Cassia Networks, Inc.
97 East Brokaw Road, Suite 130
San Jose, CA 95112
support@cassianetworks.com

Cassia SDK Implementation Guide

Release date: April 20th, 2019

Contents

Version: EN-1.4.1-04202019-YJ

1 Overview

This document shows developers how to use the Cassia RESTful APIs to integrate their Bluetooth devices with the Cassia IoT Access Controller (AC) and the Cassia Bluetooth routers, without requiring any changes to the Bluetooth devices.

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 devices.

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 and server using an HTTP/HTTPS-based communication protocol, which provides programming language flexibility (C#, Node.js, Java, and any other languages you prefer). This document helps you to use the Cassia RESTful APIs and its associated services.

The Cassia RESTful APIs provide the following functions:

- Connect and control your BLE devices.
- Support three modes: Scanning, Connecting, Broadcasting.
- Write/read data to/from the BLE device.
- Read data as notification/indication events from the BLE device.

2 Two Set of RESTful APIs

Cassia provides two sets of RESTful APIs that enable BLE device interaction with Cassia routers:

- APIs on the local router (where the application is usually on the same network as the router)
- APIs through Cassia's IoT Access Controller (Cassia AC).

The below APIs are only available through the Cassia AC. Except for these APIs, the two set of RESTful APIs are the same and will give the same result. In this document, we use the API through the Cassia AC as an example.

- Positioning APIs (chapter 6.4)
- Obtain Cassia router's status (chapter 6.2.2)
- Monitor Cassia router's status APIs (chapter 6.2.3)
- Obtain all online routers' status (chapter 6.2.4)
- Router auto-selection (chapter 6.6, introduced in firmware 1.3)
- SSE Combination (chapter 6.7, introduced in firmware 1.3)

NOTE

- The RESTful APIs through Cassia AC includes “/api” after {your AC domain}. It is not needed for the RESTful APIs on the local router.
- The RESTful APIs through Cassia AC includes “mac=<mac>” to identify which router is used. It is not needed for the RESTful APIs on the local router.
- For firmware 1.2 or below, if you want to use RESTful APIs on the local routers, you need to turn on Local RESTful API in AC console or router console. Please see below screenshots.

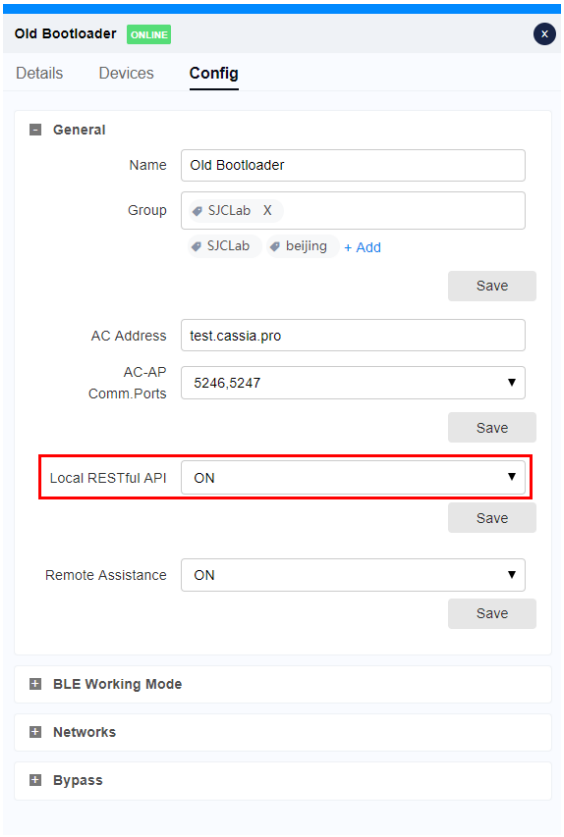


Figure 1: (v1.2) Turn on Local RESTful API in AC Console

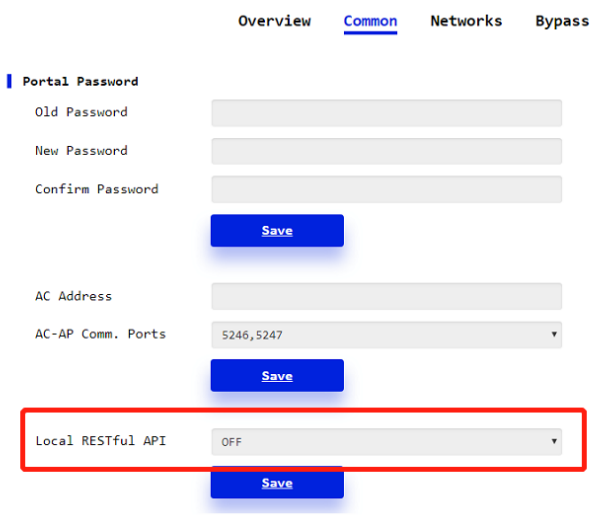


Figure 2: (v1.2) Turn on Local RESTful API in Router Console

- In firmware 1.3, if the router is configured as Standalone Mode, the local RESTful API will be automatically turned on. If the router is configured as AC Managed Mode, the local RESTful API will be turned off by default (customer can enable the local RESTful API from AC console). Please see below screenshots.

