Cisco Sensor Connect for IoT Services Programmability Guide

First Published: 2024-08-27

Last Modified: 2025-04-01

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

Preface

This preface describes the conventions of this document and information on how to obtain other documentation. It also provides information on what's new in Cisco product documentation.

Document Conventions

This document uses the following conventions:

Convention	Description
^ or Ctrl	Both the ^ symbol and Ctrl represent the Control (Ctrl) key on a keyboard. For example, the key combination ^**D** or **Ctrl-D** means that you hold down the Control key while you press the D key. (Keys are indicated in capital letters but are not case sensitive.)
bold font	Commands and keywords and user-entered text appear in **bold** font.
Italic font	Document titles, new or emphasized terms, and arguments for which you supply values are in *italic* font.
[x]	Elements in square brackets are optional.

Reader Alert Conventions

This document may use the following conventions for reader alerts:

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Related Documentation

- *Cisco Sensor Connect for IoT Services Configuration Guide*

- *Cisco Sensor Connect for IoT Services Online Help* (Refer **Initial Configuration Workflow of IoT Orchestrator** section)

Communications, services, and additional information

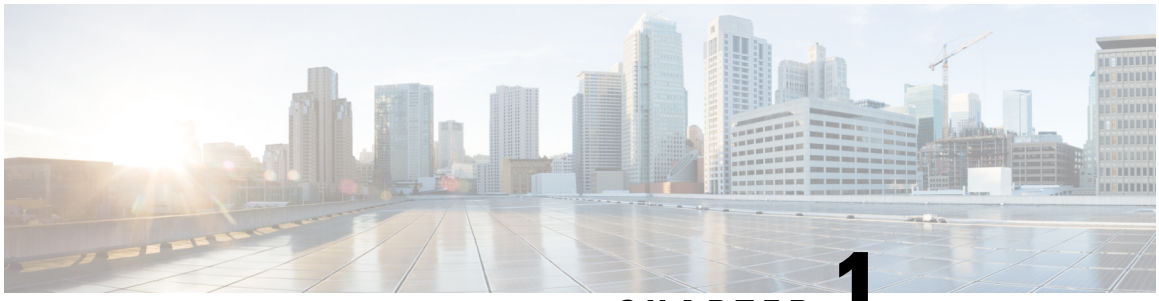
- To receive timely, relevant information from Cisco, sign up at Cisco Profile Manager.
- To get the business impact you're looking for with the technologies that matter, visit Cisco Services.
- To submit a service request, visit Cisco Support.
- To discover and browse secure, validated enterprise-class apps, products, solutions, and services, visit Cisco DevNet.
- To obtain general networking, training, and certification titles, visit Cisco Press.
- To find warranty information for a specific product or product family, access Cisco Warranty Finder.

Cisco Bug Search Tool

Cisco Bug Search Tool (BST) is a gateway to the Cisco bug-tracking system, which maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. The BST provides you with detailed defect information about your products and software.

Documentation feedback

To provide feedback about Cisco technical documentation, use the feedback form available in the right pane of every online document.



CHAPTER 1

Overview

Overview of Cisco Sensor Connect for IoT Services

Enterprise customers need easy ways to onboard, authorize, and control IoT devices onto a single unified infrastructure. The Cisco Sensor Connect addresses this by providing a set of interfaces to:

- Onboard devices into the infrastructure.
- Control and receive events from the devices.

To accomplish this, the Cisco Sensor Connect leverages the following interfaces between the network and applications:

1. **Onboarding Interface**: This interface is used to onboard a device to a network.
2. **Control Interface**: This interface is used for bi-directional communication with a non-IP device.
3. **Telemetry Interface**: This interface is used for streaming telemetry from a non-IP device.

Introduction to Cisco Sensor Connect APIs

The IoT Orchestrator is a Cisco IOx container that is an integral part of the Cisco Sensor Connect solution. You can deploy the IoT Orchestrator on the Cisco Catalyst 9800 Wireless Controllers.

