Cassia RESTful APIs for E1000, X1000, C1000 and S2000 Series

This document shows how you can use the Cassia RESTful APIs to integrate your Bluetooth LE devices with the Cassia Bluetooth routers, without requiring any changes to the Bluetooth end devices. Step-by-step instructions, RESTful APIs, debug tools, and basic troubleshooting procedures are included.

Table of Contents

1 Overview

The Cassia Bluetooth Router is the world's first long-range Bluetooth router designed for enterprise deployments, enabling seamless coverage of any size and scale. It extends Bluetooth range up to 1,000 feet, open space, line-of-sight, and enables remote control of multiple Bluetooth Low Energy (BLE) devices without requiring any changes to the Bluetooth end devices themselves.

The Cassia Restful APIs were developed to enable third party developers and device manufacturers to utilize the Bluetooth routing and extended range capabilities of the Cassia Router while using their Cloud services to connect and control multiple BLE devices per Router simultaneously. Furthermore, the Cassia Restful APIs are designed to integrate directly into your application/server using an HTTP/HTTPS based communication protocol, which provides programming language flexibility. Cassia supports C#, Node.js, and Java, but you can choose other languages as you prefer. This document helps you learn how to use the Cassia Restful APIs and its associated services.

The Cassia Restful APIs for Cassia Bluetooth routers, is built into the Cassia IoT Access Controller (AC) providing the following functions:

a. Connect and control your BLE devices.
b. Support three modes: Scanning, Connecting, Broadcasting.
c. Write/read data to/from the BLE device via the Cloud server.
d. Read data as notification/indication events from the BLE device via the Cloud server.
e. Support multiple uplink networking interfaces to flexibly adapt to various environments.

2 Architecture Diagram

2.1 What is the Cassia Access Controller?

The Cassia IoT Access Controller (AC) is a powerful IoT network management solution. It provides RESTful APIs for the business to do data collection, positioning, roaming, and security policy management, enabling the remote control of Cassia Bluetooth routers across the Internet.

2.2 Cassia RESTful APIs Working Diagram

You can operate your BLE devices via the AC's access to the Cassia routers, using a set of RESTful APIs. The Cassia router is a Linux-based thin Access Point (AP). Please

see Figure 1 for the Cassia RESTful APIs Working Diagram, using X1000 as an example.

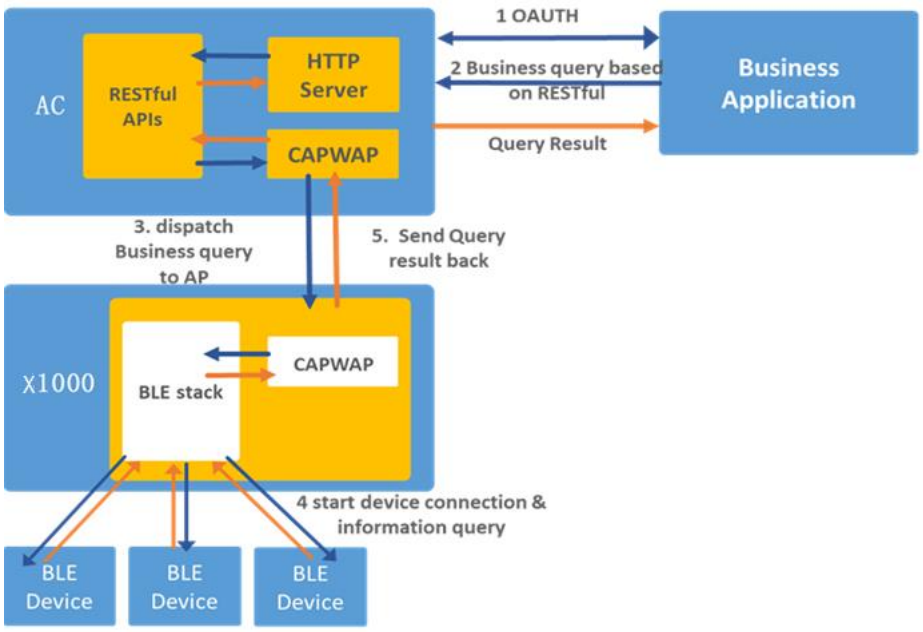


Figure 1: Cassia RESTful APIs Working Diagram

The business application initiates an OAuth authentication request (generated using its developer ID and secret) to the Cassia Access Controller first. Once the authentication succeeds, it will send an HTTP query based on Restful to the AC. Next, the AC dispatches the query to a corresponding Bluetooth router via encrypted CAPWAP. The router then executes the query upon the BLE devices, and passes the result back to the AC, and then the business application.

3 Prerequisites

Once you have the Cassia Bluetooth Router and AC, you will also need:

a) Developer account credentials

You will need a Developer Key and a Developer Secret. These developer credentials authorize the remote control of the Cassia Bluetooth router. To request your developer account credentials, contact the Cassia support team at support@cassianetworks.com or your sales representative.
Here is a sample credential:

```
client_id:tester, secret:198c776539c41234
```

b) Server license key

To use your Bluetooth Router with your AC, you will also need a server license key. The license key governs the number of Bluetooth routers that can be managed by the AC, for how long and features available. Contact the Cassia support team at support@cassianetworks.com or your sales representative. Here is an example of a license key:

```
v009-8q61-jy6a-l9fe-2gz8
```

c) Open UDP ports 5246 and 5247

- The ports that Cassia routers use to communicate with the Cassia Access Controller are UDP 5246 and 5247. If you have a Firewall that blocks those ports, please make sure to open them.
- Make sure that the CAPWAP UDP ports 5246 and 5247 are enabled and are not blocked by an intermediate device that could prevent a Cassia router from joining the IoT controller.
- If access control lists (ACLs) are in the control path between the IoT controller and its access points, you need to open new protocol ports to prevent access points from being stranded.

d) Understand the MAC address

- You can find your router's MAC address at the bottom of the router, as noted below in Figure 2.

Figure 2: Cassia Bluetooth router's MAC address

- If you are filtering MAC addresses in your security policy, please make sure to input the active MAC addresses. For example, if you are using WIFI for uplink connection, your active MAC will be Label_ MAC+1. See below table for the details.