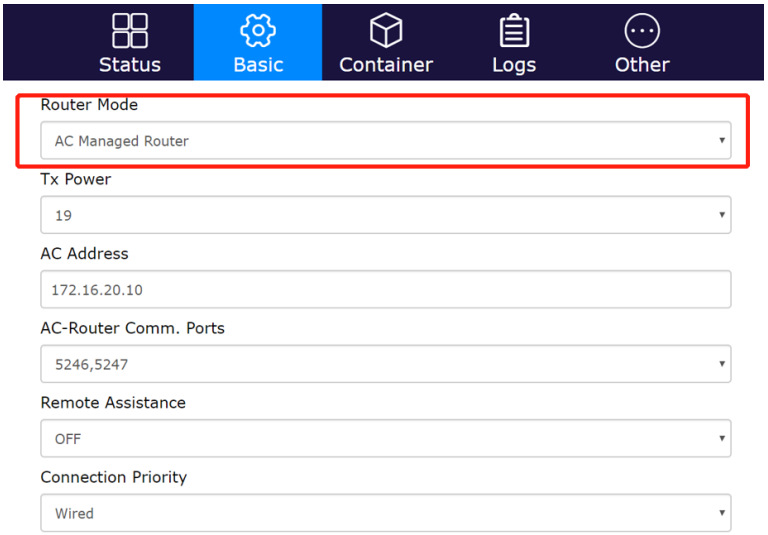


Figure 3: (v1.3) Configuration of Router Mode on Router Console

3 Architecture Diagram

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, and security policy management, enabling the remote control of Cassia Bluetooth routers across the Internet.

You can operate your BLE devices using a set of RESTful APIs, via the Cassia AC and the Cassia routers. Please see below figure for the Cassia RESTful APIs Working Diagram, using X1000 as an example.

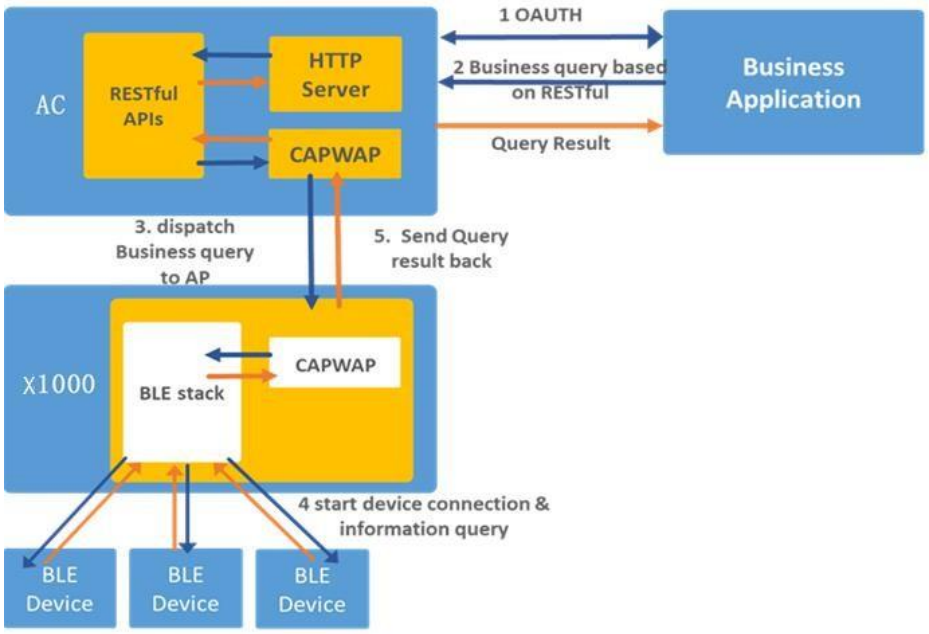


Figure 4: Cassia RESTful APIs Working Diagram

First, the business application initiates an OAuth authentication request (generated using developer credentials) to the Cassia AC. Once the authentication succeeds, it will send an HTTP query to the Cassia AC based on RESTful. Next, the Cassia 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 Cassia AC, and then to the business application.

4 Server Sent Events

SSE is a technology where a browser receives automatic updates from a server via an HTTP connection. The SSE API is standardized as a part of HTML5 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.

Five of the RESTful APIs are using SSE: they are scan (chapter 5.3.1), get device connection status (chapter 5.3.7), receive indication and notification (chapter 5.3.8), monitor Cassia router's status (chapter 5.2.3) and create combined SSE (chapter 5.6.1, firmware 1.3).

Each SSE response starts with "data:". When debugging, you can input the URL of an SSE into a web browser, then you will see the SSE output from the 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. If it is interrupted by an error or any unexpected incident, you can restart it.

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

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

5 Getting Started

Please setup the Cassia router, connect the router to Cassia AC or run the router in standalone mode. For more information, please check Router Quick Start Guide, Private AC Server Installation Guide, and Cassia User Manual at https://www.cassianetworks.com/knowledge-base/general-documents/

Now you can operate your router to do certain tasks with your Bluetooth devices through RESTful APIs:

5.1. Access local router

Your application can access local Cassia routers directly (usually in the same network), instead of through Cassia AC. Below is an example of running RESTful API in a web browser to access Cassia router in local network (for debug purpose).

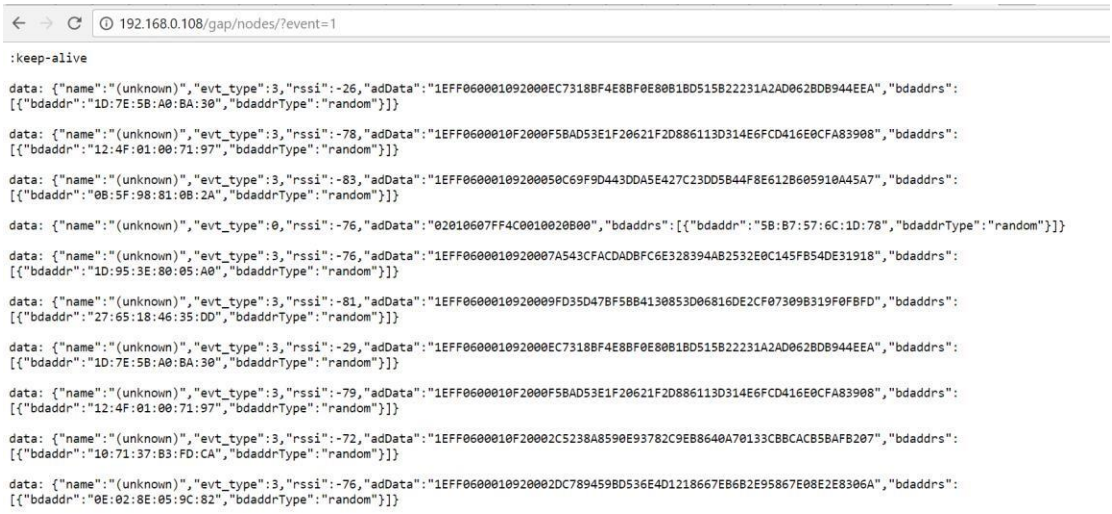


Figure 5: Access local Cassia router

5.2. Access Cassia router through the Cassia AC

Before starting to use RESTful API's through the Cassia AC, you will need developer credentials (a Developer Key and a Developer Secret). It is not needed for a RESTful API on a local router. These developer credentials authorize the remote control of the Cassia Bluetooth router. To request your developer credentials, please contact the Cassia support team at support@cassianetworks.com or contact your sales representative.

