API Guide

8.10 ARUBA IOT WEBSOCKET INTERFACE

Aruba WLAN IoT WebSocket Interface Documentation

a Hewlett Packard
Enterprise company

CHANGELIST

Version	Date	Notes
0.4	04/04/2022	Updated text and added examples describing external BLE radio usage for BLE Connections
0.3	10/20/2021	Updated certain BLE southbound api examples from AOS 8.9
0.2	06/15/2021	Updated with generic filters and USB serial device type fillter
0.1	06/15/2021	Initial Document Revision

Copyright Information

Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open-source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by Hewlett Packard Enterprise Company. To obtain such source code, send a check or money order in the amount of US $10.00 to:

Hewlett Packard Enterprise Company 6280 America Center Drive
San Jose, CA 95002
USA

CONTENTS

1. Introduction

Aruba WLAN provides a public Internet of Things (IoT) interface for applications like location services, IoT device management applications, etc. The IoT interface sets up a connection between the Aruba WLAN and the partner application. Customers can configure the IoT interface by creating an IoT transport profile on the Controller or Instant AP. The IoT transport profile can be customized to serve a diverse set of applications. This document describes the “Telemetry WebSocket” server interface for transporting IoT data such as telemetry reports, commands or IoT data packets. This document is intended for application developers who want to build a secure and scalable IoT application that can consume the IoT data made available over the WebSocket interface.

The following is a list of frequently used terms in the document:

- Access Point (AP) – The Aruba AP contains the IoT radio. The communication with IoT devices will be referred to as communication between the remote device and the AP.
- ArubaOS (AOS) – This is the software that runs on Aruba WLAN infrastructure (APs, controllers, Aruba Instant), containing the Aruba code for the IoT transport profiles.
- Northbound – This is communication from Aruba WLAN infrastructure to the partner application.
- Southbound – This is communication from the partner application to the Aruba WLAN infrastructure.
- IoT Transport Profile – This is the profile that the user will configure on the WLAN management console to setup the transport between Aruba WLAN and the partner application.

a. Solution Overview

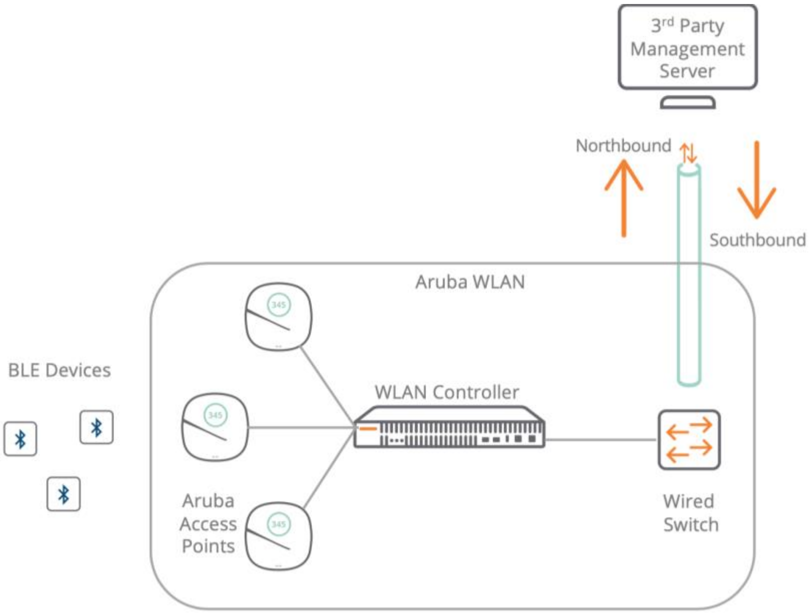


Figure 1: Overview of Aruba WLAN deployment (controller-based) with IoT Telemetry WebSocket Interface.

Figure 1 shows an overview of the different components involved in transporting IoT data between the Aruba WLAN infrastructure and the partner application/management server when the “Telemetry WebSocket” server type is configured by the user. As suggested by the name, the data is transported over a WebSocket connection. It is highly recommended to use a secure WebSocket connection to improve data confidentiality and reliability. The messages being sent over the WebSocket are formatted using Google Protocol Buffers version 2. The message structure is defined in proto definition files available on the Aruba ASP portal (search for ArubaOS WLAN InstantOS 8.10.0.x IoT Interface - Protobuf Specification on https://asp.arubanetworks.com/downloads).

For controller-based deployments, each controller establishes a single WebSocket connection per transport profile (of type Telemetry WebSocket) to the 3rd party server. Traffic from multiple APs goes over the same, shared WebSocket connection in case of controller-based deployments. In cases where multiple APs are connected to a controller, the controller becomes a bottleneck as the amount of IoT traffic increases. This requires end-users to configure the IoT transport profile with the appropriate parameters so that messages from the APs are not dropped at the controller due to resource contention (inadequate buffers). In the case of controller-less/Instant deployments, each Instant AP opens its own WebSocket connection per transport profile to the server. This shifts the scaling burden from the WLAN infrastructure to the 3rd party server, which will now need to support multiple WebSocket connections per deployment.

b. Transport Services

Once the IoT interface connection is established, it can transport data for the different IoT transport services shown in Table 1. The transport services are a UI-only concept, first made visible in ArubaOS and Instant 8.9. Once one or more transport services are selected in the UI, the user can setup the different configuration options available for each specific transport service.

Service	Purpose
BLE Telemetry	Periodic telemetry reports with structured data from each device
BLE Data	BLE advertisement frames and Scan-Response frames
Wi-Fi Data	Periodic reports about nearby Wi-Fi clients
Zigbee Data	Zigbee data frames to/from Zigbee socket devices
Serial Data	Data frames to/from USB devices plugged into the AP

Table 1: Transport Services enabled by IoT transport profiles.

To setup a transport service, the appropriate knobs need to be configured in the IoT transport profile. The transport profile knobs for the “Telemetry WebSocket” server interface are described in detail in Chapter 2. Except for the WiFi data transport services, all other transport services allow for Southbound communication with the IoT devices over the WebSocket interface. The message structure for the Northbound and Southbound messages is defined in the proto definition files available on the Aruba ASP portal.

Note: In previous versions of the Aruba IoT WebSocket Interface Guide, “BLE Connections” was described as a separate transport service. While this is no difference in functionality from previous ArubaOS or Instant versions, the “BLE Connections” (described in Chapter 7) feature is always available whenever the BLE Telemetry or BLE Data transport services are selected via the UI, hence it is no longer listed as a separate service.

Note: Please refer to the IoT section in the ArubaOS and Aruba Instant 8.10.0.0 User Guides (https://asp.arubanetworks.com/downloads) for information on additional profiles such as the IoT radio profile that are required for proper operation of the AP’s IoT radio/s to support the workings of the IoT transport profiles.

2. Configuration

Table 2 lists the attributes in the IoT transport profile that are applicable when the “Telemetry WebSocket” server type is selected in the transport profile configuration.

Category	Name	Description
Destination	Server URL	Server URL for sending telemetry
	Proxy	Proxy server for sending telemetry
Frequency	Reporting Interval	Reporting interval in seconds
Authentication	Authentication Mode	OAuth2 Authentication Mode (None/Password/Client-Credentials)
	Authentication URL	Server URL for authentication
	Username	Username for authentication
	Password	Password for authentication
	Access Token	String used by server to separate traffic from multiple entities (IAP/controller)
	Client ID	This ID identifies the sender to the server
	Client Secret	Authentication parameter used in conjunction with client ID
Device Filters	Device Class	A list of device class tags to filter the devices included in the reports
	Company Identifier	A list of BT SIG company identifiers
	Service UUID	A list of 16-bit service UUIDs
	Local Name	A list of local name sub strings
	MAC OUI	A list of MAC OUI values
	UUID	A list of UUIDs to filter the devices included in the reports. Applies only to iBeacon devices
	UID Namespace	A list of UID namespaces to filter devices included in the reports. Applies only Eddystone-UID devices
	URL	A list of URL strings to filter devices included in the reports. Applies only Eddystone-URL devices. The string listed here can be partial URL strings
	Cell Size	A proximity filter. Devices outside the cell will not be reported. Size is specified in meters. Setting to 0 disables the cell size filter
	Movement	Filters devices that do not change distance. Specified in meters. Applicable only if a cell size is set. Setting to 0 disables the movement filter
	Age	Age filter. Devices without recent activity will not be reported
	Vendor	A list of Vendor IDs or Vendor Names which are used to filter reporting
	ZSD	A set of Zigbee Socket Devices to filter. This applies only to devices that conform to the ZSD device class
	RTLS Dest. MAC	Sets the destination MAC address filter for RTLS tags device class.
	USB Serial Device Type	Specify the type of USB device from a list of available supported types.
Content specifiers	RSSI Reporting Format	Set the preferred format for RSSI reporting
	Environment type	The type of environment that the APs are deployed in. The environment determines the RF fading factor that is used for the translation from RSSI to distance
	Custom Fading Factor	For manually setting the fading factor. Applies only when Environment Type is set to Custom
	Device Count Only	For those interested in a count of devices seen, but not the actual content of those devices
	Data Filter	This is a mechanism to suppress fields in the telemetry reports, that are not required by the receiver
	BLE data forwarding	Forwards raw BLE payload for devices with known class labels
	Per Frame Filtering	Check device class of every BLE frame before forwarding it

Table 2: IoT Transport Profile Configuration Parameters relevant to “Telemetry WebSocket” server type. For more information on the proper values and formats of the configuration parameters mentioned above, please refer to the ArubaOS and Instant User Guide for 8.9 on the Aruba ASP portal (https://asp.arubanetworks.com/downloads).