Model	Label MAC	Ethernet MAC	WIFI MAC

C1000/X1000	MAC	MAC	MAC+1
E1000	MAC	MAC	MAC+1
S2000	MAC	MAC	MAC+1
S1000/S1100	MAC	MAC	MAC-1

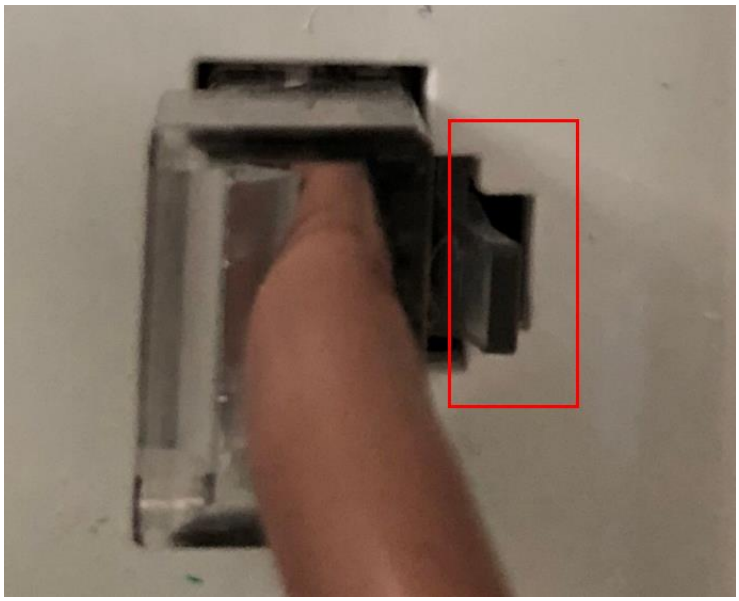
4 Getting Started

4.1 Step One: Associate your Bluetooth Router with the AC

If you have Ethernet connection and DHCP support, please follow the steps below. Otherwise, please refer to our Quick Start Guide to use WIFI hotspot function on the Cassia routers.

Plug in your Cassia router via Ethernet cable and power it on. Locate your IP address of the router from your network switch or DHCP server. Enter your IP address in a web browser and load web console. In Common section, please enter your AC Address as shown below in Figure 3.

Note: For C1000, please make sure to use a Cat5e cable to connect it to the Internet. For the rest of models, either Cat5e or Cat6 cable could work.



Plug a Cat6 cable to C1000 will leave a hole like this.

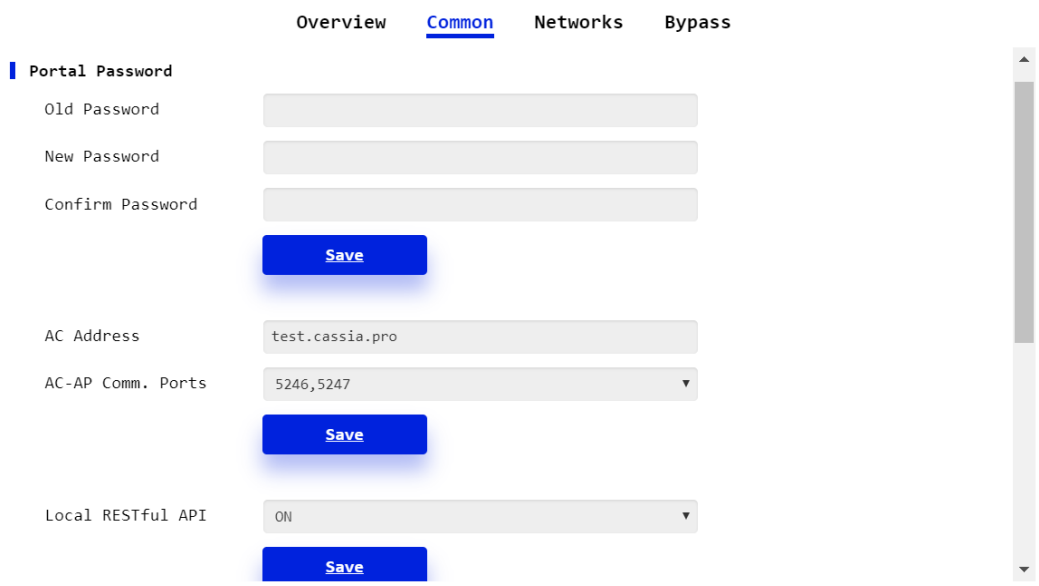


Figure 3: Input your developer key and AC Address

Note: You can skip this step if you are deploying your AC at the same subnet as your routers or using DHCP option43 or DNS for AC auto discovery.

4.2 Step Two: Access Cassia IoT Access Controller (AC)

You have the option to deploy an AC either in the Cassia Cloud, your own private cloud or a hardware appliance. For set up, login to the AC and navigate to the Settings page. As noted in the screenshot below, input your Developer Key, Developer Secret, and License Key. (Figure 4).

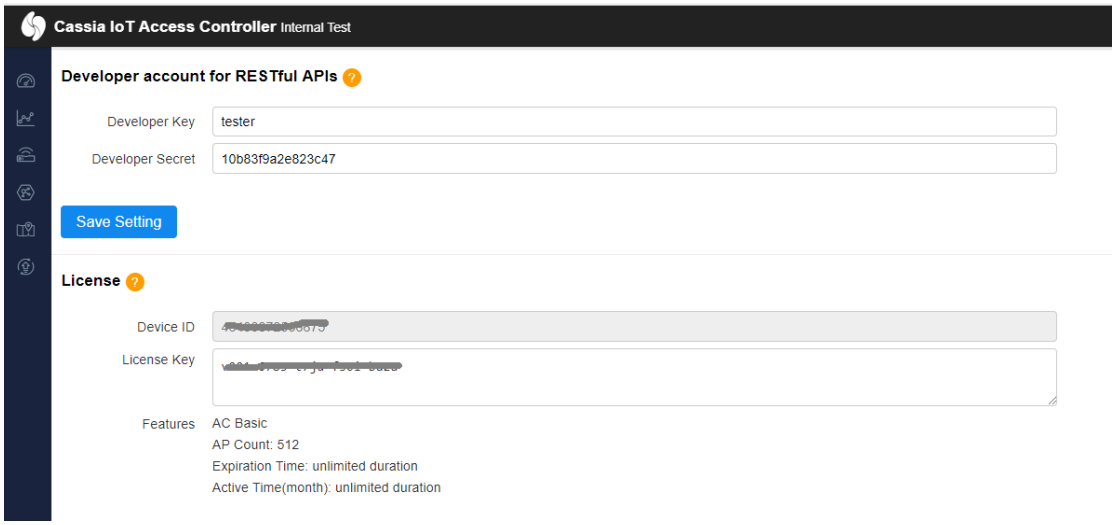


Figure 4: Input your developer key, secret, and license key

4.3 Step Three: Configure your Bluetooth Router

Navigate to the Routers page, as noted below in Figure 5.