Here is a sample:

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

Then, you need to follow below steps.

- Do an OAuth2.0 authentication with the AC using developer credentials granted. For example: you have a developer ID: tester, secret: 10b83f9a2e823c47, use base64 to encode string "tester:10b83f9a2e823c47" and get "dGVzdGVyOjEwYjgzZjlhMmU4MjNjNDc="
- 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 goes well, you will get a response like this, which includes access_token:

```
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 }
```

- Now you can use access_token to access the other RESTful APIs by appending an access_token parameter. 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.

  NOTE: Make sure to append "/api" after {your AC domain} and add "mac=<mac>" to identify which router is used.

6 RESTful APIs

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.

6.1. Common Parameters

Here are common parameters for RESTful API:

- mac: the mac address of a Cassia router (e.g. CC:1B:E0:E0:24:B4)

- node: the mac address of a BLE device (e.g. 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 (e.g. 37)
- value: the hex value written into the handle (e.g. FF000C00)
- chip (optional): 0 or 1, indicates which chip of the Cassia router is used for scan and connect. By default, the router will pick up the chip automatically based on an internal algorithm. S Series routers only support chip 0, X1000/E1000/C1000 supports 0 and 1.

6.2. Management APIs

6.2.1. Obtain Cassia router's configuration

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

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

{"capwap-uplinkmac":"CC:1B:E0:E7:FD:84","cpu":{"used":158,"total":2184},"mem":{"used":103508,
"total":225716},"capwap-state":"7\n","timeconf":{"now":"2018-09-07 09:56:42","ntp2":"stdtime.gov.hk",
"auto":true,"ntp1":"time.nist.gov"},"timezone":"Asia\/Shanghai","chipinfo":{"1":{"adv_en":"0","ant":"0","max":11,"s
can_type":"0","ver":"7","addr":"CC:EB:E0:19:88:1F","scan_en":"0","status":"Idle","speed":{"rx":0,"tx":0},"id":"1"},"
0":{"adv_en":"0","ant":"0","max":11,"scan_type":"0","ver":"7","addr":"CC:EB:E0:19:88:1E","scan_en":"0","status":
"Idle","speed":{"rx":0,"tx":0},"id":"0"}},"conn_params":{"type":0,"latency":0,"conn_max_intval":30,"supvtimeout":1
000,"conn_min_intval":7.5,"scan_window":30,"scan_intval":60},"mqtt_stat":{"e":[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],"
d":0,"s":0,"f":0},"wireless":{"proto":"static","dns":"","dns2":"","netmask":"255.255.255.0","ip":"192.168.40.1","pass
word":"cassia-E7FD84","speed":{"rx":0,"tx":0},"iface":{"mac":"CC:1B:E0:E7:FD:85","mtu":"1500",
"netmask":"255.255.255.0","ip":"192.168.40.1","tx":"428","name":"wlan0","metric":"0","rx":"0","bcast":"192.168.4
0.255"},"country":"US","gateway":"","mode":"ap","ssid":"cassia-E7FD84","dns1":""},"ssh-
login":"1","ble_power":20,"sqs_stat":{"e":[0,0,0,0,0,0],"d":0,"s":0,"f":0},"fat":"0","mac":"CC:1B:E0:E7:FD:84","cont
ainer":{"disk":{"used":"1.0G","total":"2.3G"},"kernel":"Ubuntu 16.04.3 LTS\n",
"iface":{"mac":"FE:EB:E0:BE:E0:62","mtu":"1500","tx":"636","name":"vethBU791K","metric":"0","rx":"568"},"cpu":
{"used":205,"total":2204},"status_code":3,"process":"USER PID %CPU %MEM VSZ RSS TTY STAT
START TIME COMMAND\nroot 1 0.0 0.4 2612 1056 pts\/0 Ss+ 01:55 0:00 \/bin\/bash
\/root\/start.sh\nroot 71 0.0 0.3 7980 848 ? Ss 01:55 0:00 \/usr\/sbin\/sshd\nroot 86 7.1 10.5
118496 23852 ? Ssl 01:55 0:04 PM2 v2.10.1: God Daemon (\/root\/.pm2)\nroot 99 0.0 0.6 2708
1360 pts\/0 S+ 01:55 0:00 \/bin\/bash\nroot 133 0.0 0.4 4740 1124 ? R 01:56 0:00 ps
aux\n","apps":{},"speed":{"rx":0,"tx":0},"mem":{"used":52350976,"total":134217728},"status":"running"},"capwap-
uplink":"wired","ac":{"port":"5246,5247","user":"","control_port":"5246","data_port":"5247","force_network":"1","a
ddress":""},"capwap-ip":"168.168.20.154\n","https":"0","dongle":{"keepalive":"","ifname":"ppp0",
"dialnumber":"*99#","service":"umts","defaultroute":"","username":"","pincode":"","apn":"3gnet","metric":"5","prot
o":"3g","dns":"","device":"\/dev\/ttyUSB0","maxwait":"","password":"","ipv6":"","type":"none","demand":"","peerdn
s":""},"scan":{"one_scan_time":"300","scan_interval":"15","scan_window":"10"},"wired":{"duplex":"full","proto":"dh
cp","speed":{"rx":0.381,"tx":0.2376},"iface":{"mac":"CC:1B:E0:E7:FD:84","mtu":"1500","netmask":"255.255.255.0
","ip":"168.168.20.30","tx":"27746","name":"eth0","metric":"1","rx":"31793","bcast":"168.168.20.255"},"trans_spe
ed":"100"},"chip-params":"1","version":"1.3.0.1807030130","local-api":"0","start":{"bypass":{"use":"mqtt"}},
"capwap-runtime":54,"uptime":"94","model":"X1000"}

6.2.2. 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.