While most of the above fields are self-explanatory, a deeper explanation of some of the configuration parameters shown in Table is provided below:

- **Company Identifier Filter**

  A company identifier filter will only report BLE devices that contain at least one of the configured values of the BT SIG registered company identifier as part of their advertisements or scan response packet payloads. For example: iBeacon packets contain Apple's BT SIG identifier 0x004C in the manufacturer specific advertising data element. The user can filter on iBeacon packets by specifying Apple's BT SIG identifier (004C) as part of the IoT transport profile configuration.

- **Service UUID Filter**

  A service UUID filter will only report BLE devices that contain at least one of the configured values of the BT SIG registered 16-bit service UUIDs as part of their advertisements or scan response packet payloads. For example: Eddystone packets contain Google's BT SIG service UUID 0xFEAA in the 16-bit service UUID advertising data element. The user can filter on Eddystone packets by specifying Google's BT SIG 16-bit service UUID (FEAA) as part of the IoT transport profile configuration.

- **Local Name Filter**

  The Local Name Filter will only report BLE devices that contain at least one of the configured sub-string values in the local name advertising data element in a device's advertisements or scan response packet payloads.

- **MAC OUI Filter**

  User can input a comma-separated list of MAC OUI values (should not include ":" as separator between the bytes of the MAC OUI). This filter will only report BLE devices wherein their MAC address has the same MAC OUI as that in the list of configured values.

- **USB Serial Device Type Filter**

  Prior to the availability of this knob, the serial data from ALL the connected USB devices would be forwarded to ALL the transport profiles. Now, the user can specify the USB device type from which data should be forwarded to a particular transport profile when the serial data device class filter is configured. For example: In a deployment, if APs in one area have EnOcean devices and Piera devices in a different area, then the user can configure one transport profile with filter set to EnOcean and another transport profile with filter set to Piera. This way, the data from the EnOcean devices only goes to the profile with the EnOcean USB serial device filter, and data from the Piera device will only be transported to the profile with Piera USB serial device filter.

- **Cell Size Filter**

  A proximity-based filter that will only report devices that are found to be within an "x" meter radius around the access point. This distance is calculated with an algorithm based off the RSSI value. The default value for this field is "0", which translates to the cell size filter being disabled. This field accepts integer values from 2 to 100, and the unit is meter.

- **Movement Filter**

  This filter is active when the cell size filter is also configured. When this filter is enabled, devices will only be reported if the difference between their current and prior distance is more than the configured filter value. For example, if the movement filter is configured to be 2 meters, a device that is calculated to have moved 1 meter will not be reported, while a device that moves 5 meters will be reported. The default value for this field is "0", which corresponds to the movement filter being disabled. This field accepts integer values from 2 to 30, and the unit is meter.

- ## Age Filter

  The Age Filter is used to only report devices in which we have received an update (either BLE advertisement or scan response) in the configured time. For instance, if the age filter is set to 30 seconds, only devices which have been heard in the last 30 seconds will be reported. If there is a device that received an update 45 seconds before, this device will not be reported. The default value for this field is "0", which corresponds to the age filter being disabled. This field accepts integer values from 30 to 3600, and the unit is second.

- ## Vendor Filter

  The Vendor Filter allows a user to input either Bluetooth SIG Vendor IDs, or freeform Vendor Name strings, which will be used to filter the devices reported. If this is configured, the only devices that will be reported are the devices that match the configured Vendor ID or Vendor Name.

- ## RSSI Reporting Format

  We currently support five different RSSI reporting formats when sending reports to subscribers. The reporting formats are:
  - Last: Only the last RSSI value that was observed for the device will be reported.
  - Average: The average RSSI over the reporting interval will be reported.
  - Max: The maximum RSSI value that was seen over the reporting interval will be reported. This maximum value resets each telemetry reporting interval and will be updated accordingly.
  - Bulk: The last 20 RSSI values that were observed for the device since the previous telemetry report will be reported in an array format.
  - Smooth: A single smoothed out RSSI value will be reported for each telemetry report. This is done by attempting to remove outliers from the RSSI values received by the AP.

- ## Environment Type

  We currently support five different pre-defined environment types to help adjust RSSI based distance values to better fit the environment in which the BLE devices are operating in as follows:
  - Auditorium:
  - Office:
  - Outdoor:
  - Shipboard:
  - Warehouse:

  For best results, you should choose the value that closest corresponds to the environment in which BLE is operating.

- ## Custom Fading Factor

  If the above environment type offsets do not properly fit your environment, a custom fading factor can be configured which is a custom environment type. This value will only be considered if "Environment Type" is configured to custom. This field accepts integer values in the range of 10 to 40.

- ### Data Filter

  This is a list of fields to suppress in the telemetry reports. The data filter is a string that is a comma separated list of index-paths. Each index path refers to the protobuf field numbers. For example, the value “3.3, 3.12” would suppress the ‘reported.model’ field and the ‘reported.beacons’ field in the telemetry reports.

- ### BLE Data Forwarding and Per Frame Filtering

  When BLE data forwarding is enabled, the raw payload contained within a BLE packet is forwarded to the configured server. The per frame filtering knob is a modifier on top of the BLE data forwarding knob. When only BLE data forwarding is enabled, all BLE packets for a device having a known device class filter label are forwarded. For example: If a device advertises an iBeacon frame and an Eddystone frame, and in the transport profile we have selected only iBeacon, then for this device we will forward both iBeacon and Eddystone frames. Now, if we enable the per frame filtering knob in addition to the BLE data forwarding knob, then only the raw payloads from the iBeacon frames will be forwarded.

Note: Some of the configuration options might be shown in the UI only for a particular transport service. For example: USB Serial Device Type will be displayed only when the Serial Data transport service is selected. On the other hand, the CLI will display and allow users to select certain combinations which do not go together in practice. In such cases, that specific parameter configuration will be ignored, and the transport profile behavior might not match user expectation.

Note: While transport services are a UI only artifact, some of the configuration parameters in the IoT transport profile are enabled (or disabled) when a transport service is selected (or deselected). In the transport profile UI, BLE data transport service can be enabled/disabled since it maps to the BLE data forwarding parameter described previously. On the other hand, BLE Telemetry transport service is always on (cannot be disabled) whenever BLE device classes (or generic filters such as company ID, service UUID, local name, MAC OUI) are configured in the IoT transport profile. The “reporting interval” parameter controls the frequency of the BLE telemetry messages. For applications that do not require periodic BLE telemetry messages, users are advised to configure the reporting interval to a high value.

3. Authentication and Authorization

For the IoT transport profile, authentication is optional. When authentication is desired, there are two options available to the user:

1. User Credentials: user needs to configure an authentication URL, username, password, and client ID in the profile.
2. Client Credentials: user needs to configure an authentication URL, client credentials and client ID in the profile.

Authorization is expressed using an access token that is present in every message sent to the server. In an authenticated connection, the access token is obtained during authentication. If authentication is not required, then user only needs to populate the server URL and access token.

Note: In the IoT transport profile UI, even though "Token" is grouped along with "User ID/Password" and "Client Credentials" under Authentication, note that the access token is meant to be used for authorization purpose only. For secure, scalable communication with the server, it is strongly advised to use the "User ID/Password" and "Client Credentials" methods in any customer deployment.

a. Authentication Handshake

Authentication is always done using HTTPS, subsequent API calls are done using secure WebSockets.

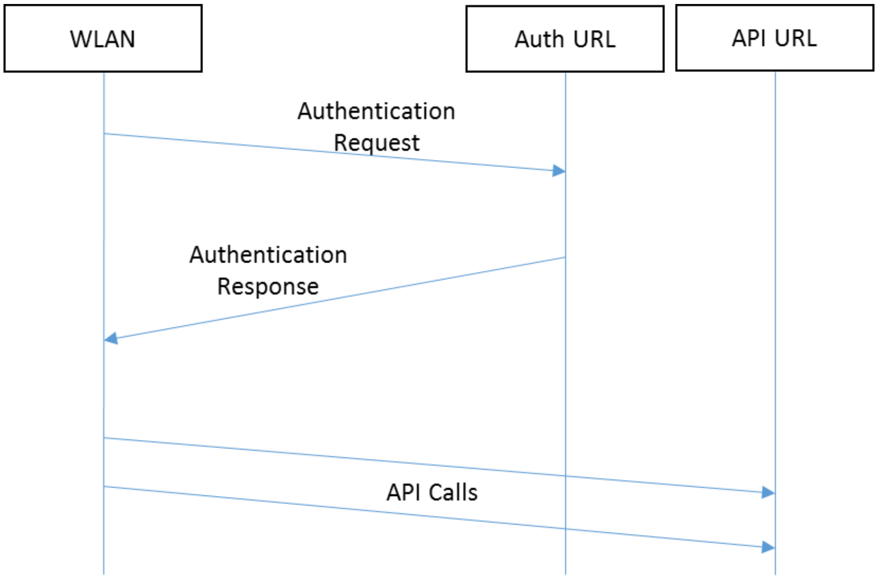


Figure 2: Authentication Workflow

b. Authentication Request

This is an HTTPS POST operation. Depending upon the type of authentication (User Credentials/Clients Credentials), the HTTPS POST body contains different JSON content as follows:

Sample JSON when User Credentials are configured

```
{
    "grant_type": “password”,
    "username": <username>,
    "password": <password>,
    "client_id": <ClientID>,
    "scope": “Aruba_IoT_Framework”
}
```

Sample JSON when Client Credentials are configured

```
{
    "grant_type": “client_credentials”,
    "client_secret": <Client Secret>,
    "client_id": <ClientID>,
    "scope": “Aruba_IoT_Framework”
}
```

- When using “User Credentials”, the “username”, “password” and “client id” fields are taken verbatim from the IoT transport profile.
- When using “Client Credentials”, the “client secret” and “client ID” fields are taken verbatim from the IoT transport profile.
- If there is no client ID configured in the IoT transport profile, then the “client_id” field will be omitted from the JSON in the POST body.

c. Authentication Response

For successful authentication, we look for the following content in the response body:

```
{
    "access_token": <access_token>,
    "token_type": "bearer",
    "refresh_token": <refresh_token>,
    "expires_in": <time>,
    "api_url": <API URL>,
}
```

- “access_token” is mandatory.
- “token_type” is optional, but if it is present, the value must be set to “bearer”.
- “refresh_token” is optional and will be used for token refresh if present instead of a full re-authentication.
- “expires_in” is optional, but if it is not present, we assume that the token is valid until we receive an error.
- “api_url” is optional. If it is present, we will use this value rather than the server URL specified in the IoT transport profile.

