Exosite

Boot Camp 2015

Device API Code Labs

Device and API Code Labs

Pre-lab todo:

1. Install Python 2.7.x on your computer and verify it is working using terminal window ($ python --version)
   https://www.python.org/downloads/

2. These labs also make use of Exosite's web application 'Portals'. You can sign-up for a free developer account here if you haven't already: https://portals.exosite.com
3. These labs require a 'Device' client in your Portal account with a couple of dataports. If you haven't already, log-in to Portals, https://portals.exosite.com and create a device or use the one you already have created from One Platform labs:
   a. Click the 'Devices' menu item on the left
   b. On the Devics table page (/manage/devices) click '+Add Device' on the right top side of the table.
   c. Type is generic, Timezone whatever you are in now, location not needed (meta only).
   d. Call your device 'My Device' or something that will help you reference it.
   e. After creating your device, click on it to open it's Device Information window. Click 'Add Data' link. Add three dataports with the following parameters:
      i. Name: Input
      ii. Type: Integer
      iii. Alias: input

      i. Name: Output
      ii. Type Integer
      iii. Alias: output

      i. Name: Control
      ii. Type Integer
      iii. Alias: control

Notes:
These labs are meant to build upon each other.
Lab Code can be found here: https://github.com/maanenson/exo_bootcamp_2015_march

Device / API Code Labs

Lab 1 - Basic HTTP Read / Write
The first Device API to look at is the HTTP REST calls to Write and Read data.

This API documentation can be found here: http://docs.exosite.com/http/

1. Get the script 'http_write.py'.
2. Try to run it as is using your code editor or command line:
   a. $ python http_write.py
3. What was the HTTP Response? Can you find that HTTP response in this list: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
4. Why do you expect you got an error response?

5. Now open 'http_write.py' using a code editor.
6. Fill in the string for your device client's CIK where it says "PUT_YOUR_DEVICE_CIK_HERE".
7. Now run the script using the code editor or using command line:
   a. $ python http_write.py
8. What was the response code? Did it match the API documentation for a 'Write'. Check in Portals (or use Exoline 'read' command) to check your 'input' dataport to verify you got a 1.

9. Get the script 'http_read.py' and open it using a code editor.
10. Fill in the string for your device client's CIK where it says "PUT_YOUR_DEVICE_CIK_HERE".
11. Now run the script using the code editor or using command line:
    a. $ python http_read.py
12. What was the response code and returned value? Is this what you expected based on the current value of 'output'?

You've now made a HTTP Client request to the Server application (Exosite's One Platform)! This is the basis for Device communication to Exosite's platform. When starting to write the application on a device that handles communicating with Exosite, these are usually the first two things to verify working.

Advanced users (BONUS)

13. Modify http_write.py to write to 2 dataports at the same time. You may need to create another dataport in Portals first (output2 for example).
14. Modify http_write.py (from previous lab) to use the hybrid read/write option to both write to the 'input' dataport and read from the 'output' dataport with one call.

Lab 2 - Wait on a value (long-polling)
For interactions where a device needs to make a decision based on a dataport, it has an option to read it over a given period, typically called ‘polling’. For applications where you want the device to not have to continuously poll, say every second or two, there are ways to ‘long-poll’, which is a way for the device to request the data, but the server will not respond until a new value is available.

This lab uses the HTTP GET method to read, adding new headers to tell the server we want to use long-polling. http://docs.exosite.com/http/#long-polling

1. Get the script ‘http_long_poll.py’ and open it using a code editor.
2. Fill in the string for your device client’s CIK where it says “PUT_YOUR_DEVICE_CIK_HERE”.
3. Now run the script using the code editor or using command line:
    a. $ python http_long_poll.py
4. You should notice that there is no response yet
5. Using ‘Exoline’ or Portals, write a value to your dataport ‘output’
    a. $ exoline write <DEVICE CIK> output --value=1
6. Did you get a response?
7. Change the timeout value in the code from 60000 to 2000 and run it again.
8. Did you get a timeout response?

Devices can take advantage of this if they have proper interrupt handlers and methods to deal with a delayed response.