NOTE: this API is only available through Cassia AC.

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

Response example for online:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

{"_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"}

Response example for offline:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

{"_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}

6.2.3. Monitor Cassia router's status

You can use this API to monitor the status of a router continuously.

NOTE: This API is a Server-Sent Events (SSE) API and is only available through Cassia AC.

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

Response example for online:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

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"}

Response example for offline:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

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

6.2.4. Obtain all online routers' status

NOTE: This API is only available through Cassia AC.

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

The return result is an array of JSON objects.

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

[{"_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_progress":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_progress":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_progress":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":"10.1.10.26","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}]

6.2.5. Reboot a router remotely

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

6.3. Traffic Related APIs

6.3.1. Scan Bluetooth devices

To use the router to scan Bluetooth Low Energy (BLE) devices through your AC:

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

This API is a Server-Sent Events (SSE) which will be running continuously. Please check figure 5 for response example.

Here are more optional parameters.

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

6.3.2. Filter scanned data based on device MAC, RSSI, name, and UUID

This API can significantly reduce the amount of packets sent from the router to the server.

NOTE: Multiple filters can be used at the same time. Scanned data is returned if all conditions are met. The wildcard is not supported.

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

Customers can filter out devices based on its RSSI level, e.g. filter out devices who's RSSI value is weaker than a certain value.

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

In addition, customers can filter out devices based on service UUID and name inside the scanned packets. The service UUID may be only part of the UUID in BLE profile. What is more, filter_uuid should not include "-".

GET http://{your AC domain}/api/gap/
nodes?event=1&mac=<hubmac>&filter_uuid=<uuid1>,<uuid2>, … , <uuidX>

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

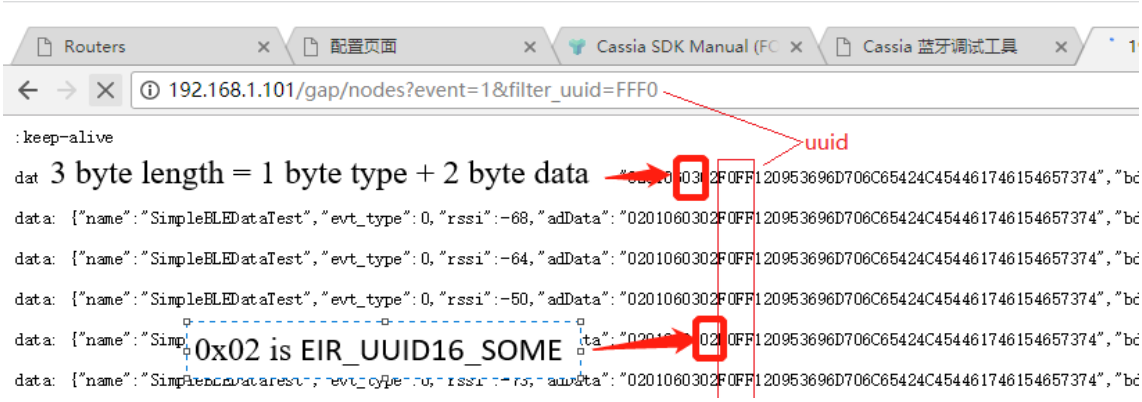
The structure of BLE advertise packets and scan response packets is [1 Byte Length (type + data) + 1 Byte Type + Data] x n. In order to filter by UUID or name, the corresponding type should be included in advertise packets (adData) or scan response packets (scanData). Below are the types.

```
#define EIR_UUID16_SOME 0x02  /* 16-bit UUID, more available */
#define EIR_UUID16_ALL  0x03  /* 16-bit UUID, all listed */
#define EIR_UUID32_SOME 0x04  /* 32-bit UUID, more available */
```

#define EIR_UUID32_ALL 0x05 /* 32-bit UUID, all listed */
#define EIR_UUID128_SOME 0x06 /* 128-bit UUID, more available */
#define EIR_UUID128_ALL 0x07 /* 128-bit UUID, all listed */
#define EIR_NAME_SHORT 0x08 /* shortened local name */
#define EIR_NAME_COMPLETE 0x09 /* complete local name */

Below is an example which includes name in scan response.

Below is an example which includes UUID in advertise packet. The uuid in this advertise packets is FOFF. Please move the last byte (FF) forward and add the rest of the bytes(F0), here comes the filter_uuid= FFF0.



6.3.3. Connect/disconnect to a target device

To use the router to connect to specific BLE devices using Cassia AC:

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

NOTE: multiple connecting requests cannot be handled simultaneously by one router. User needs to handle requests in serial, which is to wait for the response and then invoke the next connecting request.

Parameters for this API.

- type (mandatory): the BLE device's address type, either public or random.
- timeout (optional): in ms, the connection request will timeout if it can't be finished within this time. The default timeout is 5,000ms. The range of value is 200ms – 20000ms.
- auto (optional): 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. The default value is 0.

- discovergatt (optional): 0 or 1 (default)
  - ❖ Value 1 indicates the router should use the cached GATT database which was discovered during previous connection. It will save time for service discover API, but maybe the information is not updated.
  - ❖ Value 0 indicates the router should not use the cached GATT database. When customer calls service discover API, the router should read the GATT services & characteristics from the BLE device.