We currently look for the following errors for authentication:

Error Code	Error Reason	Action
401	Unauthorized	Upon reception of this error, the system will try to authenticate again

For Telemetry WebSocket connections, the access token will be included in the payload of every message. Please see the telemetry proto files for the exact definition. In the return messages, the server can include a status message to indicate if the token was invalid.

d. Token Expiration and Token Refresh

During the authentication handshake, if we receive the “expires_in” field along with the access token, we will support token expiration. As soon as the authentication handshake happens, we will store the “expires_in” time, and after that amount of time, we will consider the access token invalid. We currently support a minimum “expires_in” time for 300s, and a maximum time of 1 month. At this point we have two options, either a full re-authentication where we redo an authentication handshake and restart the connection, or if the server provided a refresh token, we would attempt to refresh the access token using the refresh token.

If a refresh token is provided, instead of immediately tearing down the connection, we will first attempt to get a fresh access token from the server. This refresh will use a grant type of “refresh_token” instead of a grant type of “password”. If we are unable to get a new access token, or the server returns a 401 Error Code, we will fall back to tearing down the connection and redoing the full authentication handshake.

```
{
    "grant_type": “refresh_token”,
    "refresh_token": <Refresh Token>,
    "client_id": <ClientID>,
    "scope": “Aruba_IoT_Framework”
}
```

4. AP Health Information

Once the WebSocket connection is set up, each AP will send an AP Health message to the server every 120s. The message contains information about the reporter AP, the onboard and external radios, and any other supported external USB serial devices.

Sample AP Health Update Message

```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "apHealthUpdate"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.9.0.0-8.9.0.0",
    "swBuild": "81161",
    "time": "1630102303"
  },
  "apHealth": {
    "apStatus": "healthy",
    "radio": [
      {
        "mac": "204c0339e22c",
        "hardware": "gen2",
        "firmware": "arubaDefault",
        "health": "healthy",
        "external": false
      }
    ],
    "usb": [
      {
        "identifier": "ENOCEAN_USB:f6a68e740ecc549496d4b63072a33920",
        "health": "healthy"
      }
    ]
  }
}
```

5. BLE Telemetry

The BLE telemetry transport service sends periodic reports about all the BLE devices that are discovered by an AP. The AP will continually listen for advertisements and scan responses. The AP will parse/decode these packets to the best of its abilities and update the telemetry in its internal table. Periodically, the contents of this table will be reported as BLE telemetry data. Once an IoT transport profile is properly configured on the AP/Controller, northbound telemetry messages will be sent to the server at every reporting interval. This will continue indefinitely until the profile is removed. These telemetry reports contain a summary of all the BLE devices that are seen by a particular AP. For each individual BLE device, we only populate the information that we have for the device.

Note: In AOS and Instant 8.9, the BLE telemetry transport service is always enabled.

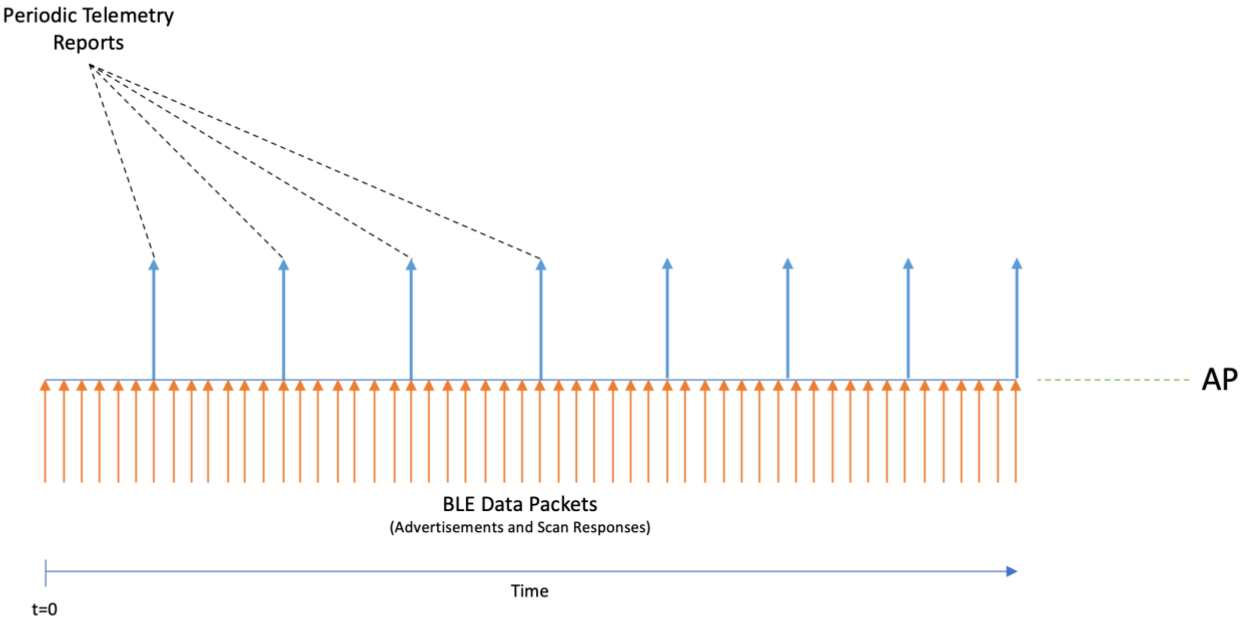

Periodic Telemetry Reports
AP
BLE Data Packets
(Advertisements and Scan Responses)
Time
t=0

Figure 3: Example of an IoT transport profile with periodic telemetry reports

Sample Telemetry Message

```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "telemetry"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.9.0.0-8.9.0.0",
    "swBuild": "81161",
    "time": "1630100306"
  },
  "reported": [
    {
      "mac": "00a05011880a",
      "deviceClass": [
        "eddystone",
        "abbSensor"
      ],
      "lastSeen": "1630100303",
      "bevent": {
        "event": "update"
      },
      "rssi": {
        "avg": -77
      },
      "beacons": [
        {
          "eddystone": {
            "power": -20,
            "uid": {
              "nid": "1000800000805f9b0131",
              "bid": "00a05011880a"
            }
          }
        }
      ],
      "sensors": {
        "temperatureC": 0
      },
      "stats": {
        "frame_cnt": 3
      },
      "vendorName": "ABB"
    },
    {
      "mac": "a0e6f82c080f",
      "deviceClass": [
        "arubaBeacon",
        "iBeacon"
      ],
      "model": "LS-BT20",
      "firmware": {
        "bankA": "1.2-15"
      },
      "lastSeen": "1630100306",
      "bevent": {
        "event": "update"
      },
      "rssi": {
        "avg": -87
      },
      "beacons": [
        {
          "ibeacon": {
            "uuid": "4152554ef99b4a3b86d0947070693a78",
            "major": 3001,
            "minor": 3001,
            "power": -69
          }
        }
      ],
      "txpower": 10,
      "sensors": {
        "battery": 50
      },
      "stats": {
        "uptime": "10380",
        "frame_cnt": 16
      },
      "vendorName": "Aruba",
      "companyIdentifier": [
        {
          "value": 76,
          "description": "Apple, Inc."
        },
        {
          "value": 283,
          "description": "Hewlett Packard Enterprise"
        }
      ]
    }
  ]
}
```

6. BLE Data

The BLE data transport service forwards all BLE advertisement and scan response frames from all classified BLE devices (match device classes or generic filters such as company identifier, service UUID, local name and MAC OUI) as configured in the transport profile. When this transport service is enabled via UI, the BLE Data Forwarding knob is set in the transport profile. The Per Frame Filtering knob allows for stricter filtering by only forwarding frames that match the configured filter, i.e., any frames originating from the BLE device that do not match the configured filter is not forwarded. These options increase the traffic over the WebSocket, as you will receive a message for every BLE advertisement and scan response for eligible devices.

Starting AOS/Instant 8.10, when an external BLE radio (the Aruba AP-USB-ZB USB device) is plugged into the AP's USB port, in each BLE data message the AP will include the "apbMac" field which will indicate the BLE radio reporting the BLE device information.