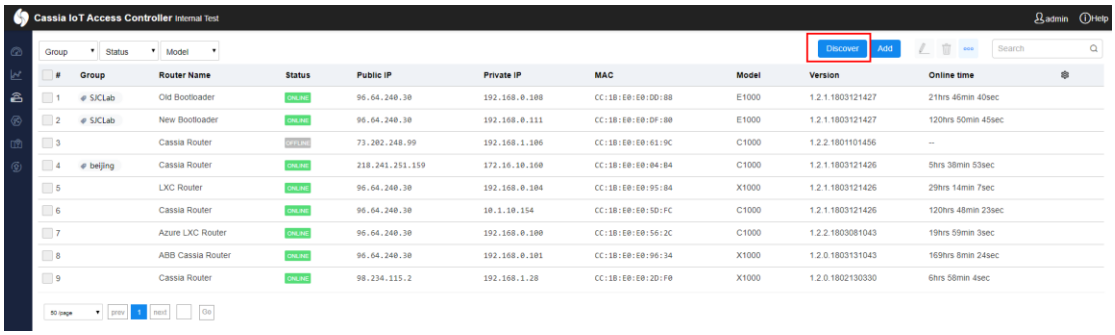


Figure 5: Cassia AC Routers Page

Search your router in the list using its MAC address. You can find your router's MAC address at the bottom of the router.

If your router is not listed in the Routers list, click the Discover button (near the top-right area of the interface), see Figure 6 to discover and add your router into the AC.

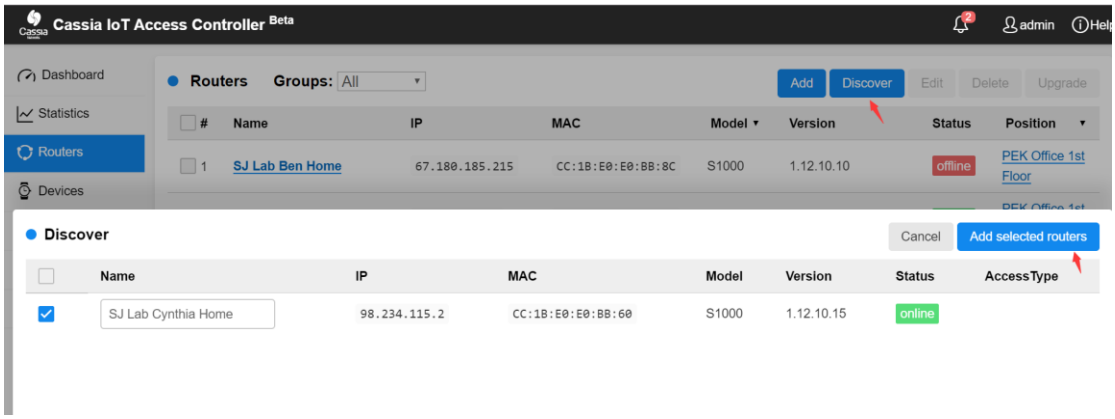


Figure 6: Discover your router and add it into the AC

4.3.1 Configure Common Settings

To configure your router, check the checkbox before it and click the Edit Button  to get to the router's configuration page, as seen in Figure 7. You can give a nickname and group tags to this router. If the AC Address field is empty, please enter your AC domain name or IP address, for example, test.cassia.pro.

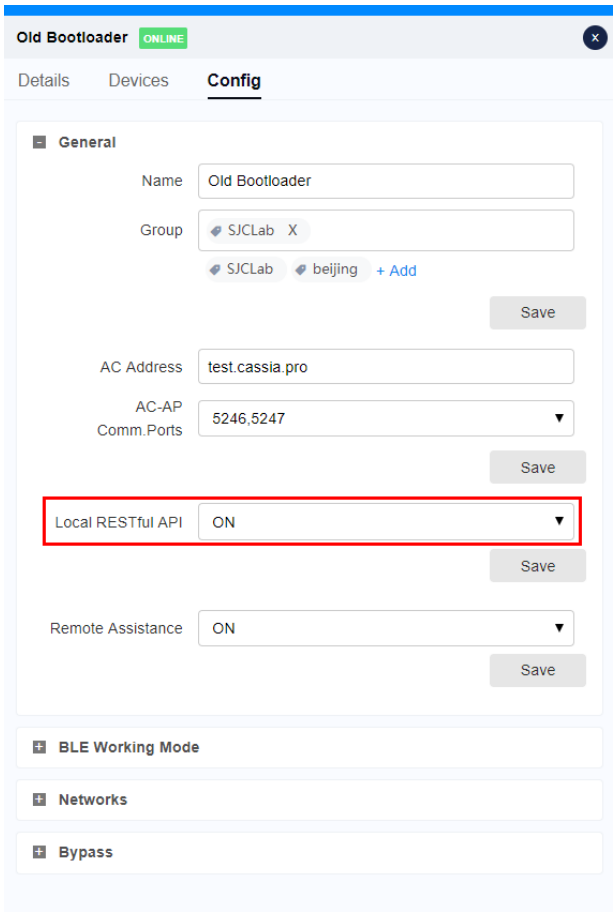


Figure 7: Router Config Edit page on the AC

Note: if you want to invoke Restful APIs directly from the router (see 4.4 a), you need turn on Local RESTful API. See above screenshot.

4.3.2 Configure Networks Settings

Next, click the Networks tab to setup your networking configurations. The Cassia Bluetooth Router supports the following networking uplinks.

- Wired: you can set it to use static IP or DHCP
- Wireless: enter your WiFi SSID and password, click Save. The router will go offline briefly and back online again.
- 3G/4G dongle: currently we support two cell dongles in US (Huawei MS2131, AT&T AirCard 313U)
- Networks Priority: When two or more network connections are activated, you can set priority levels for the networks. By default, the priority is: wired > wireless > 3G/4G.