Lab 3 - Use the RPC API to make a read request

The JSON RPC API has more functionality than the HTTP POST/GET data interface used in the previous steps. The JSON RPC allows for all functions available in the platform including read, write, create, info, list, etc. In addition, many of these functions have multiple options, like reading multiple values back for a dataport instead of just the last value.
The JSON RPC can be used by Applications (Users, business integration tools, etc) but can also be used for devices that need certain functionality.

This lab uses the Read RPC procedure: http://docs.exosite.com/rpc/#read

1. Get the script rpc_read.py' and open it using a code editor.
2. Fill in the string for your device client's CIK where it says "PUT_YOUR_DEVICE_CIK_HERE".
3. Now run the script using the code editor or using command line:
    a. $ python rpc_read.py
4. Look at the response, can you see the value you expected for the dataport 'output'?
5. Change the code to return the last 5 values and run it again.

Lab 4 - Use the RPC API to make send a log of values/timestamps
Some device applications require the device to log data and send it later on, perhaps every 24 hours. The RPC includes a ‘record’ function that allows for writing values at provided timestamps.

This lab uses the Record RPC procedure: http://docs.exosite.com/rpc/#record

1. Get the script ‘rpc_record.py’ and open it using a code editor.
2. Fill in the string for your device client’s CIK where it says “PUT_YOUR_DEVICE_CIK_HERE”.
3. Now run the script using the code editor or using command line:
    a. $ python rpc_record.py
4. Look at the request, what is the format for the data?
5. Use Exoline to see if you got these values:
    a. $ exo read <DEVICE CIK> output --limit=20 --timeformat=unix

What do you think happens if the RPC call attempts to write data at the same timestamp twice?

Lab 5 - Comparing CoAP and HTTP
Exosite has a CoAP API that has some benefits over HTTP, namely that it uses binary format over the transport and can use a lot less code space to implement on your device, which is critical for embedded microcontrollers.

This lab will help to show the different in the transport size for a HTTP request vs a CoAP request to the server. For cellular networks, this can widely affect your required data plan.

1. Get the script ‘http_and_coap_write.py’ and open it using a code editor.
2. Fill in the string for your device client’s CIK where it says “PUT_YOUR_DEVICE_CIK_HERE”.
3. Now run the script using the code editor or using command line:
    a. $ python http_and_coap_write.py
4. Look at the sizes for the request and responses. Is there a major difference?

Note that this lab code is only looking at the size of the requests and responses, but does not actually measure overhead of the transport layer (http over tcp / coap over udp) which may use more data based on that transports protocol. To get a better idea of how the actual data use differs you can use Wireshark to compare the total data use at a lower lever.

Lab 6 - Using CoAP for Reads and Writes

CoAP as a protocol was designed to be somewhat similar to HTTP. That means that Exosite was able to create a new API that is very similar in structure to our existing HTTP API. The requests in this lab should seem very similar to the requests made in lab 1.

1. Get the script ‘coap_write.py’ and open it using a code editor.
2. Fill in the string for your device client’s CIK where it says “PUT_YOUR_DEVICE_CIK_HERE”.
3. Now run the script using the code editor or using command line:
    a. $ python coap_write.py
4. Use Exoline to see that the new value has been written:
    i. $ exo read <DEVICE CIK> output

Lab 7 - Using CoAP Observe for Real-Time Push Notifications
CoAP as a protocol was designed to be somewhat similar to HTTP. That means that we were able to create a new API that is very similar in structure to our existing HTTP API.

1. Get the script ‘coap_observe.py’ and open it using a code editor.
2. Fill in the string for your device client’s CIK where it says “PUT_YOUR_DEVICE_CIK_HERE”.
3. Now run the script using the code editor or using command line:
    a. $ python coap_observe.py
4. You will immediately get a response with the current dataport value.
5. Using ‘Exoline’ or Portals, write a value to your dataport ‘output’
    a. $ exoline write <DEVICE CIK> output --value=1
6. You should see that you immediately get a notification that the value of the dataport has changed.
7. Write another value to the dataport (repeat step 9).
8. You should again see that you get a notification that the value of the dataport has changed using the same session.