Note: BLE data forwarding happens in addition to the periodic telemetry reporting. The two happen in parallel. If BLE data forwarding is the main method for which a subscriber would like to receive data, a high "reporting interval" value should be configured in the IoT transport profile.

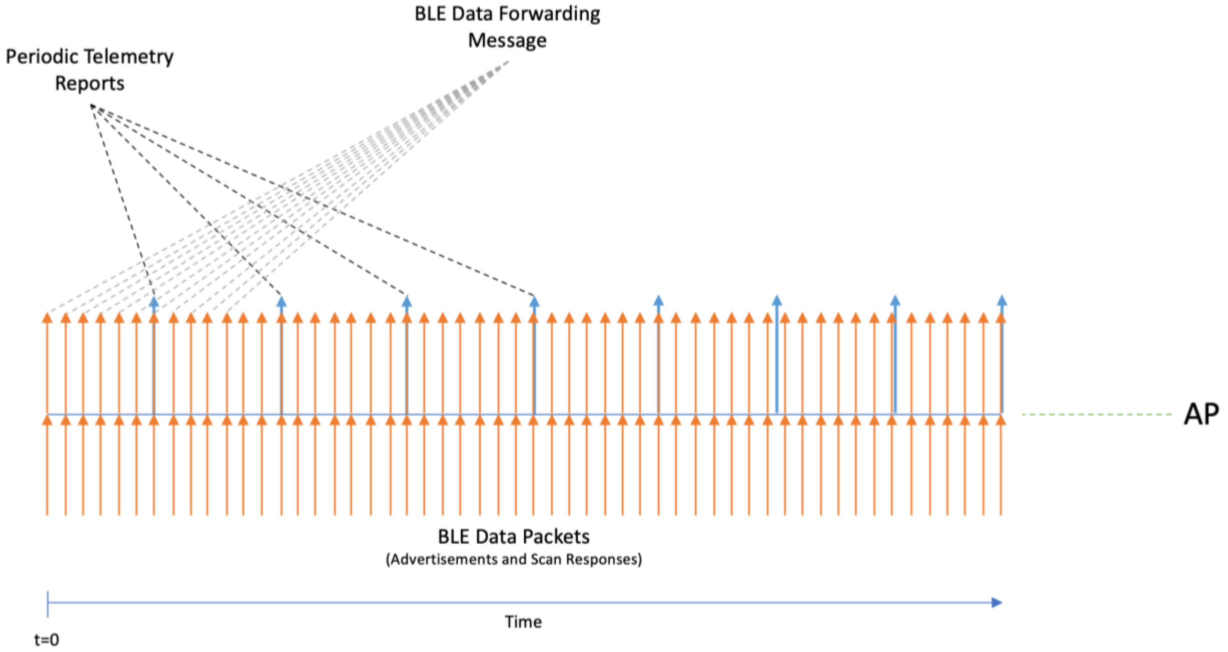

BLE Data Forwarding Message
Periodic Telemetry Reports
AP
BLE Data Packets
(Advertisements and Scan Responses)
Time
t=0

Figure 4: Example of an IoT transport profile with BLE data forwarding

Sample BLE Data Forwarding Message

```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "bleData"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.9.0.0-8.9.0.0",
    "swBuild": "81161",
    "time": "1630176248"
  },
  "bleData": [
    {
      "mac": "a0e6f82c09c9",
      "frameType": "adv_ind",
      "data": "0201061aff4c0002154152554ef99b4a3b86d0947070693a7800000000b5",
      "rssi": -58,
      "addrType": "addr_type_public",
      "apbMac": "204c0339e22c"
    }
  ]
}
```

7. BLE Connections

The BLE Connections feature allows the configured IoT server to connect to BLE devices using the Southbound APIs. ArubaOS has pre-defined primitives to connect and interact with BLE devices remotely via the SB IoT interface. This allows our partners to reach out and manage their devices via the Aruba WLAN infrastructure. This service is generic to all BLE devices. The operations map closely to the BLE GATT protocol.

Primitive	Description
Connect	Scan for a BLE device, set up a connection and discover characteristics
Disconnect	Disconnect an active connection
Read	Read from a BLE device using the GATT protocol
Write	Write to a BLE device using the GATT protocol
Notifications	Subscribe for notifications from a BLE device
Indications	Subscribe for indications from a BLE device
Authenticate	Enable supported BLE authentication method
Encrypt	Encrypt BLE data using the bonding key

a. Encoding

The commands and responses for the BLE connections service are all messages going up and down the IoT Interface WebSocket. The messages are encoded using the Google Protocol Buffer method. We use the proto2 version of the protocol. The definitions can be found on the Aruba ASP portal.

When it comes to the .proto definitions, you will notice all fields are listed as optional. This is by design to increase the forward compatibility of the API definitions in the .proto file. While this is the case, each API call has some fields that must be supplied to properly process the specified operation. These will be defined for each operation.

Starting AOS/Instant 8.10, Aruba AP's support BLE SB API using the external BLE radio when an AP-USB-ZB USB device is plugged into the AP's USB port. The "apbMac" field was added to the Aruba protobuf specification to allow the SB API caller to specify which BLE radio (internal/external) they want to use when invoking the SB API. The AP radio information is communicated periodically in the apHealthUpdate message. The "apbMac" field is optional, thus enabling backward compatibility with older BLE SB API applications. If the "apbMac" field is absent on an AP with multiple BLE radios, then the AP will select the best possible BLE radio for the SB API actions. The selection criteria are based upon RSSI of the BLE device as reported by each of the BLE radios. Typically, we will attempt to select the BLE radio that reported the stronger RSSI.

Note: To enable BLE for the AP-USB-ZB USB device, configure an IoT radio profile with radio instance set to external, radio mode set to BLE and the BLE operational mode set to "beaconing scanning".

Southbound Action Message Fields Explained

An overview of the fields in a Southbound Action message are in the chart below. This is a superset of all the fields that might be present in a southbound action. Not all the fields should be present for every command.

Field	Value	Description

meta		
version	uint64	Version of .proto definition. Currently only supported version is “1”
sbTopic	SbTopic Enum	Enum value as defined in aruba-iot-types.proto file
receiver		
apMac	MAC Address	MAC Address of AP which will process the action
actions		
actionId	string	3rd Party Server defined string to correlate responses to actions
type	ActionType Enum	Enum value as defined in aruba-iot-types.proto file
deviceMac	MAC Address	MAC Address of remote BLE device
apbMac	MAC Address	MAC Address of AP’s BLE radio
serviceUuid	bytes	String containing either 16bit or 128bit UUID for GATT Service
characteristicUuid	byes	String containing either 16bit or 128bit UUID for GATT Characteristic
timeOut	uint32	Timeout value in seconds for the action to be completed
value	bytes	Data in a byte format to be written to characteristic in write commands
authentication	Authentication submessage	BLE Security/authentication related information
bondingKey	BleBondingKey submessage	Encryption key which is generated when bonding is enabled for pairing
status		
connectCode	ConnectCode Enum	Enum value defined in aruba-iot-sb-status.proto
connectDescription	string	The server response description for the connection code

Northbound Action Status Message Fields Explained

After specific actions have been completed, status messages will be returned to the 3rd party server. These messages will differ based on the type of action that was completed. For each action type, the expected response will be described. An overview of the different fields that can be present in a response follow.

Note: Only fields that pertain specifically to BLE connections are explained here.

Field	Value	Description
meta		
version	uint64	Version of .proto definition. Currently only supported version is “1”
accessToken	string	Access Token present in all northbound frames which the server must verify
nbTopic	NbTopic Enum	Enum value as defined in aruba-iot-types.proto file

reporter		
name, mac, etc.	Varies	Information about the AP that processed the action will be listed here
actionResults		
actionId	string	3rd Party Server defined string to correlate responses to actions
type	ActionType Enum	Enum value as defined in aruba-iot-types.proto file
deviceMac	MAC Address	MAC Address of remote BLE device
apbMac	MAC Address	MAC Address of AP's BLE radio
status	ActionStatus Enum	Enum value as defined in aruba-iot-nb-action-results.proto file
statusString	string	Optional additional freeform information
bondingKey	BleBondingKey submessage	Encryption key which is generated when bonding is enabled for pairing
characteristics		
deviceMac	MAC Address	MAC Address of remote BLE device
serviceUuid	bytes	String containing either 16bit or 128bit UUID for GATT Service
characteristicUuid	bytes	String containing either 16bit or 128bit UUID for GATT Characteristic
value	bytes	Value populated after read commands or from notifications
description	string	GATT Characteristic description
Properties	CharProperty Enum	Enum value as defined in aruba-iot-nb-characteristics.proto file
status		
deviceMac	MAC Address	MAC Address of remote BLE Device
status	StatusValue Enum	Emun value as defined in aruba-iot-nb-status.proto
statusString	string	Additional freeform information string
connUpdate	ConnUpdate submessage	The negotiated MTU Value between BLE central and peripheral

b. Command Overview

i. bleConnect

The bleConnect command will send an operation for a specific AP to attempt to connect to a specified BLE device. There are no prerequisites to this command. In response to a successful bleConnect action, the server gets an "MTU Update" message (nbTopic is set to "status"), followed by a "Connection Successful" message (nbTopic is set to "actionResults").