4.4 Step Four: Remote Control Your Router

Now you can operate your Router to do certain tasks with your end devices through RESTful APIs. There are two ways you can access your router:

a) Access Cassia router from local network (Figure 8)

http://{your router's IP address}/gap/nodes/?event=1

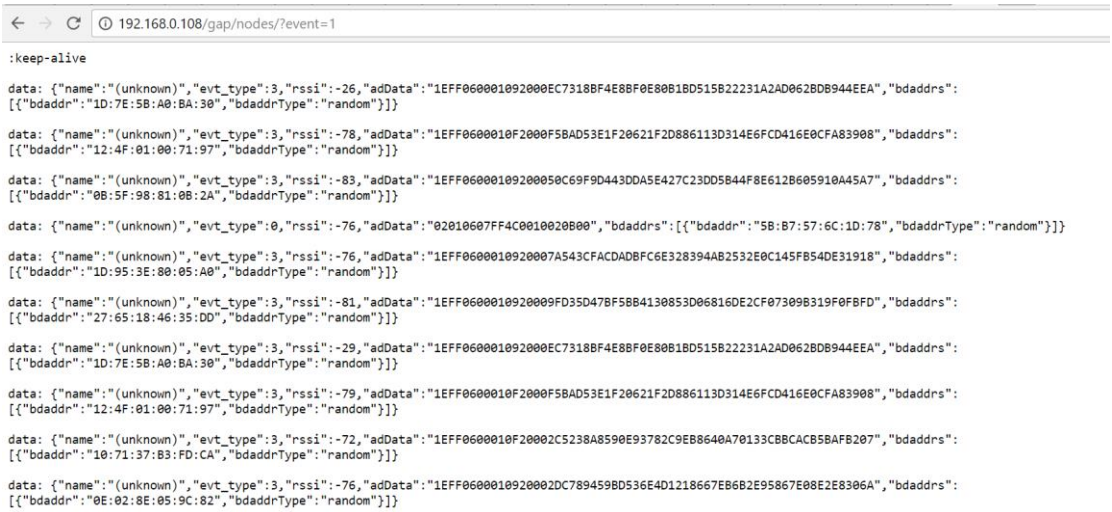


Figure-8 Access Cassia router from local network

b) Access Cassia router from the Cloud

If you would like to access your Router from the AC, you need to follow the steps below.

i. Do an OAuth2.0 authentication with the AC using client credentials granted. For example: you have a developer ID: tester, secret: 10b83f9a2e823c47, use base64 to encode string "tester:10b83f9a2e823c47" and get "dGVzdGVyOjEwYjgzZjlhMmU4MjNjNDc="

ii. Authenticate the user identity using the following http request, taking demo.cassia.pro as your AC server as an example.

```
POST api/oauth2/token HTTP/1.1
    Host: demo.cassia.pro
    Headers: {Authorization: Basic dGVzdGVyOjEwYjgzZjlhMmU4MjNjNDc=
    Content-Type: application/x-www-form-urlencoded}

    Body:
    {grant_type=client_credentials}
```

