Exosite

Boot Camp 2014

One Platform Labs

One Platform Labs

Pre-lab todo:

1. Install Exoline and verify it is working using terminal window ($ exo --help)
   https://github.com/exosite/exoline
2. These labs also make use of Exosite's web application 'Portals'. You can sign-up for a free developer account here:
   https://portals.exosite.com

Notes:

These labs are meant to build upon each other.

Lab Code can be found here: https://github.com/maanenson/exo_bootcamp_2014

Lab 1 - Platform Clients / CIKs

Examine a pre-created client and get used to Exoline.

Steps:

1. Get a CIK to examine at: http://exocamp.webscript.io/democik
2. Examine the client using Exoline info command:
   a. $ exo info <Provided CIK> --pretty
3. Examine it's resources using Exoline tree command:
   a. $ exo tree <Provided CIK>
4. Examine it's resources using Exoline 'twee' command:
   a. $ exo twee <Provided CIK>
5. Let's read one of it's resources using Exoline read command:
   a. $ exo read <Provided CIK> sensor --follow
   b. note: to break out of this use ctrl-C key combination

Can you pick out the different parameters that a One Platform Client has? Examples: meta, name, usage, etc.

Lab 2 - Create your own device client

Create a new device client to examine it. During this lab, user will see what a client is made up of in the platform and see the hierarchy.

1. Sign into https://portals.exosite.com (create account if doesn't already exist)
2. Click 'Devices' menu item, then click 'Add a device' on the Devices table.
    a. Use 'I want to create a generic device' option
    b. Device Type: generic
    c. Timezone: (select whatever you want)
    d. Location: Not necessary, this goes into meta field
    e. Name: My Device
3. Add the device and open it's information window to copy it's CIK
4. Use the Exoline info command to view the client make-up.
    ```
    $ exo info <Device CIK> --pretty
    ```
5. Use the Exoline tree command to view any resources under the client.
    ```
    $ exo tree <Device CIK>
    ```
6. Go to the User Account page (https://portals.exosite.com/account/portals and copy your Portal CIK listed under the Portal name as 'Key'.
7. Use the Exoline 'tree' command to see how the device is a client underneath the Portal account 'client'.
    a. `$ exo tree <Portal CIK>`

Lab 3 – Resources/Dataports – Write/Read some data using Exoline
Using the device client created in the previous step, user will create dataport resources and interact with them using API.

1. Using the device you created in the previous lab, use Portals to add a dataport called 'Input Data' under your Device.
   Name: Input Data
   Alias: input
   Type: integer
2. On the same device, create a dataport called 'Output Data' in Portals.
   Name: Output Data
   Alias: output
   Type: integer
3. In Exoline, use the command: $exo tree <Device CIK>
4. You should see your two dataports listed.
5. In Exoline, use the command: $exo twee <Device CIK>
6. You should see same info but a nicer format and last values for your data.
7. Let's use Exoline now to write some data to the dataport named 'Input Data'.
   a. Exoline command: $exo write <Device CIK> input 20
8. Now, let's examine the device client again using the 'twee' command. $exo twee <Device CIK> Do you see a value of 100 next to the dataport with alias 'input'?

Lab 4 - Resources/Datarules - Dispatch an alert message
Using the device client created in the previous step, user will create datarules and dispatches (Events / Alerts).

1. Using the device you created in the previous lab, use Portals to create an event called ‘Trigger High’. Start by clicking on the ‘Events’ menu item in Portals and find the ‘+Add Event’ link.
   Reference Data Source: ‘Input’
   Event Name: Trigger High
   Alias: e_triggerhigh
   Condition Type: Simple
   Comparison: Greater Than (>)
   Constant: 50

2. Examine this new resource under the client using the exoline tree command
   ```
   $exo tree <Device CIK>
   ```
   Notice this Event has created a ‘datarule’ resource under the client.
3. Now, let’s attach an dispatch to this datarule. In Portals, on that same Events page, click ‘+Add Alert’.
   Reference Event: ‘Trigger high’
   Alert Name: Trigger High Alert
   Alert Interval: 0 (make sure to use ‘0’)
   Event Action: Email
   Recipient: (Your Email)
   Subject: Triggered High Event!
   Message: This event was just triggered
4. Examine this new resource under the client using the exoline tree command
   ```
   $exo tree <Device CIK>
   ```
   Notice this Event has created a ‘dispatch’ resource under the client.

5. Examine the state of the datarule using exoline:
   ```
   $exo read <Device CIK> e_triggerhigh
   ```
6. Write data to the ‘input’ dataport:
   a. `$exo write <Device CIK> sensor 30`
7. Examine the state of the datarule again using exoline:
   ```
   $exo read <Device CIK> e_triggerhigh
   ```
8. Write data to the ‘input’ dataport again, to trigger it to go high:
   a. `$exo write <Device CIK> sensor 70`
9. Examine the state of the datarule again using exoline, you should see it with a value of ‘1’ meaning active:
   ```
   $exo read <Device CIK> e_triggerhigh
   ```
10. You should have received an email from this dispatch that was set up.