Here is an example for access the router from the local network (no "/api" and "mac=<mac>").

```
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'
```

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

```
OK
```

To disconnect:

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

```
OK
```

Get the device list connected to a router:

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

```
{"nodes":[{"type":"random","bdaddrs":{"bdaddr":"EF:A3:E6:94:CD:2D","bdadd
Type":"random"},"chipId":0,"handle":"","name":"","connectionState":"cone
ted","id":"EF:A3:E6:94:CD:2D"}]}
```

6.3.4. Discover GATT services and characteristics

Discover all services:

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json


```
[{"uuid":"00001800-0000-1000-8000
00805f9b34fb","primary":true,"handle":1},{"uuid":"00001801-0000-1000-8000
00805f9b34fb","primary":true,"handle":8},{"uuid":"0000fd00-0000-1000-8000
00805f9b34fb","primary":true,"handle":9},{"uuid":"0000180d-0000-1000-8000
00805f9b34fb","primary":true,"handle":20},{"uuid":"0000180f-0000-1000
8000-00805f9b34fb","primary":true,"handle":26},{"uuid":"0000180a-0000
1000-8000-00805f9b34fb","primary":true,"handle":30},{"uuid":"00001802
0000-1000-8000
00805f9b34fb","primary":true,"handle":43},{"uuid":"00001803-0000-1000
8000-00805f9b34fb","primary":true,"handle":46}]
```


Discover all characteristics:

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json


```
[{"handle":3,"properties":10,"uuid":"00002a00-0000-1000-8000
00805f9b34fb"},{"handle":5,"properties":2,"uuid":"00002a01-0000-1000-8000
00805f9b34fb"},{"handle":7,"properties":2,"uuid":"00002a04-0000-1000-8000
00805f9b34fb"},{"handle":11,"properties":16,"uuid":"0000fd09-0000-1000
8000-00805f9b34fb"},{"handle":14,"properties":4,"uuid":"0000fd0a-0000
1000-8000-00805f9b34fb"}]
```


Discover all characteristics in one service:

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json


```
[{"handle":48,"properties":10,"uuid":"00002a06-0000-1000-8000
00805f9b34fb"}]
```

Discover all descriptors in one characteristic:

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json


```
[{"handle":48,"properties":10,"uuid":"00002a06-0000-1000-8000
00805f9b34fb"}]
```


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

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json


```
[{"uuid":"00001800-0000-1000-8000
00805f9b34fb","primary":true,"characteristics":[{"descriptors":[{"handle"
3,"uuid":"00002a00-0000-1000-8000
00805f9b34fb"}],"handle":3,"properties":10,"uuid":"00002a00-0000-1000
8000-00805f9b34fb"},{"descriptors":[{"handle":5,"uuid":"00002a01-0000
1000-8000-00805f9b34fb"}],"handle":5,"properties":2,"uuid":"00002a01-0000
1000-8000-00805f9b34fb"},{"descriptors":[{"handle":7,"uuid":"00002a04
0000-1000-8000-00805f9b34fb"}],"handle":7,"properties":2,"uuid":"00002a04
0000-1000-8000-00805f9b34fb"}],"handle":1}]
```


6.3.5. Read/write the value of a specific characteristic

The read/write operations are based on the handle (found in the discover result) of a specific characteristic.

To read by the handle:

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

```
{"handle":"36","value":"56312e362e31"}
```

To write by the handle:

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

Optional parameters.

- noresponse (optional): 0 or 1, turn on/off the response message. Default is 0.
  1 indicates chip will not send response after complete writing.

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

```
OK
```

Below is an example for opening and closing a specific characteristic's notification and indication by Write API.

First, you need to call the Discover API to find the corresponding descriptors of the specified characteristic. Then, open the descriptors, find the UUID and its corresponding handle, e.g. "37". Now, you can use this handle in the Write API. 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" (37, 0100, 0200 and 0000 are examples).

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

```
OK
```

6.3.6. Send advertise data

Start sending advertise data:

GET http://{your AC domain}/api/advertise/start?mac=<mac>

Here are the parameters

- Interval (optional): advertising interval in ms. Default value 500ms.
- ad_type (optional): advertising type (see below table). Default value 3.

- ad_data (mandatory): advertise package. The data type is string.
- resp_data (optional): scan response package. The data type is string. When you want to send resp_data, please set ad_type=0.

Value	ad_type	Comments
0	ADV_IND	Connectable undirected advertising
1	ADV_DIRECT_IND	Connectable directed advertising
2	ADV_SCAN_IND	Scannable undirected advertising
3	ADV_NONCONN_IND	Non connectable undirected advertising
4	SCAN_RSP	Scan Response

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

```
OK
```

Stop sending advertise data:

GET http://{your AC domain}/api/advertise/stop?mac=<mac>

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

```
OK
```

6.3.7. Get device connection status

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

GET 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"}
```

6.3.8. Receive notification and indication

SSE API to continues receive notification and indication.

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

6.4. Positioning APIs

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

NOTE: Before calling any positioning APIs, please call scan API for the related routers. Positioning APIs are only available through Cassia AC.

- 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”…],
…
}

6.5. Secure Pairing APIs

Starting from 1.2 release, Cassia supports Bluetooth 4.1 Secure Simple Pairing, namely Just Works, Passkey Entry and Legacy OOB.

Here are the mapping between pair modes, APIs and typical responses.

	Step 1: API Pair Request	Step 2: API Pair-input Request
Just Works	Return 0 for pairing failed or 1 for successful	N/A
Passkey Entry	Return 5 for using passkey entry (initiator inputs)	Return 0 for pairing failed or 1 for successful
Legacy OOB	Return 3 for using legacy OOB	Return 0 for pairing failed or 1 for successful

6.5.1. Pair Request

POST http://<your AC domain>/api/management/nodes/<node>/pair?mac=<hubmac>

Body parameters

- Bond (optional): Bond to the node. Default value is 1
- legacy-oob (optional): Default value is 0, which means not using Legacy OOB. If customer wants to use Legacy OOB, please set it to 1
- io-capability (optional): See below table. Default value is KeyboardDisplay

IO capability

Value	**Comments**
DisplayOnly	Check BLE specification version 4.2
DisplayYesNo	Check BLE specification version 4.2
KeyboardOnly	Check BLE specification version 4.2
NoInputNoOutput	Check BLE specification version 4.2
KeyboardDisplay	Default value

Response parameters

Name	**Optional/Mandatory**	**Description**
HTTP 500 error	Optional	Please check chapter 8
pairingStatusCode	Optional	See below table
pairingStatus	Optional	Description of pairing status code
display	Optional	Display for pairing status codes 6

Pairing status codes

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)

6.5.2. Pair-input Request

NOTE: This API is not needed for Just Works.

POST http://<your AC domain>/api/management/nodes/<node>/pair-input?mac=<hubmac>

Body example for Passkey Entry (application/json):
{ "passkey": "123456" }

Body example for Legacy OOB (application/json):
{ "tk": "0x0123456789ABCDEF0123456789ABCDEF" }

The response format is same as pair request API.

6.5.3. Un-pair request

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

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