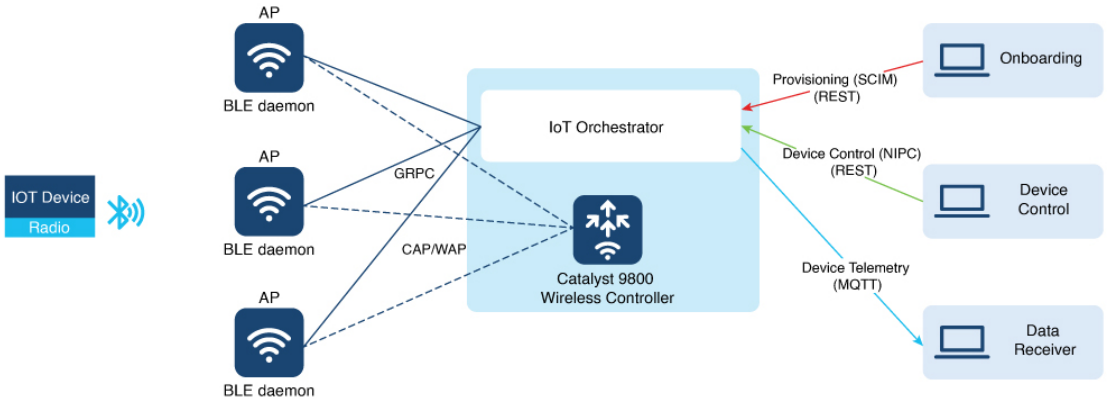
The IoT Orchestrator exposes APIs that allow applications to:

- Introduce devices to enterprise networks
- Control devices

- Receive telemetry from those devices

Note Device control and data reception are not required for IP-based end points because the devices can communicate directly using their IP address.

Figure 1: Cisco Sensor Connect APIs



The IoT Orchestrator exposes the following APIs:

- **Provisioning Interface:**
  - Leverages the IETF System for Cross-Domain Identity Management (SCIM version 2) API, specifically the SCIM for devices model.

    For information on the Device Schema Extensions to the SCIM model, see https://datatracker.ietf.org/doc/draft-ietf-scim-device-model/.

Note The SCIM has a hierarchical schema with a core device schema and extensions.

The SCIM allows an application to:

- Send a SCIM object to a SCIM server (gateway) to create, update, and delete devices in networks.

  For information on the System for Cross-domain Identity Management (SCIM) API, see RFC7643 and RFC7644.

- **Control Interface:**
  - Allows an application to connect to a non-IP device.
  - Enables data exchange with the device.
  - Register topics for streaming telemetry. The IETF draft for this protocol is called the Non-IP Control (NIPC).

- **Telemetry Interface:**

- A streaming data interface with publisher sub-topics of different types, such as:
  - Device broadcasts
  - Device streaming data, or
  - Device connection state.

API Security

The API security in the IoT Orchestrator is configured using either:

- API Key
- Certificate-Based Authentication

IoT Orchestrator in Cisco Catalyst 9800 Wireless Controller

The Cisco Catalyst 9800 Wireless Controller and APs support interfaces as mentioned in the Introduction to Cisco Sensor Connect APIs for BLE devices.

The IoT Orchestrator has two internal components:

- Gateway
- BLE Controller

These components are collocated in a single containerized application.

The IoT Orchestrator works in conjunction with a BLE daemon in the Cisco Catalyst APs. The controller unclears the AP infrastructure to the application and ensures that the device is connected to the most appropriate AP. This means that the application can communicate with a single Gateway and does not have to manage the network topology.

Gateway

The gateway supports the following functionalities:

- **Onboarding (using SCIM) Interfaces:** The gateway acts as the SCIM server.
- **Control (using NIPC) Interfaces:** The gateway provides NIPC interfaces as RESTful interfaces (Gateway is the server).
- **Telemetry (using MQTT) Interfaces:** The gateway functions as MQTT (Gateway is the MQTT broker).

Solution Overview

To communicate with or receive data from a BLE device, an application needs to first onboard the device by creating a SCIM object and send it to the SCIM server (the gateway). The SCIM object uses the device schema and has a BLE extension. If the SCIM call is successful, the gateway generates a unique Device ID and returns the complete SCIM object as registered with the Device ID.

Subsequently, the application uses the Device ID and leverages the control interfaces to communicate with the device or execute a specific NIPC operation.

The NIPC API covers the following types of APIs:

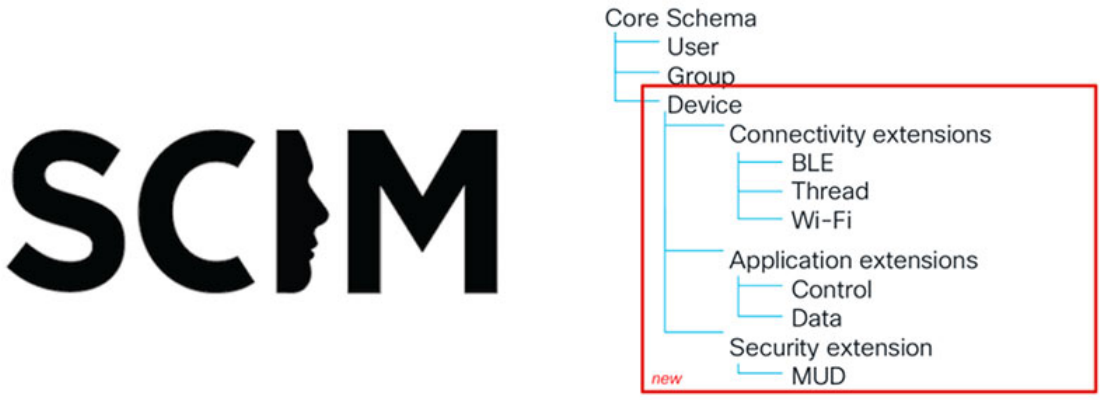
- **Connectivity**: Connect or disconnect, and pair BLE operations.
- **Data**: Read or write, start notifications or indications, discover.
- **Registrations**: Register data application, register topic (either advertisement or connection state).
- **Bulk**: API that allows for bulk operations (multiple operations in one API call).