If everything is correct, you will get a response like this:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{ token_type: 'bearer',
   access_token:
'2b6ced831413685ec33204abc2a9a476310a852f53a763b72c854fd7708499f1bc0b362
6bfcfef2a2cfe0519356c9d7cb1b514243cb29f60e76b92d4a64ea8bd',
   expires_in: 3600 }
```

a) Now you can use access_token to access the other RESTful APIs by appending an access_token parameter. Note, make sure to append /api after the AC domain name or IP. For example, http://demo.cassia.pro/api/gap/nodes?event=1&mac=[router-mac]&access_token=xxx. Or you can add {Authorization : 'Bearer ' + access_token } in http headers.

5 RESTful APIs

In the Cassia AC, most of the Bluetooth GAP/GATT operations are exposed in RESTful APIs. The signatures of those APIs are fully compliant with Bluetooth SIG's Internet Working Group RESTful API specification.

There are 2 sets of APIs/Endpoints that you can use to interact with the router: the APIs on the local router and the APIs on our Access Controller. Except for Bluetooth positioning and AC status check, the two endpoints have the same set of APIs and give you the same result for any specific operations.

5.1 Interface Parameters

RESTful Interface parameters are defined below:

- mac: the mac address of a Cassia hub (eg：CC:1B:E0:E0:24:B4)
- node: the mac address of a BLE device (eg：EF:F3:CF:F0:8B:81)
- handle: after you find the device services, based on the device's Bluetooth profile, you can identify its corresponding handle index in the UUID (eg: 37)
- value: the value written into the handle

- chip: 0 or 1, indicates which chip of the Cassia hub is being used to operate this command. This parameter is optional. (Note: S Series only support chip 0, X1000/E1000/C1000 supports 0 and 1)
- interval: advertise interval, data type is number
- ad_data: advertise package, data type is string
- rsp_data: advertise response package, data type is string

5.2 Commonly Used APIs

5.2.1 Management APIs

a. How to obtain Cassia Router's IP address?

You can use the below API to obtain the configuration of a router, including its IP address.

GET http://{your AC domain}/api/cassia/info?mac=<hubmac>

{"mac":"CC:1B:E0:E0:95:84","ac":{"address":"demo.cassia.pro","force_network":"1","user":"tester"},"chip-params":"1","chips":["CC:1B:E0:E0:95:86"],"wireless":{"password":"","proto":"dhcp","ssid":"","dns2":"","dns1":""},"ip":{"eth0":"192.168.0.108"},"wired":{"dns1":"","proto":"dhcp","dns2":""},"timezone":"America\/Vancouver","version":"1.0.3.201708231615","local-api":"1","start":{"scanning":{"filter_name":"","filter_uuid":"","filter_mac":"CC:1B:E0:E0:58:A7","use":""},"bypass":{"keyId":"AKIAJB3LERONSMBI6AVA","queueUrl":"http:\/\/sqs.us-west-1.amazonaws.com\/155703096309\/test1","key":"sxOj1tMV1pYI2q1\/NnVzlhQCVWlmjLwpXn2BVq8Q","use":"sqs"}},"antenna":0,"model":"X1000"}

b. How to obtain Cassia Router's status?

You might use below API to obtain the status of a router, either online or offline. The return result is a JSON object.

GET http://{your AC domain}/api/cassia/hubs/<hubmac>

{"_id":"5a9497eeadc22500524e27e5","id":"CC:1B:E0:E0:DF:80","mac":"CC:1B:E0:E0:DF:80","name":"New Bootloader","group":"SJCLab","status":"online","model":"E1000","version":"1.2.1.1803121427","position":"","time":1519687662258,"ip":"96.64.240.30","localip":"192.168.0.106","uptime":807183,"offline_time":1523468797,"online_time":1523468804,"update_status":"update_ok","update_reason":"","update_version":"1.2.1.1803121427","update_progress":100,"groupcolor":"undefined"}

Offline example.

{"_id":"5a9f5bb26d48ab005290b45f","id":"CC:1B:E0:E0:61:9C","mac":"CC:1B:E0:E0:61:9C","name":"Cassia Router","group":"","status":"offline","model":"C1000","version":"1.2.2.1801101456","position":"","time":1520393138130,"ip":"73.202.248.99","localip":"192.168.1.106","uptime":1708,"offline_time":1520893570,"online_time":1520891832,"update_status":"update_ok","update_reason":"","update_version":"","update_progress":0}

c. How to obtain all online Routers' status?

GET http://{your AC domain}/api/cassia/hubs

The return result is an array of JSON objects.

[{"_id":"5a9497eeadc22500524e27e5","id":"CC:1B:E0:E0:DF:80","mac":"CC:1B:E0:E0:DF:80","name":"New
Bootloader","group":"SJCLab","status":"online","model":"E1000","version":"1.2.1.1803121427","position":"","time":1519687662258,"ip":"96.64.240.30","localip":"192
.168.0.106","uptime":806403,"offline_time":1523468797,"online_time":1523468804,"update_status":"update_ok","update_reason":"","update_version":"1.2.1.1803121427"
,"update_progress":100,"groupcolor":"undefined"},{"_id":"5aa852cb8373af00511af379","id":"CC:1B:E0:E0:95:84","mac":"CC:1B:E0:E0:95:84","name":"LXC
Router","group":"","status":"online","model":"X1000","version":"1.2.1.1803121426","position":"","time":1520980683723,"ip":"96.64.240.30","localip":"192.168.0.109
","uptime":806402,"offline_time":1523468798,"online_time":1523468805,"update_status":"update_ok","update_reason":"","update_version":"1.2.1.1803121426","update_p
rogress":100},{"_id":"5aa857e18373af00511af3c8","id":"CC:1B:E0:E0:56:2C","mac":"CC:1B:E0:E0:56:2C","name":"Azure LXC
Router","group":"","status":"online","model":"C1000","version":"1.2.7.1804041630","position":"","time":1520981985617,"ip":"96.64.240.30","localip":"10.1.10.142",
"uptime":351539,"offline_time":1523923662,"online_time":1523923668,"update_status":"update_ok","update_reason":"","update_version":"1.2.7.1804041630","update_pro
gress":100},{"_id":"5aa990c78373af00511b0609","id":"CC:1B:E0:E0:96:34","mac":"CC:1B:E0:E0:96:34","name":"ABB Cassia
Router","group":"","status":"online","model":"X1000","version":"1.2.0.1803131043","position":"","time":1521062087711,"ip":"96.64.240.30","localip":"192.168.0.110
","uptime":806401,"offline_time":1523468799,"online_time":1523468806,"update_status":"update_ok","update_reason":"","update_version":"","update_progress":0},
{"_id":"5ab207828373af00511b846d","id":"CC:1B:E0:E0:2D:F0","mac":"CC:1B:E0:E0:2D:F0","name":"Cassia
Router","group":"","status":"online","model":"X1000","version":"1.2.0.1802130330","position":"","time":1521616770703,"ip":"98.234.115.2","localip":"192.168.1.28"
,"uptime":1730,"offline_time":1524273446,"online_time":1524273477,"update_status":"update_ok","update_reason":"","update_version":"","update_progress":0},
{"_id":"5abaf0768373af00511c097c","id":"CC:1B:E0:E0:DD:88","mac":"CC:1B:E0:E0:DD:88","name":"Old
Bootloader","group":"","status":"online","model":"E1000","version":"1.2.1.1803121427","position":"","time":1522200694908,"ip":"96.64.240.30","localip":"192.168.0
.111","uptime":806396,"offline_time":1523468804,"online_time":1523468811,"update_status":"update_ok","update_reason":"","update_version":"","update_progress":0},
{"_id":"5abd940a8373af00511c3122","id":"CC:1B:E0:E0:52:90","mac":"CC:1B:E0:E0:52:90","name":"Zan Compute
","group":"SJCLab","status":"online","model":"C1000","version":"1.2.7.1804041630","position":"","time":1522373642805,"ip":"96.64.240.30","localip":"10.1.10.174",
"uptime":184268,"offline_time":1524090843,"online_time":1524090939,"update_status":"update_ok","update_reason":"","update_version":"1.2.7.1804041630","update_pro
gress":100,"groupcolor":"undefined"},{"_id":"5ac564748373af00511ca5d1","id":"CC:1B:E0:E0:64:44","mac":"CC:1B:E0:E0:64:44","name":"Zan Cynthia
Test","group":"SJCLab","status":"online","model":"C1000","version":"1.2.7.1804041630","position":"","time":1522885748412,"ip":"96.64.240.30","localip":"10.1.10.2
6","uptime":174065,"offline_time":1524101134,"online_time":1524101142,"update_status":"update_fail","update_reason":"download
interrupted","update_version":"1.2.0.1803131043","update_progress":3.274724245071411,"groupcolor":"undefined"},
{"_id":"5ac564748373af00511ca5d2","id":"CC:1B:E0:E0:5A:C0","mac":"CC:1B:E0:E0:5A:C0","name":"Zan Cynthia
Test","group":"SJCLab","status":"online","model":"C1000","version":"1.2.7.1804181819","position":"","time":1522885748415,"groupcolor":"undefined","ip":"216.3.13.
179","localip":"10.161.94.54","uptime":40880,"offline_time":1524234317,"online_time":1524234327,"update_status":"update_ok","update_reason":"","update_version":"
1.2.7.1804181819","update_progress":100}]

d. How to Reboot a Router remotely?

GET http://{your AC domain}/api/cassia/reboot?mac=<hubmac>&access_token=XXX

5.2.2 GATT APIs

e. How to use Cassia Router to scan for Bluetooth devices in its range?

To use the Router to scan for Bluetooth Low Energy (BLE) devices using your AC's endpoints:

GET http://{your AC domain}/api/gap/nodes?event=1&active=&mac=<hubmac>

This API is a SeverSide Event (SSE) which will be running continuously. Refer to section 6 for the details of SSE.

- active: 0 or 1, 0 indicates passive scanning and 1 active scanning. This parameter is optional. If you don't specify, by default Cassia routers will perform passive scanning.
- filter_duplicates: 0 or 1, turn on/off to filter duplicated records.

f. How to filter scanned data based on MAC address, device name or others?

This is a useful API which can significantly reduce the amount of traffic sent from the Router to the server.

GET http://{your AC domain}/api/gap/nodes?event=1&mac=<hubmac>&filter_mac=<mac>

http://{your AC domain}/api/gap/nodes?event=1&mac=<hubmac>&filter_mac=<mac1>,<mac2>,<mac3>

http://{your AC domain}/api/gap/nodes?event=1&mac=<hubmac>&filter_name=<name1>,<name2>

You can also filter out devices based on its RSSI level, e.g. Filter out devices RSSI value weaker than a certain value.

```
GET http://{your AC domain}/api
/gap/nodes?event=1&mac=<hubmac>&filter_rssi=<num>
```

In addition, you can filter out a device based on its service UUID inside its advertise packet.