Let’s clean up after this lab by deleting the alert dispatch that you created here. This will make sure you are not sent emails while we try out some other features. Go to the Events page in Portals, click on your Alert to open the pop-out information window. At the bottom is a ‘delete’ box, use this to remove the alert.

Lab 5 - Lua Script in the Platform - Hello World!
Before we get into doing things with the Lua scripts, let's just try it out first. There is another type of datarule called 'scripts' that users can insert Lua scripts into and provide advanced processing and alerting features. [Lua -> http://www.lua.org/about.html]

Script API: http://docs.exosite.com/scripting/

1. Using the device you created in the previous labs, use Portals to create a script called 'hello world'. Start by clicking on the 'Scripts' menu item in Portals and find the '+Add Script' link on the scripts table.
    Reference Device: <Your Device -> My Device
    (make sure to not select your Portal, as you can run scripts at portal level.)
    Script Name: hello world
    Example Script: (none)

2. Examine this new resource under the client using the exoline tree command
    a. $exo tree <Device CIK>
    b. Notice this Script has created a 'datarule' resource under the client with format of 'string'.
3. Grab the source code for 'hello_world.lua' and copy into the Lua editor in Portals.
4. Click 'Update' in the script editor window to save it.
5. Watch the debug window in the script editor to say 'Hello World'.
6. The script should cause an error, can you determine why?
7. Fix the error and re-run the script by clicking 'Update'.
8. Note the state of the script, is it running or not?

Lab 6 - Resources/Scripts - Create an advanced alert handler

The datarule and dispatch that were created showed other the other two resources in an Exosite platform client besides dataports. There is another type of datarule called 'scripts' that users can insert Lua scripts into and provide advanced processing and alerting features. [Lua -> http://www.lua.org/about.html]

Script API: http://docs.exosite.com/scripting/

Steps:

1. Using the device you created in the previous labs, use Portals to create a script called 'Advanced Alert Handler'. Start by clicking on the 'Scripts' menu item in Portals and find the '+Add Script' link on the scripts table.

   Reference Device: <Your Device -> My Device

   (make sure to not select your Portal, as you can run scripts at portal level.)

   Script Name: Advanced Alert Handler

   Example Script: (none)

2. Examine this new resource under the client using the exoline tree command
   a. $exo tree <Device CIK>
   b. Notice this Script has created a 'datarule' resource under the client with format of 'string'.
3. Grab the source code for 'advanced_alert_handler_1.lua' and copy into the Lua editor in Portals.
4. Update the string emailaddress with your own email address
   a. format: "username@company.com"
5. Click 'Update' in the script editor window to save it.
6. Watch the debug window in the script editor to say 'Running'.
7. Let's examine this resource, use Exoline to see what it looks like:
   a. exo info <DEVICE CIK> <SCRIPT RID> --pretty
   b. *Note: Portals does not assign an 'alias' to the script dataport resource, so you need to use the scripts RID. You can get this from the exo tree command ran above.*
8. Write a new value in your dataport to trigger the waiting script:
   a. exo write <DEVICE CIK> input --value=50
9. Now write another value that we know will trigger the event condition we are looking for:
   a. exo write <DEVICE CIK> input --value=95
10. Verify you got an email
11. You can real-time see the script debug output also by using exoline to read the resource. Try to run the following command in a new terminal window using exoline and then run the previous two steps to have the script trigger:
    a. exo read <DEVICE CIK> <SCRIPT RID> --follow

b. *Note: debug messages for scripts are flushed out in bulk every ~10 seconds, not immediately, so even though you may see a delay in the debug output, it does not affect the execution speed of the script.*

You’ve now created your first Exosite platform script!

How could you expand this to save your ‘state’ of the condition you are looking for and prevent sending many alerts when the state hasn’t changed if more data comes in that continues to be higher than your trigger value?

Can you modify the email message to include the time?

Lab 7 - Resources/Scripts - Create a data processing handler
Lua scripts run inside of the platform open up the door to real-time processing of data. This lab will show how to create an example script that processes the data as it comes in and saves a new value to another dataport.

Script API: http://docs.exosite.com/scripting/

1. Using the device you created in the previous labs, use Portals to create a script called ‘Data Processing’. Start by clicking on the ‘Scripts’ menu item in Portals and find the ‘+Add Script’ link on the scripts table.
   Reference Device: <Your Device -> My Device>
   (make sure to not select your Portal, as you can run scripts at portal level.)
   Script Name: Data Processing
   Example Script: (none)

2. Grab the source code for ‘data_processing_lua_script.lua’ and copy into the Lua editor in Portals.
3. Click ‘Update’ in the script editor window to save it.
4. Watch the debug window in the script editor to say ‘Running’.
5. Let’s examine this resource, use Exoline to see what it looks like:
   a. exo info <DEVICE CIK> <SCRIPT RID> --pretty
   b. *Note: Portals does not assign an ‘alias’ to the script dataport resource, so you need to use the scripts RID. You can get this from the exo tree command ran above.*
6. Using a second terminal window, run the following command line to watch the ‘output’ dataport:
   a. exo read <DEVICE CIK> output --follow
7. Write a new value in your dataport to trigger the waiting script:
   a. exo write <DEVICE CIK> input --value=500
8. Watch as the ‘output’ dataport is updated

Lab 8 - Scripts can manage and interact with client elements also
The scripting interface can not only interact with client resources like read and write but also can create resources and provide information about clients. This next lab will run a script that will examine information about the device client, which can be used by the scripts to perform necessary actions, and email you it's findings.