About System for Cross-Domain Identity Management (SCIM)

The System for Cross-Domain Identity Management (SCIM) is an open standard designed to manage user identity information.

SCIM provides a defined schema to represent users and groups, and a RESTful API to run create, read, update, delete (CRUD) operations on the user and group resources.

Figure 2: SCIM



For more information, see SCIM.

About Non-IP Control (NIPC)

The use cases in building management, healthcare, workplaces, manufacturing, logistics, and hospitality have introduced low-power devices into these environments. These devices typically do not support IP-based interface. Hence, there is a need for gateway functions to allow these devices to communicate with the applications that manage them.

The control interface addresses the device and group objects as ID. Therefore, it is essential to declare a device to the gateway before addressing a NIPC operation in a device. You can address the NIPC operation in a device using SCIM.

Note

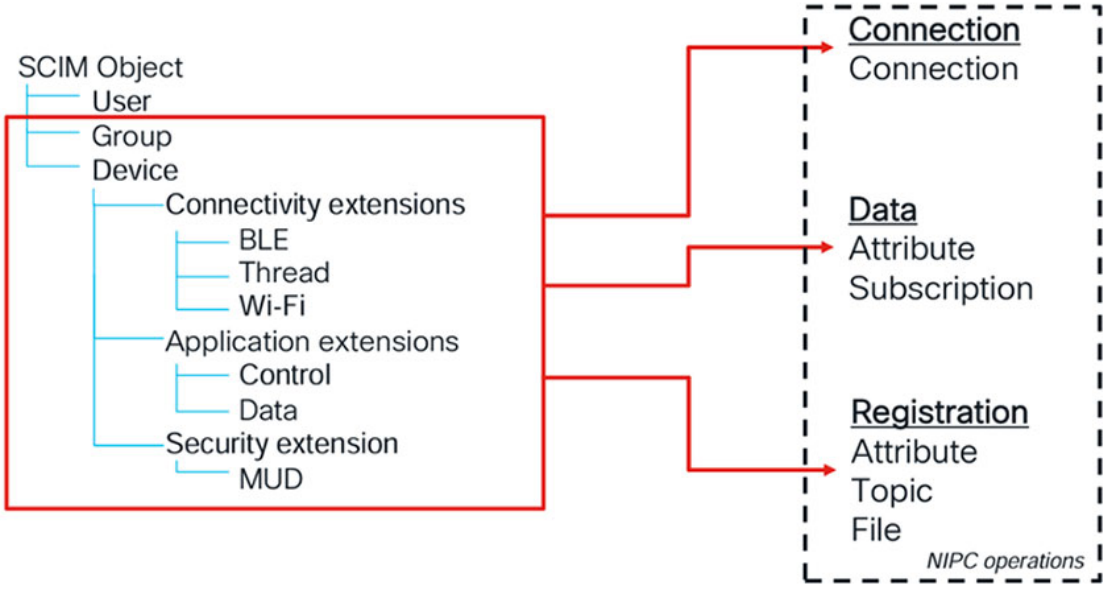
- The NIPC operation can be performed using a device or group ID.
- The existing NIPC APIs were developed while the standard was still evolving, resulting in some aspects being implemented differently. These APIs will be updated to align with the latest draft version of NIPC. Please note that backward compatibility with APIs from version 1.x will not be supported.

The gateway leverages information from the SCIM object to execute a specific NIPC operation. For instance, the keying information in SCIM object may be required to connect to a device.

Advantages of using NIPC

- Enables bi-directional communication with non-IP devices.
- Allows an application to register publisher or subscriber topics to support the data streaming interface.

Figure 3: Non-IP Control (NIPC)



For more information, see NIPC.

Note SCIM and NIPC APIs have a 75-second timeout.

Each BLE device supports only one pending API request. Issuing another API for a BLE device before completion causes the request to fail.