```
GET http://{your AC domain}/api
/gap/nodes?event=1&mac=<hubmac>&filter_uuid=<uuid>
```

Note: above APIs related to filtering only apply for the traffic going through the Cassia AC. (APIs for bypass traffic filtering are coming. When added, we will update the document reflect that they are available.)

g. How to connect/disconnect to a target device?

To use the Router to connect to specific BLE devices using Cassia AC's endpoints:

```
POST http://{your AC domain}/api/gap/nodes/<node>/connection?mac=<hubmac>
```

We have added a few parameters in release 1.2 for this API:

- type: the BLE device's address type, either public or random
- timeout: in ms, the connection request will timeout if it can't be finished within this time. Default timeout is set to 20,000ms. For S Series, the timeout value can't be configured, while for X1000/E1000/C1000, this parameter is configurable, the minimum value is 200ms.
- auto: 0 or 1, indicates whether or not the BLE device will be automatically reconnected after it is disconnected unexpectedly. Return value: 200 for success, 500 for error.

For example,

```
curl -X POST -H "content-type: application/json" -d
'{"timeout":"1000","type":"public","auto":"1"}'
'http://172.16.10.6/gap/nodes/CC:1B:E0:E8:09:2B/connection'
```

To disconnect:

DELETE http://{your AC domain}/api/gap/nodes/<node>/connection?mac=<hubmac>

Get the device list connected to a router:

GET http://{your AC domain}/api/gap/nodes?connection_state=connected&mac=<hubmac>

h. How to discover GATT services and characteristics?

Discover all services:

GET http://{your AC domain}/api/gatt/nodes/<node>/services?mac=<hubmac>&all=1

Discover all characteristics

GET http://{your AC domain}/api/gatt/nodes/<node>/characteristics?mac=<hubmac>&all=1

Discover all characteristics in one service

GET http://{your AC domain}/api/gatt/nodes/<node>/services/<service_uuid>/characteristics

Discover all descriptors in one characteristic

GET http://{your AC domain}/api/gatt/nodes/<node>/characteristics/<characteristic_uuid>/descriptors

Discover a specific service by service UUID:

GET http://{your AC domain}/api/gatt/nodes/<node>/services?mac=<hubmac>&all=1&uuid=<uuid>

Discover a characteristics by characteristics UUID:

GET http://{your AC domain}/api/gatt/nodes/<node>/characteristics?mac=<hubmac>&uuid=<uuid>

Discover all services, characteristics, and descriptors all at once:

GET http://{your AC domain}/api/gatt/nodes/<node>/services/characteristics/descriptors?all=1

i. How to read/write the value of a specific characteristic?

The read/write operations are based on the handle of a specific characteristic.
The handle of a specific characteristic can be found in the discovery result.
To read by handle:

GET http://{your AC domain}/api/gatt/nodes/<node>/handle/<handle>/value?mac=<hubmac>

To write by handle:

GET http://{your AC domain}/api/gatt/nodes/<node>/handle/<handle>/value/<value>?mac=<hubmac>

j. How to subscribe to a notification/indication to a specific characteristic?

If you need to open a specific characteristic's notification or indication, you need to call the "discover service" interface first. To do so, find the corresponding descriptors of the specified characteristic, open the descriptors, find the UUID that contains "00002902" and its corresponding handle, say "37". Use this handle for the API call. To open the notification, set the value to "0100"; to open the indication, set the value to "0200"; to close the notification / indication, set the value to "0000".

GET http://{your AC domain}/api/gatt/nodes/<node>/handle/37/value/0100?mac=xxxx

k. How to get advertise data?

- interval: advertise interval in ms
- ad_data: advertise package, data type is string
- rsp_data: advertise response package, data type is string
- chip: you can specify which chip to use, 0 or 1. This parameter is optional. If you don't specify it, the router will pick up the chip automatically based on an internal algorithm.

Begin advertise data:

GET http://{your AC domain}/api/advertise/start?mac=&chip=&interval=&ad_data=&resp_data=

Stop advertise data:

GET http://{your AC_
domain}/api/advertise/stop?mac=&chip=&interval=&ad_data=&resp_data=

5.2.3 Positioning APIs

I. How to obtain the location info of your device?

Cassia supports room-based Bluetooth location tracking. Below are the related APIs.

- To identify the closest router a BLE device is located:

GET http://{your AC domain}/api/middleware/position/by-device/<device_mac>

It will return {“hubMac”:”hubMac1”}, e.g. {“hubMac”:”CC:1B:E0:E0:01:47”}.

- To obtain the closest router list for all the BLE devices that the AC can detect:

GET http://{your AC domain}/api/middleware/position/by-device/*

It will return a list:

```
{
    “device1”:{“hubMac”:”hubMac1”},
    “device2”:{“hubMac”:”hubMac2”},
    …
 }
e.g. {
        “11:22:33:44:55:66”:{“hubMac”:”CC:1B:E0:E0:01:47”},
        “11:22:33:44:55:77”:{“hubMac”:”CC:1B:E0:E0:01:48”},
}
```

- To get the list of BLE devices around a Cassia router:

GET http://{your AC domain}/api/middleware/position/by-ap/<hub_mac>

It will return [“device1”,” device2”,”device3”…]. For example, [“11:22:33:44:55:66”,”11:22:33:44:55:AA”…].

- To get the list of BLE devices for all the routers within the AC:

GET http://{your AC domain}/api/middleware/position/by-ap/*

It will return：

```
{
        “hubMac1”:[ “device1”,”device2”,”device3”…],
```

"hubMac2":["device1","device2","device3"…],
….
}
e.g.: {
"CC:1B:E0:E0:11:22":["11:22:33:44:55:66","11:22:33:44:55:AA"…],
…
}

5.2.4 Secure Pairing

Starting from 1.2 release, we have supported Bluetooth 4.1 Secure Simple Pairing on our routers, namely Just Works, Passkey Entry and Legacy OOB. Below are the Restful APIs.

- Invoke from the router directly:

Post http://<cassia router ip>/management/nodes/<node>/pair
Post http://<cassia router ip>/management/nodes/<node>/pair-input
DELETE http://<cassia router ip>/management/nodes/<node>/pair

- Invoke from the AC:

Post http://<your AC domain>/api/management/nodes/<node>/pair?mac=<hub-mac>
Post http://<your AC domain>/api/management/nodes/<node>/pair-input?mac=<hub-mac>
DELETE http://<your AC domain>/api/management/nodes/<node>/pair?mac=<hub-mac>

Please see below table for the return response.

Pair Mode	Post http://<gateway>/management/nodes/<node>/pair	Post http://<gateway>/management/nodes/<node>/pair-input	Delete http://<gateway>/management/nodes/<node>/pair
Just Works	Return 0 for pairing failed or 1 for sucessful	None	Return 0 for pairing failed or 1 for sucessful
Passkey Entry	Return to enter the number for passkey	Return 0 for pairing failed or 1 for sucessful	
Legacy OOB	Return 3 for using OOB	Return 0 for pairing failed or 1 for sucessful	

The API return values are indicated using status codes as the following table.

Status Code	Status Description
0	Pairing Failed
1	Pairing Successful
2	Pairing Aborted
3	LE Legacy Pairing OOB Expected
4	LE Secure Connections Pairing OOB Expected
5	Passkey Input Expected
6	Passkey Display Expected
7	Numeric Comparison Expected (LE Secure Connections Pairing only)

- OOB Legacy Pairing

For LE legacy OOB pairing, the intention to use LE Legacy OOB Pairing must be known before starting bonding process, as initiator and responder both must set OOB bit during Pairing Exchange phase.

Step #1

POST http://<gateway>/management/nodes/<node>/pair

Body (application/json):