Request Required Parameters	• ActionId • Type

	- Device MAC
Request Optional Parameters	- Timeout
Possible Responses	- success - actionTimeout - apNotFound - deviceNotFound - alreadyConnected

Example Request

```
{
        "meta": {
                "access_token": "0123456789",
                "version": 1,
                "sbTopic": "actions"
        },
        "actions": [{
                "deviceMac": "e4:f2:05:7e:e8:68",
                "serviceUuid": null,
                "value": null,
                "characteristicUuid": null,
                "actionId": "10000001",
                "timeOut": 30,
                "type": "bleConnect"
        }],
        "receiver": {
                "apMac": "90:4c:81:cf:38:86",
                "all": false
        }
}
```

Example Response

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "status"
        },
        "reporter": {
                "name": "515-2",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1630347095"
        },
        "status": {
                "deviceMac": "e4f2057ee868",
                "status": "connectionUpdate",
                "statusString": "MTU Value Updated",
                "connUpdate": {
                        "mtu_value": 247
                }
        }
}

{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "actionResults"
        },
        "reporter": {
                "name": "515-2",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1630347095"
        },
        "results": [{
                "actionId": "10000001",
                "type": "bleConnect",
                "deviceMac": "e4f2057ee868",
                "status": "success",
                "statusString": "Connection Successful!"
        }]
}
```

Service and Characteristic Discovery

After connecting to a BLE device, the bleConnect command also triggers a full GATT service and characteristic discovery. This will happen without user intervention. Once this action is completed, the full list of characteristics will be forwarded back to the partner application in the “characteristics” northbound topic. This is the indication that the full service and characteristic discovery has been completed. An example of this response is provided below.

Example Response

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "characteristics"
        },
        "reporter": {
                "name": "515-2",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1630347097"
        },
        "characteristics": [{
                        "deviceMac": "e4f2057ee868",
                        "serviceUuid": "1800",
                        "characteristicUuid": "2a00",
                        "properties": [
                                "read"
                        ]
                },
                {
                        "deviceMac": "e4f2057ee868",
                        "serviceUuid": "1800",
                        "characteristicUuid": "2a01",
                        "properties": [
                                "read"
                        ]
                },
                {
                        "deviceMac": "e4f2057ee868",
                        "serviceUuid": "1801",
                        "characteristicUuid": "2a05",
                        "properties": [
                                "indicate"
                        ]
                },
                {
                        "deviceMac": "e4f2057ee868",
                        "serviceUuid": "19b10000e8f2537e4f6cd104768a1214",
                        "characteristicUuid": "19b10001e8f2537e4f6cd104768a1214",
                        "properties": [
                                "read",
                                "writeWithResponse"
                        ]
                }
        ]
}
```

ii. bleDisconnect

The bleDisconnect command will terminate the connection between the specified AP, and a specific remote BLE device.

Request Required Parameters	• ActionId • Type • Device MAC
Request Optional Parameters	• Timeout
Possible Responses	• success • apNotFound • deviceNotFound • actionTimeout • notConnected

Example Request

```
{
        "meta": {
                "access_token": "0123456789",
                "version": 1,
                "sbTopic": "actions"
        },
        "actions": [{
                "deviceMac": "e4:f2:05:7e:e8:68",
                "serviceUuid": null,
                "value": null,
                "characteristicUuid": null,
                "actionId": "10000010",
                "timeOut": 20,
                "type": "bleDisconnect"
        }],
        "receiver": {
                "apMac": "90:4c:81:cf:38:86",
                "all": false
        }
}
```

Example Response

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "actionResults"
        },
        "reporter": {
                "name": "515-2",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1630347097"
        },
        "results": [{
                "actionId": "10000010",
                "type": "bleDisconnect",
                "deviceMac": "e4f2057ee868",
                "status": "success",
                "statusString": "Disconnect Successful!"
        }]
}
```

iii. gattRead

The gattRead command must only be called after a connection between an AP and remote BLE device has been established. This command will read the value of the specified GATT characteristic on the remote device.

The response to gattRead differs slightly from the previous commands, as it can come in two different forms. In case of a successful operation, the 3rd party server will receive a response with the northbound topic (nbTopic) set to “characteristics”. The message will contain a “results” topic containing the actionId value among others. It will also contain the status enum and a status string that says “gattRead Successful!”. The actual value of the gattRead command will be contained in the “characteristics” topic, alongwith the device MAC, service UUID and characteristic UUID.

If an error occurred during a gattRead operation, the 3rd party server will receive a response with the northbound topic (nbTopic) set to “actionResults”. The response will have a “results” topic with the status enum for the gattRead failure.

Request Required Parameters	• ActionId • Type • Device MAC • Service UUID • Characteristic UUID
Request Optional Parameters	• Timeout
Possible Responses	• success • apNotFound • deviceNotFound • actionTimeout • notConnected • characteristicNotFound

Example Request

```
{
        "meta": {
                "access_token": "0123456789",
                "version": 1,
                "sbTopic": "actions"
        },
        "actions": [{
                "deviceMac": "e4:f2:05:7e:e8:68",
                "serviceUuid": "1800",
                "value": null,
                "characteristicUuid": "2a00",
                "actionId": "10000012",
                "timeOut": 20,
                "type": "gattRead"
        }],
        "receiver": {
                "apMac": "90:4c:81:cf:38:86",
                "all": false
        }
}
```

gattRead Responses

gattRead Response #1: Success

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "characteristics"
        },
        "reporter": {
                "name": "515-2",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1634687322"
        },
        "results": [{
                "actionId": "10000012",
                "type": "gattRead",
                "deviceMac": "e4f2057ee868",
                "status": "success",
                "statusString": "gattRead Successful!"
        }],
        "characteristics": [{
                "deviceMac": "e4f2057ee868",
                "serviceUuid": "1800",
                "characteristicUuid": "2a00",
                "value": "4152554e"
        }]
}
```

gattRead Response #2: Failure

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "actionResults"
        },
        "reporter": {
                "name": "i515",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1634777487"
        },
        "results": [{
                "actionId": "10000012",
                "type": "gattRead",
                "deviceMac": "e4f2057ee868",
                "status": "notConnected"
        }]
}
```

gattRead Response #3: Failure

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "actionResults"
        },
        "reporter": {
                "name": "i515",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1634796273"
        },
        "results": [{
                "actionId": "00000014",
                "type": "gattRead",
                "deviceMac": "e4f2057ee868",
                "status": "characteristicNotFound"
        }]
}
```

iv. gattWrite

The gattWrite command must only be called after a connection between an AP and remote BLE device has been established. This command will perform a GATT write ***without*** response to the specified characteristic, with the specified value. It is the 3rd party server's responsibility to know the capabilities of the characteristic beforehand or use the results of the service-characteristic discovery (after the bleConnect step) to determine that information and set the southbound message payload appropriately.

After the timeout specified in the southbound message expires, the results of the GATT write without response transaction are sent to the server with the "actionResults" northbound message topic (nbTopic). The message contains a "results" topic with the status set to "actionTimeout" and the statusString set to "gattwrite successful".

Request Required Parameters	• ActionId • Type • Device MAC • Service UUID • Characteristic UUID • Value
Request Optional Parameters	• Timeout
Possible Responses	• actionTimeout • apNotFound • deviceNotFound • notConnected • characteristicNotFound

Example Request

```
{
        "meta": {
                "access_token": "0123456789",
                "version": 1,
                "sbTopic": "actions"
        },
        "actions": [{
                "deviceMac": "e4:f2:05:7e:e8:68",
                "serviceUuid": "19b10000e8f2537e4f6cd104768a1214",
                "value": "0A",
                "characteristicUuid": "2103",
                "actionId": "20000012",
                "timeOut": 20,
                "type": "gattWrite"
        }],
        "receiver": {
                "apMac": "90:4c:81:cf:38:86",
                "all": false
        }
}
```

Example Response

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "actionResults"
        },
        "reporter": {
                "name": "i515",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1634798868"
        },
        "results": [{
                "actionId": "20000012",
                "type": "gattWrite",
                "deviceMac": "e4f2057ee868",
                "status": "actionTimeout",
                "statusString": "gattwrite successful"
        }]
}
```

v. gattWriteWithResponse

The gattWriteWithResponse command must only be called after a connection between an AP and remote BLE device has been established. This command will perform a GATT write ***with*** response to the specified characteristic, with the specified value. It is the 3rd party server's responsibility to know the capabilities of the characteristic beforehand or use the results of the service-characteristic discovery (after the bleConnect step) to determine that information and set the southbound message payload appropriately.

Unlike the gattWrite command, the gattWriteWithResponse command will send a message with the "actionResults" northbound topic to the server on success after a response packet is received from the connected device. The message contains a "results" topic with the status set to "success" and the statusString set to "WriteWithResponse Successful!".

Request Required Parameters	- ActionId - Type - Device MAC - Service UUID - Characteristic UUID - Value
Request Optional Parameters	- Timeout
Possible Responses	- success - actionTimeout - apNotFound - deviceNotFound - notConnected - characteristicNotFound

Example Request

```
{
        "meta": {
                "access_token": "0123456789",
                "version": 1,
                "sbTopic": "actions"
        },
        "actions": [{
                "deviceMac": "e4:f2:05:7e:e8:68",
                "serviceUuid": "19b10000e8f2537e4f6cd104768a1214",
                "value": "AA",
                "characteristicUuid": "2103",
                "actionId": "20000012",
                "timeOut": 5,
                "type": "gattWriteWithResponse"
        }],
        "receiver": {
                "apMac": "90:4c:81:cf:38:86",
                "all": false
        }
}
```

Example Response

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "actionResults"
        },
        "reporter": {
                "name": "i515",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1634800495"
        },
        "results": [{
                "actionId": "20000012",
                "type": "gattWriteWithResponse",
                "deviceMac": "e4f2057ee868",
                "status": "success",
                "statusString": "WriteWithResponse Successful!"
        }]
}
```

vi. gattNotification

The gattNotification command must only be called after a connection between an AP and remote BLE device has been established. This command will attempt to subscribe or unsubscribe to notifications for the specified characteristic. To subscribe to notifications, you must send a value of “1”, and to unsubscribe from notifications, you must send a value of “0”.