About Message Queuing Telemetry Transport (MQTT)

The Message Queuing Telemetry Transport (MQTT) is a messaging protocol used to establish communication between multiple devices. This protocol relies on the publish-subscribe model widely used for communication in IoT framework.

Note The MQTT broker is Mosquitto, and the version is 2.0.20. The MQTT message format version is v3.1.

Publish-Subscribe Model

This model separates the client that sends messages (publisher) from the client that receives messages (subscriber). Publisher and subscribers do not need to establish a direct connection and MQTT broker is responsible to route and distribute all messages.

For example, consider the temperature sensor. It connects to the MQTT server as a client and publishes the temperature data to a topic (Temperature), and the server receives the message and forwards it to the client subscribed to the Temperature topic.

Topic

The MQTT protocol route messages based on topic. For information on the MQTT topics, see MQTT Topics and Wildcards: A Beginner's Guide.

About MQTT Message Batching

To improve network efficiency and achieve high MQTT throughput in scale scenarios, the IoT Orchestrator application supports MQTT message batching. As a result, a new protobuf message called **DataBatch** has been introduced. It acts as a wrapper for multiple data subscription messages, and each DataBatch message can contain multiple DataSubscription messages.

Note The partner application that consumes MQTT messages from the IoT Orchestrator must update the data application protocol buffer from the Pre-GA Release prototype to the GA Release prototype.

You need a GitHub account to access the following links:

- For information on the Data Application Protocol Buffer (Pre-GA Release), see the https://github.com/ietf-wg-asdf/asdf-nipc/blob/nipc-asdf-01/proto/data_app.proto.
- For information on the Data Application Protocol Buffer (GA Release), see the https://github.com/ietf-wg-asdf/asdf-nipc/blob/cisco-iot-orchestrator-1.1/proto/data_app.proto.

About Shared Subscriptions

A Shared Subscription is a feature in MQTT that allows multiple clients (subscribers) to share the workload of receiving messages from a single topic. Instead of each subscriber receiving all published messages (as in a normal subscription), MQTT brokers distribute messages among the subscribers in a round-robin or load-balanced manner. This strategy distributes messages among multiple subscribers, preventing any single

subscriber client from being overwhelmed. It also helps achieve high MQTT throughput by preventing a single subscriber client from becoming a bottleneck.

Note The NIPC API for registering the topic, registering the data application, and subscribing to the topic remains unchanged from the Pre-GA version of the IoT Orchestrator. No changes are required for the partner application. The expected MQTT subscriber throughput, based on the 9800 platform on which the IoT Orchestrator application is deployed, is presented in the following table:

Table 1: Platforms and MQTT Subscribers (Recommended)

Platforms	MQTT Subscribers (Recommended)
Cisco Catalyst 9800-L Wireless Controller	2
Cisco Catalyst 9800-40 Wireless Controller	5
Cisco Catalyst 9800-80 Wireless Controller	8
Cisco Catalyst CW9800M Wireless Controller	5
Cisco Catalyst CW9800H1 and CW9800H2 Wireless Controllers	8

About Onboarding BLE Devices

The Internet of Things present a management challenge in many dimensions. One of them being the ability to onboard and manage large number of devices. There are many models to bootstrap trust between devices and network deployments.

The System for Cross Identity Management (SCIM) defines a protocol and a schema to provision users. However, the SCIM is extended to provision devices. The protocol and core schema are designed to allow such extensions.

Why SCIM for devices

The SCIM considers only the information necessary to bootstrap trust so that the device can establish connectivity.

What's Next

For information about the API definition, see **Onboarding BLE Devices using SCIM**.

About Control Operations on BLE Devices

There is a need for non-IP devices to support processes in manufacturing, healthcare, retail, home, and office. Similarly, wireless access points are deployed nearly everywhere, many of which have radios that can transmit and receive different frame types (such as BLE). To integrate both these use cases to leverage a single wireless infrastructure and avoid the need of parallel infrastructure, you need a non-IP device gateway function.

The non-IP device gateway provides the following functions:

- Authenticate and authorize application clients to communicate with devices.
- APIs that allow an application to set up a connection with a device.
- APIs that allow an application to exchange data with a device.
- APIs that allow a device to create registrations in the network.

All these APIs along with the onboarding API (SCIM for devices) allows application to perform a complete set of operations in non-IP devices.

The following are the supported API tags:

Table 2: Supported API Tags

API Tags	Description
connectivity	APIs that allow applications to manage device connections.
data	APIs that allow applications to exchange data with non-IP devices.
registrations	APIs that allow applications to make registrations in the network for devices.
bulk	Compound API that allows applications to combine requests into a single call.

What's Next

For information about the API definition, see **Control Operations on BLE Devices**.