```
OK
```

6.5.4. Just Works example

POST http://<your AC domain>/api/management/nodes/<node>/pair?mac=<hubmac>

Body example (application/json):
{ "bond": 1 [, "io-capability": "NoInputNoOutput"] }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

```
{ "pairingStatusCode": 1, "pairingStatus": "Pairing Successful" }
```

6.5.5. Passkey Entry example: initiator inputs

Step #1

POST http://<your AC domain>/api/management/nodes/<node>/pair?mac=<hubmac>

Body example (application/json):
{ "bond"=1 [, "io-capability": "KeyboardDisplay"] }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

```
{ "pairingStatusCode": 5, "pairingStatus": "Passkey Input Expected" }
```

Step #2

POST http://<your AC domain>/api/management/nodes/<node>/pair-input?mac=<hubmac>

Body example (application/json):
{ "passkey": "123456" }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

```
{ "pairingStatusCode": 1, "pairingStatus": "Pairing Successful" }
```

6.5.6. LE Legacy Pairing OOB example

Step #1

POST http://<your AC domain>/api/management/nodes/<node>/pair?mac=<hubmac>

Body example (application/json):
{ "bond": 1, "legacy-oob": 1 }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

```
{ "pairingStatusCode": 3, "pairingStatus": "LE Legacy Pairing OOB
```

```
Expected" }
```

Step #2

POST http://<your AC domain>/api/management/nodes/<node>/pair-input?mac=<hubmac>

Body example (application/json):
{ "tk": "0x0123456789ABCDEF0123456789ABCDEF" }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json

```
{ "pairingStatusCode": 1, "pairingStatus": "Pairing Successful" }
```

6.6. Router Auto-Selection APIs

From firmware 1.3, Cassia AC can select one router automatically from a list of candidates, and then connect the BLE device by using this router. The selection is based on RSSI, router load, and router capabilities.

If a customer wants to connect a BLE device with a specific router, or he wants to use a customized router selection algorithm, he should use the APIs in chapter 5.3.3.

NOTE: these APIs are only available through Cassia AC.

6.6.1. Router auto-selection

This API will enable/disable router auto-selection function. If the flag is 1, the router auto-selection function will be enabled. If the flag is 0, the router auto-selection function will be disabled.

NOTE: This API should be called before using any other router auto-selection APIs. The user can also switch on/off router auto-selection function in Cassia AC settings, like below snapshot.

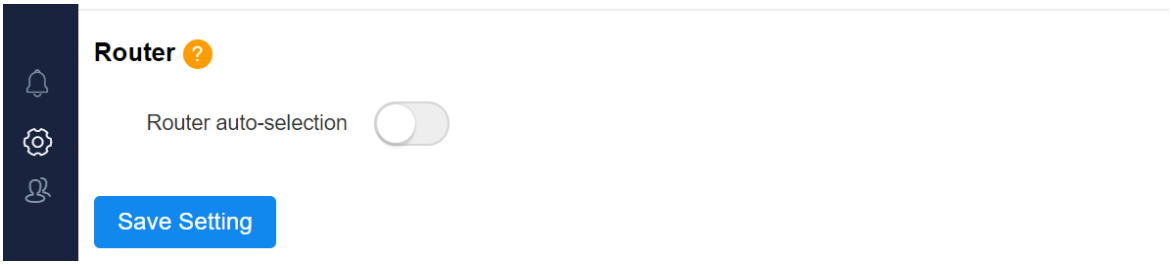


Figure 6: Router auto-selection configuration in AC

POST http://{your AC domain}/api/aps/ap-select-switch

Body example (application/json):
{ “flag”:1 }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: application/json
{ "status": "success", "flag": 1 }

6.6.2. Connect a device

This API will automatically select one router from a list of candidates, and use it to connected the device.

POST http://{your AC domain}/api/aps/connections/connect

Parameters for json Body:

- aps: the list of routers which will be used for this auto-select connect request. The user can use one or multiple router’s MAC or * for “aps”. If the user uses *, it means all the online routers that controlled by the AC should be included
- devices: only one device’ MAC address can be added in “devices”

Body example (application/json):
{ "aps": ["CC:1B:E0:E7:FE:F8","CC:1B:E0:E7:FE:F9","CC:1B:E0:E7:FE:FA"], "devices": ["F7:1
8:BC:18:F0:3A"] }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

6.6.3. Disconnect a device

This API will disconnect a device. In json Body, only one device’ MAC address can be added in “devices”.

POST http://{your AC domain}/api/aps/connections/disconnect

Body example (application/json):
{ "devices": ["F7:18:BC:18:F0:3A"] }

Response example:

Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

```
OK
```

6.7. SSE Combination APIs

These APIs simplify the handling of multiple routers. They also improve the scalability of AC in terms of the number of routers supported in given hardware resources.

NOTE: these APIs are only available through Cassia AC.

Before firmware 1.3, if an application wants to control routers with RESTful APIs through AC, the application has to create three SSE tunnels for each router: one for scan data, one for notification/indication data, and one for connected device status.

From firmware 1.3, the application only needs to create one SSE tunnel with AC. This SSE tunnel can receive scan data, notification/indication data, and connected device status for all routers controlled by this AC.

6.7.1. Create combined SSE

This API will create one combined SSE connection with AC. This SSE connection can receive scan data, notification/indication data, and connected device status for all the routers controlled by this AC.

GET http://{your AC domain}/api/aps/events

When invoke this API, AC will return a message immediately which include all router's information, for example:

```
data:
{"dataType":"state","aps":{"CC:1B:E0:E7:FE:F8":{"_id":"5a93755b028e6c00519
ce1dc","id":"CC:1B:E0:E7:FE:F8","mac":"CC:1B:E0:E7:FE:F8","name":"Cassia
Router","group":"","status":"online","model":"X1000","version":"1.2.0.1803
131043","position":"","time":1519613275655,"ip":"192.168.1.202","localip":
"192.168.1.202","uptime":14873,"offline_time":0,"online_time":1522052125,"
update_status":"update_ok","update_reason":"","update_version":"","update_
progress":0,"notify":true,"connection-
state":true},"CA:79:F5:B6:1F:04":{"devices":{"CA:79:F5:B6:1F:04":{"type":"
random","bdaddrs":{"bdaddr":"CA:79:F5:B6:1F:04","bdaddrType":"random"},"ch
ipId":0,"handle":"","name":"","connectionState":"connected","id":"CA:79:F5
:B6:1F:04"}}}}}
```

One keep-alive message will be returned every 30 seconds to make sure the SSE link is up and running.

If scanning is open, this SSE tunnel will send scanning data to user application through AC. For example:

```
data:
{"dataType":"scan","ap":"CC:1B:E0:E7:FE:F8","bdaddrs":[{"bdaddr":"CC:1B:E0
:E0:98:16","bdaddrType":"public"}],"adData":"0201060D084361737369615F53313
03030","name":"Cassia_S1000","rssi":-34,"evt_type":3}
```

If notification is open (default configuration), this SSE tunnel will return the notification messages to user application when any router has notification messages to AC.

```
data:
{"dataType":"notification","ap":"CC:1B:E0:E7:FE:F8","value":"FF000C0002051
00101010126","device":"CA:79:F5:B6:1F:04","handle":16}
```

If connection-state is open (default configuration), this SSE tunnel will return the device's connection status to user application when any device's status changes. For example:

```
data:
{"handle":"CA:79:F5:B6:1F:04","connectionState":"disconnected","dataType":
"connection_state","ap":"CC:1B:E0:E7:FE:F8"}

data:
{"handle":"CA:79:F5:B6:1F:04","connectionState":"connected","dataType":"co
nnection_state","ap":"CC:1B:E0:E7:FE:F8"}
```

If ap-state is open (default configuration), this SSE tunnel will return the ap-state information when router's status is changed between online and offline. For example:

```
data:
{"dataType":"ap_state","ap":"CC:1B:E0:E7:FE:F8","mac":"CC:1B:E0:E7:FE:F8",
"status":"offline","offline_time":1522067273296}
```

6.7.2. Open scan

This API will open router scanning for all the routers in the router list. The SSE tunnel will receive scan data.

POST http://{your AC domain}/api/aps/scan/open

Body example (application/json):
{"aps":["CC:1B:E0:E7:FE:F8","CC:1B:E0:E7:FE:F8","CC:1B:E0:E7:FE:F8","CC:1B:E0:E7:FE:F8
"],"chip":0,"active":0,"filter_name":"cassia"}

Parameters for json Body:

- aps (mandatory): one or multiple router's MAC address
- chip (optional): 0 or 1. It means which chip to scan
- active (optional): 0 or 1. 0 means enable passive scanning; 1 means enable active scanning
- filter_name (optional): filter for device name
- filter_mac (optional): filter for device MAC
- filter_uuid (optional): filter for device UUID

Response example:

Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

6.7.3. Close scan

This API will close the router scanning for all the routers in the router list. The user application will not receive scan data anymore.

```
POST http://{your AC domain}/api/aps/scan/close
```

Parameters for json Body:

- aps: one or multiple router's MAC address

Body example (application/json):
{ "aps": ["CC:1B:E0:E7:FE:F8", "CC:1B:E0:E7:FE:F8"] }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

6.7.4. Open notify

This API will open the notification messages on SSE tunnel. The notification data will be sent to the user application on this SSE tunnel.

```
POST http://{your AC domain}/api/aps/notify/open
```

Parameters for json Body:

- aps: one or multiple router's MAC address

Body example (application/json):
{ "aps": ["CC:1B:E0:E7:FE:F8"， "CC:1B:E0:E7:FE:F8"] }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

6.7.5. Close notify

This API will close the notification messages on SSE tunnel. The notification data will not be sent to the user application on this SSE tunnel anymore.

POST http://{your AC domain}/api/aps/notify/close

Parameters for json Body:

- aps: one or multiple router's MAC address

Body example (application/json):
{ "aps": ["CC:1B:E0:E7:FE:F8", "CC:1B:E0:E7:FE:F8"] }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

6.7.6. Open connection-state report

This API will open the connection-state monitoring on SSE tunnel. The connection-state data will be sent to the user application on this SSE tunnel when the state of the connected device changed.

POST http://{your AC domain}/api/aps/connection-state/open

Parameters for json Body:

- aps: one or multiple router's MAC address

Body example (application/json):
{ "aps": ["CC:1B:E0:E7:FE:F8", "CC:1B:E0:E7:FE:F8"] }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

6.7.7. Close connection-state report

This API will close the connection-state monitoring on SSE tunnel. The connection-state data will not be sent to the user application on this SSE tunnel anymore.

POST http://{your AC domain}/api/aps/connection-state/close

Parameters for json Body:

- aps: one or multiple router's MAC address

Body example (application/json):
{ "aps": ["CC:1B:E0:E7:FE:F8", "CC:1B:E0:E7:FE:F8"] }

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

6.7.8. Open ap-state report

This API will open the ap-state monitoring for all routers on SSE tunnel. The data of ap-state will be sent to the user application on this SSE tunnel when the router state changed between online and offline.