Request Required Parameters	• ActionId • Type • Device MAC • Service UUID • Characteristic UUID • Value
Request Optional Parameters	• Timeout
Possible Responses	• success • notConnected • characteristicNotFound • apNotFound • deviceNotFound

Example Request

```
{
        "meta": {
                "access_token": null,
                "version": 1,
                "sbTopic": "actions"
        },
        "actions": [{
                "deviceMac": "e4:f2:05:7e:e8:68",
                "serviceUuid": "fa01",
                "value": "01",
                "characteristicUuid": "2103",
                "actionId": "00000002",
                "timeOut": 20,
                "type": "gattNotification"
        }],
        "receiver": {
                "apMac": "90:4c:81:cf:38:86",
                "all": false
        }
}
```

gattNotification Responses

For gattNotification actions, the response structure differs from previous actions. When you subscribe or unsubscribe to notifications on a GATT characteristic, if the AP was successfully able to perform the operation, you will get a success status returned in the “actionResults” northbound message topic. Following a successful subscription to notifications, any notifications on the GATT characteristics that the 3rd party server has subscribed to will be forwarded back asynchronously. An example of a successful subscription, and an example of a notification value being forwarded are shown as follows:

gattNotification Response – Successful subscription

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "actionResults"
        },
        "reporter": {
                "name": "i515",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1634801697"
        },
        "results": [{
                "actionId": "00000002",
                "type": "gattNotification",
                "deviceMac": "e4f2057ee868",
                "status": "success",
                "statusString": "Notifications will now be forwarded"
        }]
}
```

gattNotification Response – Forwarded Notification Value

```
{
        "meta": {
                "version": "1",
                "access_token": "0123456789",
                "nbTopic": "characteristics"
        },
        "reporter": {
                "name": "i515",
                "mac": "904c81cf3886",
                "ipv4": "192.168.8.122",
                "hwType": "AP-515",
                "swVersion": "8.9.0.0-8.9.0.0",
                "swBuild": "81161",
                "time": "1634801712"
        },
        "results": [{
                "actionId": "00000002",
                "type": "gattNotification",
                "deviceMac": "e4f2057ee868",
                "status": "success",
                "statusString": "gattNotification value update received"
        }],
        "characteristics": [{
                "deviceMac": "e4f2057ee868",
                "serviceUuid": "fa01",
                "characteristicUuid": "2103",
                "value": "07"
        }]
}
```

vii. gattIndication

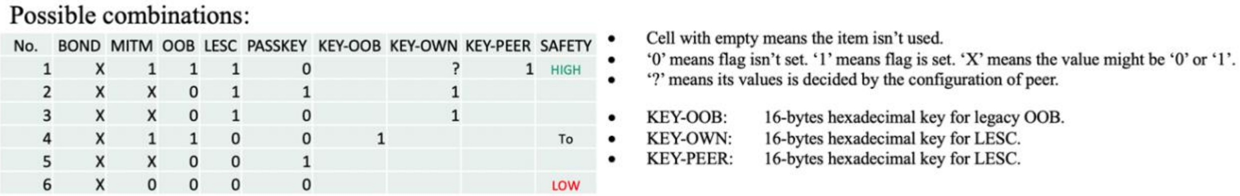
The gattIndication actions are very similar to gattNotification actions. All the actions and responses are the same, just the action type will be “gattIndication” as opposed to “gattNotification”. The same subscription response messages are sent to the third-party server. The only difference between the two, is gattNotification and gattIndication work different at the BLE level, and some peripheral BLE devices require gattIndication.

viii. bleAuthenticate

After connection is established, authentication might be required because of security considerations. The table below shows the possible authentication combinations supported by our infrastructure (AuthenticationMethod):

Method	Description
none	Legacy mode. No MITM is enabled
passkey	Legacy mode. Passkey is used. MITM is enabled.
oob	Legacy mode. KeyOob is required. MITM is enabled
lescNone	LESC mode. No MITM is enabled
lescPasskey	LESC mode. Passkey is used. MITM is enabled
lescOob	LESC mode. KeyOob is used. MITM is enabled

Below figure shows the details of different combinations:



Possible combinations:

No.	BOND	MITM	OOB	LESC	PASSKEY	KEY-OOB	KEY-OWN	KEY-PEER	SAFETY
1	X	1	1	1	0		?	1	HIGH
2	X	X	0	1	1		1		
3	X	X	0	1	0		1		
4	X	1	1	0	0	1			To
5	X	X	0	0	1				
6	X	0	0	0	0				LOW

- Cell with empty means the item isn't used.
- '0' means flag isn't set. '1' means flag is set. 'X' means the value might be '0' or '1'.
- '?' means its values is decided by the configuration of peer.

- KEY-OOB: 16-bytes hexadecimal key for legacy OOB.
- KEY-OWN: 16-bytes hexadecimal key for LESC.
- KEY-PEER: 16-bytes hexadecimal key for LESC.

The message 'Authentication' is used to construct the information used for authentication:

Field	Type	Description
method	AuthenticationMethod	This is a required field. Specify which method is used for authentication.
bonding	bool	Whether bonding is enabled, which also could depend on whether peer's bond is enabled.
passkey	string	Whether passkey is used. Passkey is 6 numeric digits ('0' - '9'). '0' will be prefixed if length of passkey is less than 6. Required when method is 'passkey' or 'lescPasskey'.
keyOob	bytes	16-bytes hexadecimal key. 0 will be prefixed if its length is less than 16. This is required when method is 'oob' or 'lescOob'.
keyOwn	bytes	16-bytes hexadecimal key. 0 will be prefixed if its length is less than 16. This is required when lesc is used, and peer uses the server's keyOwn or passkey is used

If bonding is enabled, a bonding key will be returned once authentication succeeds.

Sample lescOob Request

```
{
  "meta": {
    "access_token": null,
    "version": 1,
    "sbTopic": "actions"
  },
  "actions": [
    {
      "deviceMac": "c3bb362273bb",
      "serviceUuid": null,
      "value": null,
      "characteristicUuid": null,
      "actionId": "1111",
      "timeOut": 60,
      "type": "bleConnect"
    },
    {
      "serviceUuid": null,
      "value": null,
      "characteristicUuid": null,
      "authentication": {
        "bonding": true,
        "method": "lescOob",
        "passkey": "123456",
        "keyOwn":"1234567890ABCDEF",
        "keyOob":"FEDCBA0987654321"
      },
      "actionId": "2222",
      "timeOut": 60,
      "deviceMac": "c3bb362273bb",
      "type": "bleAuthenticate"
    }
  ],
  "receiver": {
    "apMac": "204c03c1a519",
    "all": false
  }
}
```

Sample Response corresponding to Request

```
{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "actionResults"
  },
  "reporter": {
    "name": "AP555",
    "mac": "9c8cd8cf2d77",
    "ipv4": "192.16.1.5",
    "ipv6": "fe80::9e8c:d8ff:fecf:2d77",
    "hwType": "AP-555",
    "swVersion": "8.8.0.0-mm-dev",
    "swBuild": "77282",
    "time": "1601480094"
  },
  "results": [
    {
      "actionId": "2222",
      "type": "bleAuthenticate",
      "deviceMac": "c3bb362273bb",
      "status": "success",
      "bondingKey": {
        "key":
"ce07afb0b8a0269e9e9f30fd7ba2518ba9f493f184c4398ae486996859ef47440181bbb83c09bb3142da4d70a6ca3c1d04d1e4
f223686de5f1942809192b6d692670b5672f2adebf4bd3e20ee69b438b"
      }
    }
  ]
}
```

ix. bleEncrypt

After authentication is successful, the link will be encrypted. As authentication can take time, we provide a method to encrypt link quickly without performing full authentication. The bonding key from the last successful authentication is cached. We can encrypt the link directly with this bonding key. The bonding key will be changed after each successful authentication.

Sample SB Request

```
{
  "meta": {
    "access_token": null,
    "version": 1,
    "sbTopic": "actions"
  },
  "actions": [
    {
      "deviceMac": "c3bb362273bb",
      "serviceUuid": null,
      "value": null,
      "characteristicUuid": null,
      "actionId": "1111",
      "timeOut": 60,
      "type": "bleConnect"
    },
    {
      "serviceUuid": null,
      "value": null,
      "characteristicUuid": null,
      "bondingKey": {
        "key":
"b59ef03bcc80b3a20b8e1ab7768b5dfa03b987679c927c4c1ed90632f90c24b5279d08d081e1c349c99d5257bff624a9e48d91
23f8d685616ea5ccf247d6145208dbc4fcef4e495cc8a8bfc952235ef6"
      },
      "actionId": "2222",
      "timeOut": 60,
      "deviceMac": "c3bb362273bb",
      "type": "bleEncrypt"
    }
],
  "receiver": {
    "apMac": "204c03c1a519",
    "all": false
  }
}
```

Sample NB Response