About Data Telemetry from BLE Devices

The NIPC is extensible in two ways:

- **Protocol Extensions**: This allows for extensions to the schema to integrate new non-IP communication protocols.
- **Interface Extensions**: This allows for defining extensions, such as publish/subscribe interface or data streaming interface.

Publish or Subscribe Interface

The publish or subscribe interface or data streaming interface is an MQTT publishing interface. Pub or sub topics can be created and managed by means of the **/register/topic** NIPC element.

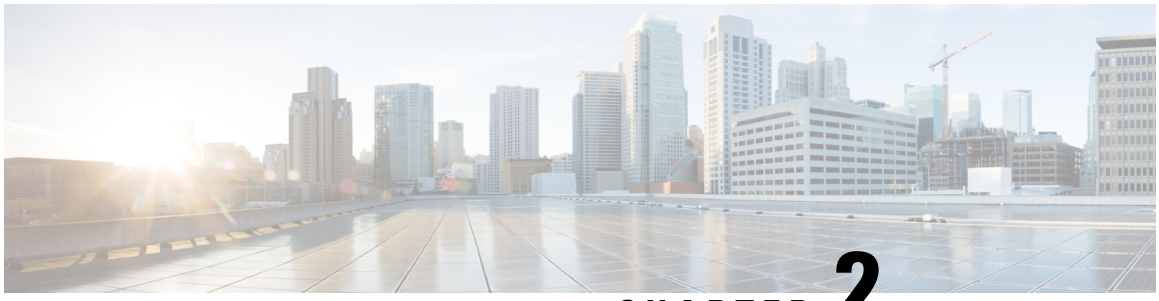
What's Next

For information about the API definition, see **Data Telemetry from BLE Devices**.

About Security Model

All RESTful and MQTT interfaces in the Gateway are TLS secured. You will need to configure a server certificate in the Gateway.

Client authentication can be done either using an API key or a certificate. For more information, see the **Uploading Certificate and Key to Open HTTP Server and Listen for APIs** section in the *Cisco Sensor Connect for IoT Services Configuration Guide*.



CHAPTER 2

Onboarding BLE Devices using SCIM

SCIM API Definition

API for Onboarding Device	Description
`POST /scim/v2/Devices`	This is the API used for onboarding a BLE device using SCIM.
`GET /scim/v2/Devices/{id}?onboardApp={onboardAppID}`	
`DELETE /scim/v2/Devices/{id}?onboardApp={onboardAppID}`	

SCIM HTTP Methods

The SCIM protocol specifies end points and HTTP methods to manage resources.

The following are the supported HTTP methods:

HTTP Method	SCIM Usage
GET	Retrieve a resource.
POST	Create a new resource.
DELETE	Deletes a resource.

SCIM API Resource and Versions

The resource and endpoint supported in the SCIM APIs are **Device** and **/Devices** respectively.

The URL path covers the base URL and version identifier as a segment. The SCIM APIs support **v2** as the version identifier.

/scim/v2/Devices are the SCIM API resource and version supported for HTTP POST, GET, or DELETE.

SCIM Schema

SCIM Core Device Schema

The SCIM core device schema supports the following attributes:

Attributes	Description	Sample

schemas	The schemas section of the SCIM APIs cover the list of schemas part of the API requests.
	The following are the supported schemas:
	• Core device schema
	• BLE device extension schema
	• Endpoint application extension schema
		`"schemas": [`
		`"urn:ietf:params:scim:schemas:core:2.0:Device",`
		`"urn:ietf:params:scim:schemas:extension:ble:2.0:Device",`
		`"urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"`
		`]`

deviceDisplayName	The attribute is of "string" type and provides a human-readable name for a device.	`"deviceDisplayName": "BLE Heart Monitor",`
adminState	The attribute is of "Boolean" type and is a mutable attribute. If **adminState** is set to TRUE, connect, disconnect, and subscribe commands that control app sends to the controller for the devices will be processed by the application. If **adminState** is set to FALSE, any command coming from the control app for the device will be rejected by the application.	`"adminState": true,`

BLE Device Extension Schema

The SCIM device extension schema supports the following attributes:

Attributes	Description	Sample
DeviceMacAddress	• Is a string value that represents a public MAC address assigned by the manufacturer. • It is a unique 48-bit value. • It is required, case-sensitive, mutable, and returned as default. • The following is the regex pattern: `^[0-9A-Fa-f]{2}(:[0-9A-Fa-f]{2}){5}$`	`"deviceMacAddress": "CA:2B:5C:EC:95:46",`
isRandom	• Is a Boolean flag. • If set to **FALSE**, the device uses a public MAC address. • If set to **TRUE**, the device uses a static random MAC address. • This attribute is not required, mutable, and returned by default. • The default value is **FALSE**.	`"isRandom": false,`
versionSupport	• Provides all the BLE versions supported by the device in the form of an array. • For example, [4.1, 4.2, 5.0, 5.1, 5.2,5.3]. • This attribute is required, mutable, and returned as default.	`"versionSupport": [ "5.3" ],`
pairingMethods	• Is an array of pairing methods associated with the BLE device. • May require sub-attributes, such as key or password for the device pairing process. • This attribute is required, case-sensitive, mutable, and returned by default.	`"pairingMethods": ["urn:ietf:params:scim:schemas:extension:pairing:1.0:Device", "urn:ietf:params:scim:schemas:extension:pairing:JustWorks:2.0:Device"], "urn:ietf:params:scim:schemas:extension:pairing:1.0:Device": null, "urn:ietf:params:scim:schemas:extension:pairing:JustWorks:2.0:Device": { "key": null }`

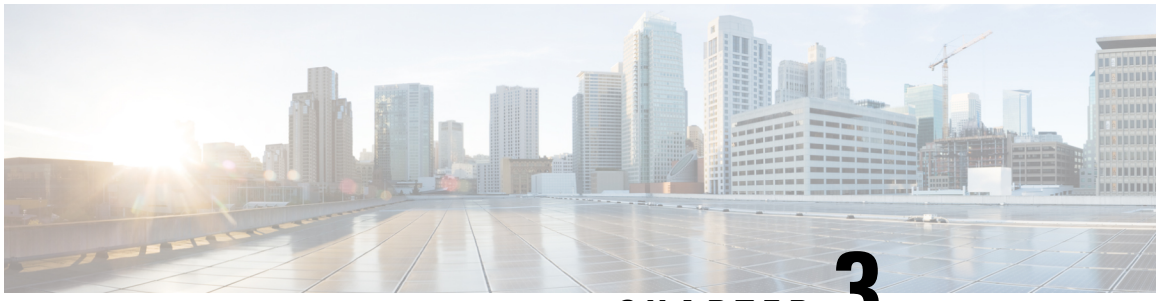
Attributes	Description	Sample
mobility	• Is a Boolean flag. • When set to **FALSE**, this feature disables seamless movement between APs, requiring manual intervention for the IoT Orchestrator to handle connections and disconnections. • When set to **TRUE**, this feature allows devices to move between APs seamlessly, with the IoT Orchestrator automatically handling connections and disconnections without manual intervention. • This attribute is not required, mutable, and returned by default. • The default value is FALSE.	`"mobility": true,`

API Example

The following example lists the SCIM object for onboarding the BLE device:

```
{
  "schemas": [
    "urn:ietf:params:scim:schemas:core:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
  ],
  "deviceDisplayName": "BLE Heart Monitor",
  "adminState": true,
  "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
    "versionSupport": [
      "5.3"
    ],
    "deviceMacAddress": "CA:2B:5C:EC:95:46",
    "isRandom": false,
    "mobility": false,
    "pairingMethods": [
      "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device",
      "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device"
    ],
    "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": null,
    "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
      "key": null
    }
  },
```

```
  "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
    "onboardingUrl": "onboardApplication",
    "deviceControlUrl": [
      "controlApplication"
    ],
    "dataReceiverUrl": []
  }
}
```



CHAPTER 3

Control Operations on BLE Devices

Non-IP Control (NIPC) Open API Definition

API	Description
`POST /control/connectivity/connect`	This API is used to connect a BLE device to the network.
`POST /control/data/read`	This API is used to read a value from an attribute in the BLE device.
`POST /control/data/write`	This API is used to write a value to an attribute in the BLE device.
`POST /control/data/discover`	This API is used to discover services in a BLE device.
`POST /control/connectivity/disconnect`	This API is used to disconnect a connected BLE device from the network.

Other Supported API Types

API	Path	Summary
Register Data App	`/registration/registerDataApp`	Register a data application.
Unregister Data App	`/registration/unregisterDataApp`	Unregister a data application.

API	Path	Summary
Register Topic	`/registration/registerTopic`	Register a publish or subscribe topic.
Unregister Topic	`/registration/unregisterTopic`	Unregister a publish or subscribe topic.
Subscribe Topic	`/data/subscribe`	Subscribe to the streaming data from an attribute in the device.
Unsubscribe Topic	`/data/unsubscribe`	Unsubscribe to the streaming data from an attribute in the device.