Script API: http://docs.exosite.com/scripting/

1. Using the device you created in the previous labs, use Portals to create a script called 'Device Information Report'. Start by clicking on the 'Scripts' menu item in Portals and find the '+Add Script' link on the scripts table.
   Reference Device: <Your Device -> My Device
   (make sure to not select your Portal, as you can run scripts at portal level.)
   Script Name: Device Information Report
   Example Script: (none)

2. Grab the source code for 'device_information_report.lua' and copy into the Lua editor in Portals.
3. Click 'Update' in the script editor window to save it.
4. Watch the debug window in the script editor to follow what it is doing.
5. This script should send you an email with information about your device client. Compare this emailed report with what you see with the following Exoline commands:
   a. $exo info <Device CIK> --pretty
   b. $exo tree <Device CIK>
6. For Advanced Users - Notice how messy the report is because of the Lua Table / JSON string creation (json.encode). Can you try to add functions to make the JSON information (device info and device listing) 'pretty'?
   a. Tip: Search for Lua Table Serialization or Lua Pretty Table, etc

Lab 9 - Create a Script that will emulate data
Scripts can write to dataports, thus making them able to make decisions based on other data and write values for the devices to read or for applications. In this case, we'll create a script that will emulate creating data, which is useful when starting to develop User Interfaces with data in Exosite and you do not need to rely on having an actual device up and running.

1. Using the device you created in the previous labs, use Portals to create a script called 'Emulate Data'. Start by clicking on the 'Scripts' menu item in Portals and find the '+Add Script' link on the scripts table.
   Reference Device: <Your Device -> My Device
   (make sure to not select your Portal, as you can run scripts at portal level.)
   Script Name: Emulate Data
   Example Script: (none)

2. Grab the source code for 'create_and_generate_emulated_data.lua' and copy into the Lua editor in Portals.
3. Click 'Update' in the script editor window to save it.
4. Watch the debug window in the script editor to follow what it is doing.
5. Run the exoline twee command to see if it created the new dataports:
   a. $exo twee <Device CIK>
6. Let's see if it's actually creating data by using Exoline commands:
   a. $exo read <Device CIK> temperature --follow

How does the script create a delay to update new emulated data?

Lab 10 – Sharing Data/Devices
The Exosite platform supports ‘sharing’ of resources, both publically and privately, this includes clients and resources like dataports.

1. Using the device you created in the previous lab, use Portals to add a dataport called ‘Input Data’ under your Device. Go to the ‘Data’ page and click ‘+Add Data’.
2. Select ‘From Public Data Source’.
3. Search for ‘training’, you should find one called ‘Mikes Public Data’. Select this dataport.
4. Make sure to specify ‘Assign To Device’ as your device (not your portal).
5. Specify an alias of ‘shareddata’.
6. Using Exoline, run the following command:
    a. `$ exo read <DEVICE CIK> shareddata --follow`
    b. Wait for the instructor to write some data …
7. You can look up information about this about this dataport:
    a. `$ exo info <DEVICE CIK> shareddata`

_ _ Create a Private Share _ _

8. Now, share a private dataport. In Portals, open the information window for your ‘input’ dataport under your device. Use the ‘Share Data’ box and create a share. If you enter nothing in the email box, a share will be created, or you can enter another user’s email address and Portals will send them an email and make it available to them in the Add Data window.
9. In Exoline, the info command to examine the ‘input’ dataport again:
    a. `$exo info <Device CIK> input --pretty`
    b. Note it has a ‘Share’ listed underneath it.
    c. *Note it should also still have a ‘Subscriber’ which is your Event datarule looking for a high value.*
10. You should see a ‘Share’ now listed with a code and if another client has activated that share, it will show that client’s RID as ‘activator’.

Lab 11 - Inspect API calls

Use Exoline's -d option to see Debug HTTP request information on some calls to examine the API format and transport. Exoline uses Exosite's JSON RPC API, sent over HTTP. Exoline uses the Exosite python library to make these JSON RPC requests.

http://docs.exosite.com/rpc/
https://github.com/exosite-labs/pyonep

Run the following commands using your existing device from previous lab steps.

1. `$ exo -d info <DEVICE CIK>`
2. `$ exo -d read c28765ad55e810804ececb70a0be49aee5e2ed44 temperature input`
3. Can you find the RPC requests in each of these and their responses?