```
{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "actionResults"
  },
  "reporter": {
    "name": "AP515",
    "mac": "904c81cf378c",
    "ipv4": "192.16.1.6",
    "ipv6": "fe80::924c:81ff:fecf:378c",
    "hwType": "AP-515",
    "swVersion": "8.8.0.0-mm-dev",
    "swBuild": "77185",
    "time": "1600684063"
  },
  "results": [
    {
      "actionId": "2222",
      "type": "bleEncrypt",
      "deviceMac": "c3bb362273bb",
      "status": "success"
    }
  ]
}
```

8. Wi-Fi telemetry

The Wi-Fi telemetry service sends periodic reports about all the Wi-Fi devices that are discovered by an AP. The AP sees over the air wireless frames from devices that are in the vicinity of the AP. The AP classifies these devices into (a) associated stations: devices for which we observe bi-directional frames, i.e., going from AP to station and from station to AP, and (b) unassociated stations: devices for which we observe frames either going to the devices or from the device to its associated AP. At every reporting interval, in the periodic report for each station, we will send the tuple of station MAC address, received signal strength (RSSI), and device class.

To enable the WiFi telemetry service in the IoT transport profile, the user will need to include the wifi-assoc-sta and wifi-unassoc-sta classes in the device class filter. The configured server will receive one or more Google Protocol buffer encoded messages, depending upon the number of observed stations, at every reporting interval.

Note: WiFi telemetry is only available when the server type is set to Telemetry-Websocket.

Sample Message

```
{
   "meta": {
      "version": "1",
      "access_token": "any",
      "nbTopic": "telemetry"
   },
   "reporter": {
      "name": "Aruba_AP1",
      "mac": "004e35c76a08",
      "ipv4": "10.5.0.120",
      "ipv6": "fe80::24e:35ff:fec7:6a08",
      "hwType": "AP-365",
      "swVersion": "8.6.0.4",
      "swBuild": "74969",
      "time": "1587663421"
   },
   "wifiData": [
      {
         "mac": "9a5da1e7a59d",
         "deviceClass": [
            "wifiUnassocSta"
         ],
         "rssi": -87
      },
      {
         "mac": "205415caed1a",
         "deviceClass": [
            "wifiAssocSta"
         ],
         "rssi": -75
      }
   ]
}
```

9. Wi-Fi RTLS data

The WiFi RTLS data telemetry service forwards the wireless data frames that originate from unassociated Wi-Fi tags to the configured server. Wireless packets from unassociated Wi-Fi tags are distinguished from other frames based on the wireless packet type, and the values of the toDS and fromDS flags in the frame control field. When the incoming packet is a data frame with either toDS = 1 and fromDS = 1, or toDS = 0 and fromDS = 0, then the AP tries to match the MAC address from the Address 1 field to the destination MAC address configured in the transport profile. If it is a match then the AP generates a report with the device MAC address, received signal strength (RSSI), device class (set to “wifiTag”) and the payload of the wireless frame.

To enable the WiFi RTLS data telemetry service in the IoT transport profile, the user will need to include the wifi-tags class in the device class filter. Whenever the AP sees a frame that matches the MAC address configured in the rtlsDestMAC field in the IoT transport profile, it will immediately send a report to the configured server as a Google Protocol buffer encoded message.

Note: WiFi telemetry is only available when the server type is set to Telemetry-Websocket.

Example Message

```
{
   "meta": {
      "version": "1",
      "access_token": "test",
      "nbTopic": "telemetry"
   },
   "reporter": {
      "name": "Aruba_AP1",
      "mac": "004e35c76a08",
      "ipv4": "10.5.0.120",
      "ipv6": "fe80::24e:35ff:fec7:6a08",
      "hwType": "AP-365",
      "swVersion": "8.6.0.4",
      "swBuild": "74969",
      "time": "1590518273"
   },
   "wifiData": [
      {
         "mac": "000ccc48c5a1",
         "deviceClass": [
            "wifiTag"
         ],
         "rssi": -51,
         "rtls_payload": "00130b060200020033020722bc5a000006770407000ccc00001200"
      }
   ]
}
```

10. Zigbee Sockets Data

Aruba defines a new concept for Zigbee data communication called Zigbee Socket Device (ZSD), which can simplify the usage for sending/receiving data over Zigbee. ZSD specifies two parts:

- inbound sockets: Inbound socket is used for receiving data from peer device.
- outbound sockets: Outbound socket is used for sending data to peer device.

Socket consists of 4 members: source-endpoint, destination-endpoint, profile ID and cluster ID. These four parameters are defined into a message 'ZbE2PC' (e2pc) in the API. When a ZSD is bound to an ATW transport profile by configuration, all data related to the e2pc can be transmitted over the ZSD. In fact, e2pc specifies a data tunnel between server and clients. Different services have different e2pc. Sometimes, e2pc for sending is also different from the one for receiving. Similarly, e2pc is like the port of TCP/UDP. Different ports can indicate different services. In the Zigbee world, each connected device has a short network address which is allocated by coordinator/router. This short network address can be treated as the IP address. In Aruba implementation, we use the IEEE address of client device to send data, which is more generic. In the Zigbee stack, we will convert the IEEE address to short address if we have it.

The Northbound message from the inbound socket contains the following fields:

Request Required Parameters	• radio_mac IEEE MAC of radio where data is received from
Request Optional Parameters	• report Send data to server. • ack This is used for acknowledgement for the SbZbMsg when 'reqid' is specified. The ack includes 'result' (SUCCEEDED, FAILED) and 'code' which gives the exact failure reason. • response This is used for the 'read' request from SbZbMsg.
Possible Responses	So far, we have no acknowledgement or response for the 'report'.

Sample Northbound Message:

```
{
    "meta":{
        "version":1,
        "nbTopic":"zbNbData"
    },
    "reporter":{
        "mac":"80:8d:b7:c0:0d:95"                    //AP MAC
    },
    "zigbee":{
        "radioMac":"20:4c:03:ff:fe:13:8c:84",        //AP Zigbee radio MAC
        "report":{
            "mac":"00:13:a2:00:41:58:3a:7c",         //EndDevice MAC
            "e2pc":{
                "destination":{
                    "endpoint":2,
                    "profileId":28674,
                    "clusterId":64514
                },
                "sourceEndpoint":52
            },
            "payload":"000F000A001122334455"
        }
    }
}
```

The Southbound message destined for the outbound socket contains the following fields:

Request Required Parameters	• radio_mac IEEE MAC of radio used for sending data
Request Optional Parameters	• send Send data to peer device. It includes 4 required parameters: reqid, mac, e2pc and payload. • request Not implemented in AOS. it includes these operations: ‘read’, ‘write’, ‘action’(for other actions).
Possible Responses	• SUCCEEDED • FAILED

Sample Southbound Message:

```
{
    "meta":{
        "version":1,
        "sbTopic":3
    },
    "receiver":{
        "apMac":"80:8d:b7:c0:0d:95"                   //AP MAC
    },
    "zigbee":{
        "radioMac":"20:4c:03:ff:fe:13:8c:84",         //AP Zigbee radio MAC
        "send":{
            "mac":"00:13:a2:00:41:58:3a:ce",          //EndDevice MAC
            "e2pc":{
                "destination":{
                    "endpoint":101,
                    "clusterId":28673,
                    "profileId":61441
                },
                "sourceEndpoint":151
            },
            "payload":"7001000A001122334455",
            "reqid":"1"
        }
    }
}
```

11. Serial data

Starting AOS 8.7, Aruba APs support data forwarding service for 3rd party IoT radios that are connected to the AP via the USB port. Every 3rd party radio requires custom integration involving bundling the device driver, port configuration and message parsing subroutines into the AP's software image. When the 3rd party IoT radio is plugged into the USB port, it presents itself as a serial over USB device to the AP after the appropriate driver installation.

The serial data sent by the 3rd party radio to the AP is encoded into a Google Protocol Buffer formatted message and forwarded to the server configured in the IoT transport profile. The server can also send a Google Protocol Buffer formatted message to the AP (Southbound), which will be forwarded to the 3rd party device, i.e., the serial data bytes will be written to the serial port. The serial data forwarding service is only available when the server type is Telemetry-WebSocket. In every SB communication the server needs to populate the correct USB serial device identifier to ensure that the message is forwarded to the correct device. The device identifier is part of each AP Health Update message and NB serial data messages. In addition, the device identifier is included in the output of the “ble-config” CLI command.

Example Northbound Message

```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "serialDataNb"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.8.0.0-8.8.0.0",
    "swBuild": "79680",
    "time": "1616700759"
  },
  "nbSData": [
    {
      "nbSerialPayload": "55000807013dd006490412f2948001ffffffff4d009d",
      "nbDeviceId": "ENOCEAN_USB:f6a68e740ecc549496d4b63072a33920"
    }
  ]
}
```

Example Southbound Message

```
{
    "meta": {
        "access_token": "0123456789",
        "version": 1,
        "sbTopic": "serialDataSb"
    },
    "sbSData": [{
        "sbDeviceId": "ENOCEAN_USB:f6a68e740ecc549496d4b63072a33920",
        "sbSerialPayload": "5500010005700309"
    }],
    "receiver": {
        "apMac": "904c81cf3886",
        "all": false
    }
}
```

Example Response to Southbound Message

```
{
  "meta": {
    "version": "1",
    "access_token": "0123456789",
    "nbTopic": "serialDataNb"
  },
  "reporter": {
    "name": "515-2",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.122",
    "hwType": "AP-515",
    "swVersion": "8.8.0.0-8.8.0.0",
    "swBuild": "79680",
    "time": "1616700509"
  },
  "nbSData": [
    {
      "nbSerialPayload":
"55002100022600010208000103040004129f1e454f010354434d35313555000000000000000000039",
      "nbDeviceId": "ENOCEAN_USB:f6a68e740ecc549496d4b63072a33920"
    }
  ]
}
```

12. Appendix

a. BLE Connections Example using external radio

As mentioned in Section 7: BLE Connections, starting AOS 8.10, the AP-USB-ZB USB device can be used to connect and interact with BLE devices remotely via the SB IoT interface. In the following example, we demonstrate the different NB and SB messages that capture the interactions between the AP's' external radio and the remote BLE device.

Example CLI output for "show ap debug ble-config"

```
i515# show ap debug ble-config

============================================================
                     IOT Radio Profiles
============================================================
Profile Name              : int
Radio Instance            : Internal
Radio Mode                : ZigBee
ZigBee Mode               : Coordinator
ZigBee Channel            : auto
BLE/ZigBee Tx Power (dBm) : 4
-----------------------------------------------
Profile Name              : ext
Radio Instance            : External
Radio Mode                : BLE
BLE Mode                  : beaconing scanning
BLE Console               : Off
BLE/ZigBee Tx Power (dBm) : 0
-----------------------------------------------
...
...