```
GET http://{your AC domain}/api/aps/ap-state/open
```

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

6.7.9. Close ap-state report

This API will close the ap-state monitoring for all routers on SSE tunnel. The data of ap-state will not be sent to the user application on this SSE tunnel anymore.

```
GET http://{your AC domain}/api/aps/ap-state/close
```

Response example:
Status-Line : HTTP/1.1 200 OK/r/n
Header : (general-header)
Message-body: text/plain

OK

7 Bluetooth Debug Tool

We integrated Cassia RESTful APIs into a Bluetooth debug tool with a visual interface. After a RESTful API call, the debug tool will show the response messages. It will help developers to integrate business applications and Bluetooth devices with the Cassia Bluetooth router and

AC. This tool is available at http://www.bluetooth.tech/nativeHubControl/index.html

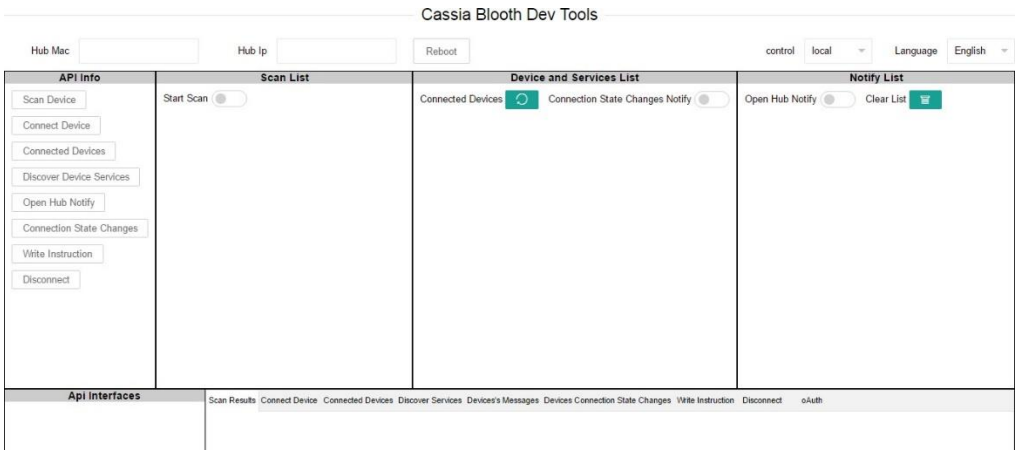


Figure 7: Cassia Bluetooth Debug Tool

API Info: This area contains the user's most commonly used commands. When you click on the buttons, the parameters and descriptions related to that command will pop up.

Scan List: When turning on this option, 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.

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.

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.

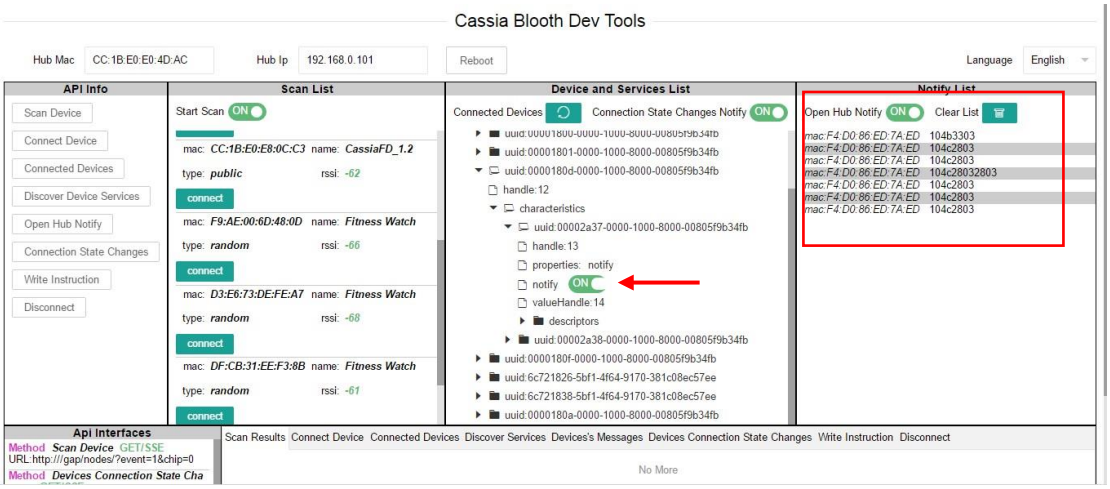


Figure 8: Cassia Bluetooth Debug Tool example

From firmware 1.4, Cassia AC integrated Bluetooth debug tool. For more information, please check Cassia AC Bluetooth Debug Tool User Guide at https://www.cassianetworks.com/knowledge-base/general-documents/

8 Error Messages

NOTE: For Connect and Pair APIs, avoid calling a new API before the previous API call has finished, otherwise the router will respond with a “chip is busy” error.

For HTTP 500 error, the following are the common error codes which is included in the content of HTTP response message.

- "parameter invalid": wrong parameter value, such as chip ID, MAC address or advertise type is wrong.
- "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.
- "memory alloc error": when the Bluetooth chip does not have enough memory to complete the operation it will return this error.
- "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 the 20s.
- "chip is not ready": this error will be reported when sending commands to the chip fails.
- "chip is busy": the Connect and Pair API are mutually exclusive. If calling a connect request before the previous connect request finishes, the system will return "chip is busy".
- "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.
- "device not connect": same as "device not found" error.
- "operation not supported": reserved for future use.
- "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.
- "no resources": the Bluetooth chip in Cassia routers can store the pair information up to 10 seconds. If you pair too many devices, the system will report this error.
- "Service Not Found": couldn't find a Characteristic inside a Service.
- "type not supported": When Bypass scan was set up, the protocol type you specified is not supported by this firmware.
- "please set bypass params first": Bypass mode is enabled, but no bypass parameters have been set.
- "failure": an error for all other failures not specified yet.

Appendix A -- 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 B -- Sample Code to Get Access Token

```
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);
     });
 });
```