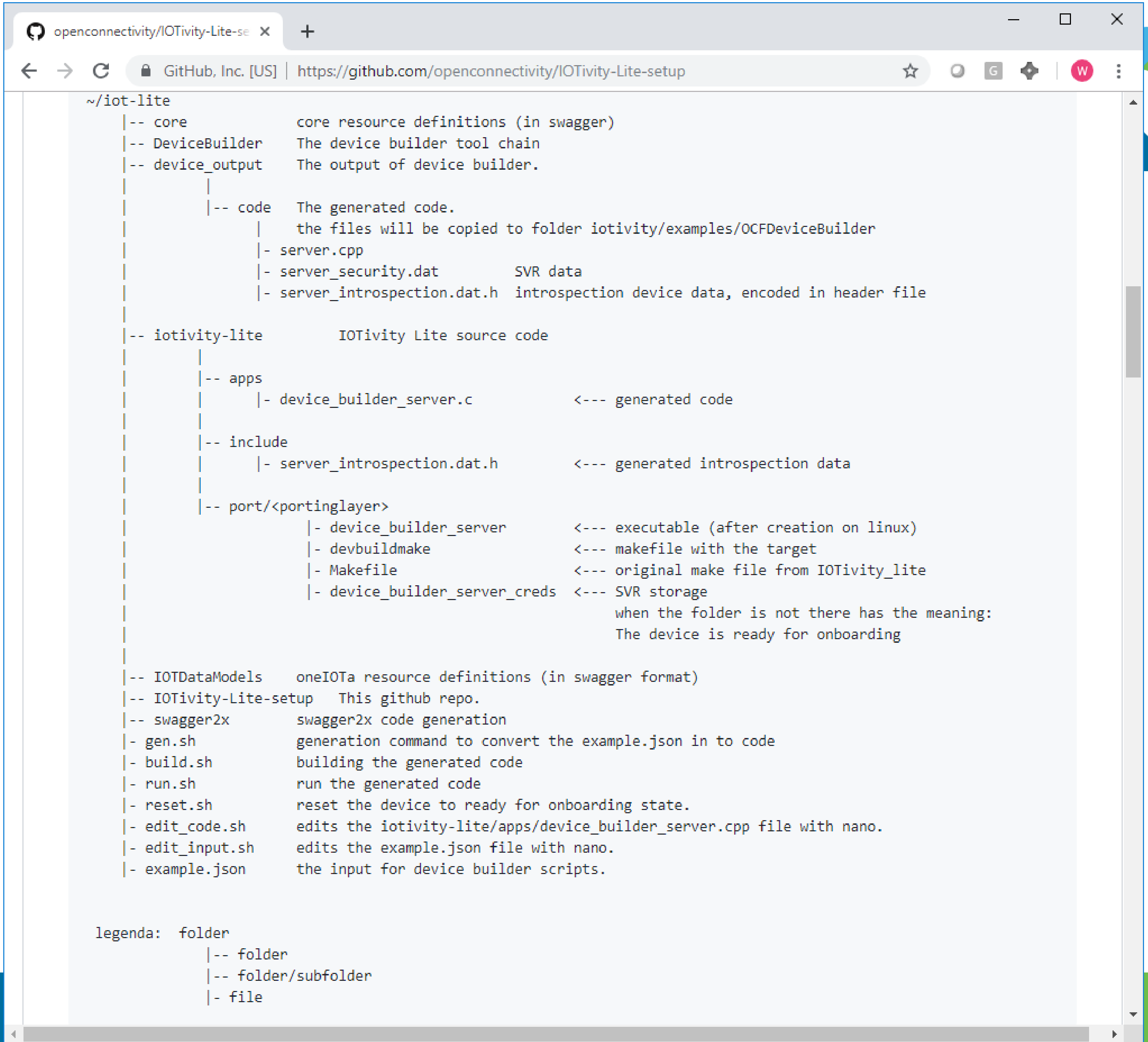
Everything is set up by executing:

```
curl https://openconnectivity.github.io/IOTivity-Lite-setup/install.sh | bash
```

Folder structure

All folders are placed in the top folder

In this case ~/iot-lite



openconnectivity/IOTivity-Lite-se
GitHub, Inc. [US]
https://github.com/openconnectivity/IOTivity-Lite-setup

```
~/iot-lite
    |-- core             core resource definitions (in swagger)
    |-- DeviceBuilder    The device builder tool chain
    |-- device_output    The output of device builder.
    |         |
    |         |-- code   The generated code.
    |               |    the files will be copied to folder iotivity/examples/OCFDeviceBuilder
    |               |- server.cpp
    |               |- server_security.dat         SVR data
    |               |- server_introspection.dat.h  introspection device data, encoded in header file
    |
    |-- iotivity-lite         IOTivity Lite source code
    |        |
    |        |-- apps
    |        |      |- device_builder_server.c            <--- generated code
    |        |
    |        |-- include
    |        |      |- server_introspection.dat.h         <--- generated introspection data
    |        |
    |        |-- port/<portinglayer>
    |                     |- device_builder_server        <--- executable (after creation on linux)
    |                     |- devbuildmake                 <--- makefile with the target
    |                     |- Makefile                     <--- original make file from IOTivity_lite
    |                     |- device_builder_server_creds  <--- SVR storage
    |                                                          when the folder is not there has the meaning:
    |                                                          The device is ready for onboarding
    |
    |-- IOTDataModels    oneIOTa resource definitions (in swagger format)
    |-- IOTivity-Lite-setup   This github repo.
    |-- swagger2x        swagger2x code generation
    |- gen.sh            generation command to convert the example.json in to code
    |- build.sh          building the generated code
    |- run.sh            run the generated code
    |- reset.sh          reset the device to ready for onboarding state.
    |- edit_code.sh      edits the iotivity-lite/apps/device_builder_server.cpp file with nano.
    |- edit_input.sh     edits the example.json file with nano.
    |- example.json      the input for device builder scripts.


 legenda:  folder
              |-- folder
              |-- folder/subfolder
              |- file
```

Preconfigured commands (in iot-lite folder)

- Generate code with gen.sh
- Compile code with build.sh
- Edit code with edit_code.sh (using nano, editing in the iotivity-lite tree)
- Run code with run.sh
- Reset to onboarding state with reset.sh
  - Using just works, script removes files from the iotivity-lite tree.

The scripts that are convenience wrappers around command line tools.

- All paths, etc. are filled in.
- Code is generated from example.json.
- Code is generated in iotivity-lite/apps

Example input file: binary switch

```
[   {
      "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"  ]
    } ]
```

Path to be used

rt as look up value

Which interfaces are supported

Which properties you do not want

Needed for introspection

OPEN CONNECTIVITY FOUNDATION®