```
{ "bond": 1, "legacy-oob": 1 }
200 – OK - application/json
{
"pairingStatusCode": 3,
"pairingStatus": "LE Legacy Pairing OOB Expected",
"pairingID": "0x0123456789ABCDEF0123456789ABCDEF"
}
```

Responses:

Step #2

POST http://<gateway>/management/nodes/<node>/pair-input

Body (application/json):

```
{ "pairingID": "0x0123456789ABCDEF0123456789ABCDEF",
"tk": "0x0123456789ABCDEF0123456789ABCDEF" }
```

Responses:

6 Server Sent Events (SSE)

```
200 – OK - application/json
{
  "pairingStatusCode": 1,
  "pairingStatus": "Pairing Successful"
}
```

6.1 What is SSE?

SSE is a technology where a browser receives automatic updates from a server via an HTTP connection. The Server-Sent Events EventSource API is standardized as a part of HTML5[1] by the W3C. SSE is used to send message updates or continuous data streams to a browser client. It needs to be manually terminated, otherwise it will keep on running until an error occurs.

Each SSE response starts with “data:”. When debugging, you can input the URL of a SSE into a web browser. In the program, an SSE request won’t return any data if you call the interface like a normal HTTP request, because a normal HTTP request only returns output when it finishes. In addition, when calling an SSE, you should monitor this thread, in case it is interrupted by an error or any unexpected incident, then you can restart it.

6.2 SSE in RESTful APIs

Three of the RESTful APIs are using SSE: scan, get device connection status, receive indication and notification.

a. Scan: See the following scan call as an example:

http://{your AC domain}/api/gap/nodes?event=1&mac=<hubMac>&access_token=<value>

This API call will constantly return the scan data from a Router with MAC address CC:1B:E0:E0:24:B4, until you stop the thread.

Note: If you use tools like CURL for HTTP request, the tool will return data when the HTTP request ends. However, our scan API is an SSE which NEVER ends and sends data in a stream, so it will hang the page. You should add the following snippet:

```
if ($stream = fopen($url_for_scan, 'r')) {
    while(($line = fgets($stream)) !== false) {
        echo $line;
    }
}
```

b. Get device connection status

SSE to get the connection status of all the devices that have connected to a router:

```
http://{your AC domain}/api/management/nodes/connection-state?mac=<hubmac>
```

When a device status is changed from disconnected to connected, or from connected to disconnected, you will get a response. For example,

```
data: {"handle":"CC:1B:E0:E8:0D:F2","connectionState":"connected"}
data: {"handle":"88:C6:26:92:58:77","connectionState":"disconnected"}
```

c. SSE to receive notification and indication

```
http://{your AC domain}/api/gatt/nodes?mac=<hubmac>
```

6.3 Best Practice

For a best practice in implementing your application, use the following guidelines:

a. Setup an SSE http://{your AC domain}/api/cassia/hubStatus

(without any parameters, need to OAuth before being used)

- When a Router is online, you will get a response like this:

```
data:
{"model":"X1000","ip":"96.64.240.30","mac":"CC:1B:E0:E0:98:50",
"version":"1.1.1.1710261111","uptime":0,"user":"tester","locali
p":"192.168.0.105","whitelist":true,"status":"online"}
```

- When a Router is offline, you will get a response like this.