Connecting to a BLE Device

Connect Request - API Definition

Fields	Description
`location 'https://10.195.78.33:8081/control/connectivity/connect'`	Connect a BLE device to the network.
`header 'x-api-key: 2215dec3718288ad9028d39864f87d4c'`	API key
`header 'Content-Type: application/json'`	Content type is JSON.
`data`	
`technology`	BLE
`id`	Device ID
`serviceID`	Service ID of the BLE device.
`"controlApp": "controlApplication"`	Control application address

Connect Response - API Definition

Fields	Description
status	Status message
id	Device ID
requestID	Request ID
serviceID	Service ID
handle	Handle number

Fields	Description
characteristicID	Characteristic ID
descriptorID	Descriptor ID
flags	Flags

API Example – Connect Request and Connect Response

Connect Request:

```
curl -k --location 'https: //10.195.78.33:8081/control/connectivity/connect' \
--header 'x-api-key: 2215dec3718288ad9028d39864f87d4c' \
--header 'Content-Type: application/json' \
--data '
{
"technology": "ble",
"id": "f837c89a-e1dd-4b90-a446-06016c0d2b75",
"ble": {
"services": [
{
"serviceID": "1800"
},
{
"serviceID": "2022"
},
{
"serviceID": "1822"
},
{
"serviceID": "180F"
}
]
},
"controlApp": "controlApplication"
}
'
```

Connect Response:

```
{
"status": "SUCCESS",
"id": "f837c89a-e1dd-4b90-a446-06016c0d2b75",
"requestID": "75310f89-79c6-4e7b-93cb-f58a8f18fc57",
"services": [
{
"serviceID": "1800",
"characteristics": [
{
"characteristicID": "3443",
"descriptors": [
{
"descriptorID": "9059"
}
],
"flags": [
"read",
```

```
"write",
"notify"
]
}
]
},
{
"serviceID": "2022",
"handle": 1,
"characteristics": [
{
"characteristicID": "8065",
"descriptors": [
{
"descriptorID": "9620"
}
],
"flags": [
"read",
"write",
"notify"
]
}
]
},
{
"serviceID": "1822",
"handle": 2,
"characteristics": [
{
"characteristicID": "8912",
"descriptors": [
{
"descriptorID": "1554"
}
],
"flags": [
"read",
"write",
"notify"
]
}
]
},
{
"serviceID": "180F",
"handle": 3,
"characteristics": [
{
"characteristicID": "2810",
"descriptors": [
{
"descriptorID": "1517"
}
],
"flags": [
"read",
"write",
"notify"
]
}
]
}
```

```
]
}
```

Reading from a BLE Device

Read Request - API Definition

Fields	Description
`location 'https: //10.195.78.33:8081/control/data/read'`	Read a value from an attribute in the BLE device.
`header 'x-api-key: 2215dec3718288ad9028d39864f87d4c'`	API key
`header 'Content-Type: application/json'`	Content type is JSON.
`technology`	BLE
`id`	Device ID
`serviceID`	Service ID of the BLE device.
`characteristicID`	Characteristic ID of the BLE device.
`"controlApp": "controlApplication"`	Control application address.

Read Response - API Definition

Fields	Description
status	Status message
id	Device ID
requestID	Request ID
value	Value

API Example – Read Request and Read Response

Read Request:

```
curl -k --location 'https: //10.195.78.33:8081/control/data/read' \
--header 'x-api-key: 2215dec3718288ad9028d39864f87d4c' \
--header 'Content-Type: application/json' \
--data '
{
"technology": "ble",
"id": "decf0dc6-9e9d-4d39-8c8b-00c541699b6a",
```

```
"ble": {
"serviceID": "1800",
"characteristicID": "2A00"
},
"controlApp": "controlApplication"
}
'
```

Read Response:

```
{
"status": "SUCCESS",
"id": " decf0dc6-9e9d-4d39-8c8b-00c541699b6a",
"requestID": "8ba846aa-6133-4392-b911-9850a90791f2",
"value": "2342432"
}
```

Writing to a BLE Device

Write Request - API Definition

Fields	Description
`location 'https: //10.195.78.33:8081/control/data/write'`	Write a value to an attribute in the BLE device.
`header 'x-api-key: 2215dec3718288ad9028d39864f87d4c'`	API key
`header 'Content-Type: application/json'`	Content type is JSON.
`technology`	BLE
`id`	Device ID
`value`	Value
`serviceID`	Service ID of the BLE device.
`characteristicID`	Characteristic ID of the BLE device.
`"controlApp": "controlApplication"`	Control application address.