============================================================
                      Attached Radios
============================================================
Radio Configuration
-------------------
Item                                    Value
----                                    -----
Radio Information                       NORDIC ONBOARD Internal BLE Zigbee
Radio Profile Type                      1
Zigbee Supported                        Yes
APB MAC Address                         20:4c:03:39:e2:2c
APB Radio Profile                       0x0002
APB Radio BLE Configured TxPower        0dBm
APB Radio BLE Operational TxPower       0dBm
...
...
-----------------
Radio Configuration
-------------------
Item                                 Value
----                                 -----
Radio Information                    NORDIC USB External BLE Zigbee
Radio Profile Type                   1
Zigbee Supported                     No
APB MAC Address                      20:4c:03:a7:b7:5d
APB Radio Profile                    0x000d
...
```

Example AP Health Information Message

```
{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "apHealthUpdate"
  },
  "reporter": {
    "name": "i515",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.124",
    "hwType": "AP-515",
    "swVersion": "8.10.0.0-8.10.0.0",
    "swBuild": "akamthe_akamthe_sbx0_5_000000",
    "time": "1649133933"
  },
  "apHealth": {
    "apStatus": "healthy",
    "radio": [
      {
        "mac": "204c0339e22c",
        "hardware": "gen2",
        "firmware": "arubaDefault",
        "health": "healthy",
        "external": false
      },
      {
        "mac": "204c03a7b75d",
        "hardware": "gen2",
        "firmware": "arubaDefault",
        "health": "healthy",
        "external": true
      }
    ]
  }
}
```

Example Southbound API Actions Message

```
{
    "meta": {
       "access_token": null,
       "version": 1,
       "sbTopic": "actions"
    },
    "actions": [
       {
          "deviceMac": "e4:f2:05:7e:e8:68",
          "serviceUuid": null,
          "value": null,
          "characteristicUuid": null,
          "actionId": "00000001",
          "timeOut": 30,
          "type": "bleConnect"
       },
       {
         "deviceMac": "e4:f2:05:7e:e8:68",
         "serviceUuid": "fa01",
         "value": "01",
         "characteristicUuid": "2103",
         "actionId": "00000002",
         "timeOut": 20,
         "type": "gattNotification"
       },
       {
         "deviceMac": "e4:f2:05:7e:e8:68",
         "serviceUuid": "fa01",
         "value": null,
         "characteristicUuid": "2103",
         "actionId": "00000003",
         "timeOut": 20,
         "type": "gattRead"
     }
    ],
    "receiver": {
       "apMac": "90:4c:81:cf:38:86",
       "all": false
    }
 }
```

Example Southbound API Response Messages

```
 {
   "meta": {
     "version": "1",
     "access_token": "1234567890",
     "nbTopic": "status"
   },
   "reporter": {
     "name": "i515",
     "mac": "904c81cf3886",
     "ipv4": "192.168.8.124",
     "hwType": "AP-515",
     "swVersion": "8.10.0.0-8.10.0.0",
     "swBuild": "akamthe_akamthe_sbx0_5_000000",
     "time": "1649133997"
   },
   "status": {
     "deviceMac": "e4f2057ee868",
     "status": "connectionUpdate",
     "statusString": "MTU Value Updated",
     "connUpdate": {
       "mtu_value": 247
     }
   }
 }

 {
   "meta": {
     "version": "1",
     "access_token": "1234567890",
     "nbTopic": "actionResults"
   },
   "reporter": {
     "name": "i515",
     "mac": "904c81cf3886",
     "ipv4": "192.168.8.124",
     "hwType": "AP-515",
     "swVersion": "8.10.0.0-8.10.0.0",
     "swBuild": "akamthe_akamthe_sbx0_5_000000",
     "time": "1649133997"
   },
   "results": [
     {
       "actionId": "00000001",
       "type": "bleConnect",
       "deviceMac": "e4f2057ee868",
       "status": "success",
       "statusString": "Connection Successful!",
       "apbMac": "204c03a7b75d"
     }
   ]
 }
```

```
{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "characteristics"
  },
  "reporter": {
    "name": "i515",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.124",
    "hwType": "AP-515",
    "swVersion": "8.10.0.0-8.10.0.0",
    "swBuild": "akamthe_akamthe_sbx0_5_000000",
    "time": "1649133998"
  },
  "characteristics": [
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "1800",
      "characteristicUuid": "2a00",
      "properties": [
        "read"
      ]
    },
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "1800",
      "characteristicUuid": "2a01",
      "properties": [
        "read"
      ]
    },
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "1801",
      "characteristicUuid": "2a05",
      "properties": [
        "indicate"
      ]
    },
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "fa01",
      "characteristicUuid": "2101",
      "properties": [
        "read",
        "notify"
      ]
    },
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "fa01",
      "characteristicUuid": "2102",
      "properties": [
        "read",
        "writeWithResponse",
        "notify"
      ]
    },
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "fa01",
      "characteristicUuid": "2103",
      "properties": [
        "read",
        "writeWithoutResponse",
        "writeWithResponse",
        "notify"
      ]
    }
  ]
}

{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "actionResults"
  },
  "reporter": {
    "name": "i515",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.124",
    "hwType": "AP-515",
    "swVersion": "8.10.0.0-8.10.0.0",
    "swBuild": "akamthe_akamthe_sbx0_5_000000",
    "time": "1649133999"
  },
  "results": [
    {
      "actionId": "00000002",
      "type": "gattNotification",
      "deviceMac": "e4f2057ee868",
      "status": "success",
      "statusString": "Notifications will now be forwarded",
      "apbMac": "204c03a7b75d"
    }
  ]
}

{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "characteristics"
  },
  "reporter": {
    "name": "i515",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.124",
    "hwType": "AP-515",
    "swVersion": "8.10.0.0-8.10.0.0",
    "swBuild": "akamthe_akamthe_sbx0_5_000000",
    "time": "1649133999"
  },
  "results": [
    {
      "actionId": "00000003",
      "type": "gattRead",
      "deviceMac": "e4f2057ee868",
      "status": "success",
      "statusString": "gattRead Successful!",
      "apbMac": "204c03a7b75d"
    }
  ],
  "characteristics": [
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "fa01",
      "characteristicUuid": "2103",
      "value": "3b"
    }
  ]
```

```
}

{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "characteristics"
  },
  "reporter": {
    "name": "i515",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.124",
    "hwType": "AP-515",
    "swVersion": "8.10.0.0-8.10.0.0",
    "swBuild": "akamthe_akamthe_sbx0_5_000000",
    "time": "1649134008"
  },
  "results": [
    {
      "actionId": "00000002",
      "type": "gattNotification",
      "deviceMac": "e4f2057ee868",
      "status": "success",
      "statusString": "gattNotification value update received",
      "apbMac": "204c03a7b75d"
    }
  ],
  "characteristics": [
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "fa01",
      "characteristicUuid": "2103",
      "value": "3c"
    }
  ]
}

{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "characteristics"
  },
  "reporter": {
    "name": "i515",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.124",
    "hwType": "AP-515",
    "swVersion": "8.10.0.0-8.10.0.0",
    "swBuild": "akamthe_akamthe_sbx0_5_000000",
    "time": "1649134019"
  },
  "results": [
    {
      "actionId": "00000002",
      "type": "gattNotification",
      "deviceMac": "e4f2057ee868",
      "status": "success",
      "statusString": "gattNotification value update received",
      "apbMac": "204c03a7b75d"
    }
  ],
  "characteristics": [
    {
      "deviceMac": "e4f2057ee868",
      "serviceUuid": "fa01",
      "characteristicUuid": "2103",
      "value": "3d"
    }
  ]
}

{
  "meta": {
    "version": "1",
    "access_token": "1234567890",
    "nbTopic": "actionResults"
  },
  "reporter": {
    "name": "i515",
    "mac": "904c81cf3886",
    "ipv4": "192.168.8.124",
    "hwType": "AP-515",
    "swVersion": "8.10.0.0-8.10.0.0",
    "swBuild": "akamthe_akamthe_sbx0_5_000000",
    "time": "1649134026"
  },
  "results": [
    {
      "actionId": "10000111",
      "type": "bleDisconnect",
      "deviceMac": "e4f2057ee868",
      "status": "success",
      "statusString": "Disconnect Successful!",
      "apbMac": "204c03a7b75d"
    }
  ]
}
```