```
data: {"mac":" CC:1B:E0:E0:98:50","status":"offline"}
```

b. When a Router's status is changed to “online”, check the SSEs that you are using. If the SSEs exist, destroy them first, then setup new ones. For example:

http://{your AC domain}/api/gatt/nodes?mac=<hubmac>

If an SSE is gone, setup a new one directly.

c. When a Router's status is changed to "offline", you can destroy the previous SSEs.

7 Bluetooth Debug tool

This is a development tool for developers to integrate Bluetooth devices with the Cassia Bluetooth router. With this tool, we put the RESTful APIs into a visual interface. After an API call, the tool will show you the response messages, allowing developers to quickly implement the integration using a programming language they are familiar with. This tool is available at http://www.bluetooth.tech/nativeHubControl/index.html

See Figure 8 below.

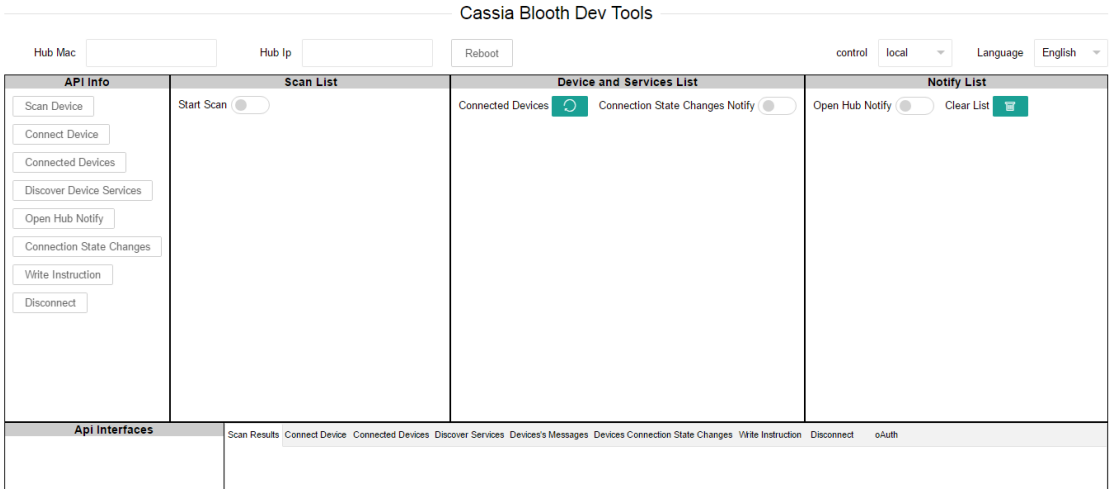


Figure 8: Cassia Bluetooth Debug Tool

API Info: This area contains the user's most commonly used commands. Clicking on each button which will pop up parameters and descriptions related to that command.

Scan List: Turn on this option and the Cassia Router will start scanning for all BLE devices within its range. The BLE devices need to be in broadcast mode. You can connect to one or multiple devices from there. See below Figure 9 for an example.

Device and Services List: Turn on this option to see the connected devices. Based on the device's Bluetooth profile, you can write value or turn on the notifications. See below Figure 10 for an example.

Notify List: If you have turned on the notification of the correct handle, you will see a stream of raw data flowing in the Notify List window. See below Figure 9 for an example.

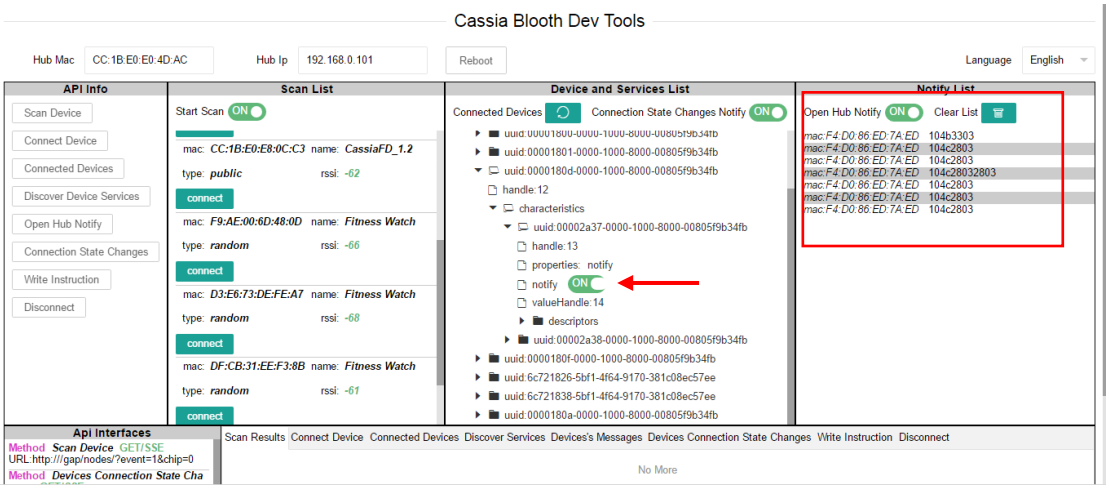


Figure 9 Cassia Bluetooth Debug Tool example

8 Troubleshooting Error Messages

For HTTP 500 error, the following are the common errors.

1) "parameter invalid": wrong parameter value, such as, chip ID, MAC address or advertise type is wrong.
2) "device not found": it is possible that this device is disconnected. A GATT call to query the attribute of a disconnected device will return this error.
3) "memory alloc error": when the Bluetooth chip does not have enough memory to complete the operation it will return this error.
4) "operation timeout": each operation has a time-out value, especially those time-consuming operations, such as connection. When connecting to a device that does not exist, the operation will timeout after 20s.
5) "chip is not ready": this error will be reported when sending commands to the chip fails.
6) "chip is busy": when sending the same command to the chip, the system will return "chip is busy", such as at the beginning of a scan. All the GATT commands are mutually exclusive, continuous call will return this busy error.
7) "incorrect mode": Our S Series only supports one role, either master or slave (due to the memory limit). These two roles are different, mainly reflected in the broadcast and scanning. When the router is a slave, it cannot conduct scanning; when the router is the master, it cannot send advertise which is

connectable. If you set the unsupported parameters to the chip, the system will return this error.

8) "device not connect": same as "device not found" error.
9) "operation not supported": reserved for future use.
10) "failure": an error for all other failures not specified yet.
11) "need pair operation": some devices require an operation for pairing after a successful connection. If a GATT function call happens prior to the pairing, the system will Return this error.
12) "no resources": our Bluetooth chip can store the pair information up to 10 seconds. If you pair too many devices, the system will report this error.
13) "Service Not Found": couldn't find a Characteristic inside a Service.
14) "type not supported": When set up Bypass scan, the protocol type you specified is not supported by this firmware.
15) "please set bypass params first": You have enabled to use Bypass mode, but haven't set the bypass parameters.

9 Migrate from C1000-2B Firmware to X1000

For users who have migrated from C1000-2B version to X1000, you need to make two changes: change host and add "/api" to the beginning of the URL. See below for an example.

Example code on C1000-2B

```
var host = "http://api.cassianetworks.com";
// get token
$.ajax({url: host+"/oauth2/token", headers: headers, type:"post", success: function(data){
  // ...
}});
```

In X1000/E1000/C1000/S Series, you used it this way：

```
var host = "http://demo.cassia.pro/api";
// get token
$.ajax({url: host+"/oauth2/token", headers: headers, type:"post", success: function(data){
  // ...
}});
```

Or

```
POST api/oauth2/token HTTP/1.1
  Host: demo.cassia.pro
  Headers: {Authorization: Basic dGVzdGVyOjEwYjgzZjlhMmU4MjNjNDc=
```

Content-Type: application/x-www-form-urlencoded}

Body:

{grant_type=client_credentials}

Appendix -- Sample Code in Node.js

```
var credentials = {
   id: 'tester',
   secret: '816213f8b5c2877d'
};

var access_token = '';

var request = require('request');
var options = {
       url : 'http://demo.cassia.pro/api/oauth2/token',
       method : 'POST',
       form : {'grant_type' : 'client_credentials'},
       headers : {
              Authorization : 'Basic ' + new Buffer(credentials.id + ':' + credentials.secret, 'ascii').toString('base64'),
       }
};

request(options, function(error, req, body) {
       if (error) {
              console.log(error);
              return;
       }

       var data = JSON.parse(body);
       access_token = data.access_token;
       console.log(data);

       var options = {
              url : 'http://demo.cassia.pro/api/client',    //you can change this to the IP address and port your Router
is using.
              method : 'GET',
              // form : {'grant_type' : 'client_credentials'},
              headers : {
                     Authorization : 'Bearer ' + access_token,
              }
       };
       });
});
     request(options, function(error, request, body) {
              console.log(body);
       });
});
```