Write Response – API Definition

Fields	Description
status	Status message
id	Device ID
requestID	Request ID

Fields	Description
value	Value

Discovering Services from a BLE Device

Service Discovery Request - API Definition

Fields	Description
`location 'https://10.195.78.33:8081/control/data/discover'`	Discover services in a BLE device.
`header 'x-api-key: 2215dec3718288ad9028d39864f87d4c'`	API key
`header 'Content-Type: application/json'`	Content type is JSON.
`technology`	BLE
`id`	Device ID
`serviceID`	Service ID of the BLE device.
`"controlApp": "controlApplication"`	Control application address

Service Discovery Response - API Definition

Fields	Description
status	Status message
id	Device ID
requestID	Request ID
serviceID	Service ID
handle	Handle
characteristicID	Characteristic ID
descriptorID	Descriptor ID
Flags	Flags: read, write, notify

API Example - Service Discovery Request and Service Discovery Response

Service Discovery Request:

```
curl -k --location 'https: //10.195.78.33:8081/control/data/discover' \
--header 'x-api-key: 2215dec3718288ad9028d39864f87d4c' \
--header 'Content-Type: application/json' \
--data '
{
"technology": "ble",
"id": "f837c89a-e1dd-4b90-a446-06016c0d2b75",
"ble": {
"services": [
{
"serviceID": "1800"
},
{
"serviceID": "2022"
},
{
"serviceID": "1822"
},
{
"serviceID": "180F"
}
]
},
"controlApp": "controlApplication"
}
'
```

Service Discovery Response:

```
{
"status": "SUCCESS",
"id": "f837c89a-e1dd-4b90-a446-06016c0d2b75",
"requestID": "d4a225b1-fc62-478e-ad54-928c8973351d",
"services": [
{
"serviceID": "1800",
"characteristics": [
{
"characteristicID": "1797",
"descriptors": [
{
"descriptorID": "7062"
}
],
"flags": [
"read",
"write",
"notify"
]
}
]
},
{
"serviceID": "2022",
"handle": 1,
"characteristics": [
```

```
{
"characteristicID": "4654",
"descriptors": [
{
"descriptorID": "8815"
}
],
"flags": [
"read",
"write",
"notify"
]
}
]
},
{
"serviceID": "1822",
"handle": 2,
"characteristics": [
{
"characteristicID": "7126",
"descriptors": [
{
"descriptorID": "4823"
}
],
"flags": [
"read",
"write",
"notify"
]
}
]
},
{
"serviceID": "180F",
"handle": 3,
"characteristics": [
{
"characteristicID": "4675",
"descriptors": [
{
"descriptorID": "9631"
}
],
"flags": [
"read",
"write",
"notify"
]
}
]
}
]
}
```

Disconnecting from a BLE Device

Disconnect Request - API Definition

Fields	Description
`location 'https://10.195.78.33:8081/control/connectivity/disconnect'`	Disconnect a connected BLE device from the network.
`header 'x-api-key: 2215dec3718288ad9028d39864f87d4c'`	API key
`header 'Content-Type: application/json'`	Content type is JSON.
`data`	
`technology`	BLE
`id`	Device ID
`"controlApp": "controlApplication"`	Control application address

Disconnect Response - API Definition

Fields	Description
status	Status message
id	Device ID
requestID	Request ID

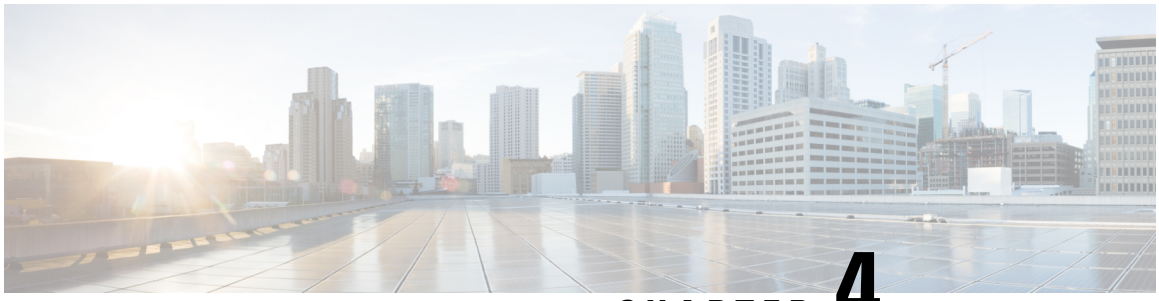
API Example – Disconnect from a BLE Device

Disconnect Request:

```
curl -k --location 'https://10.195.78.33:8081/control/connectivity/disconnect'
\
--header 'x-api-key: 2215dec3718288ad9028d39864f87d4c' \
--header 'Content-Type: application/json' \
--data '
{
"technology": "ble",
"id" : "decf0dc6-9e9d-4d39-8c8b-00c541699b6a",
"controlApp" : "controlApplication"
}
'
```

Disconnect Response:


```
{
"status": "SUCCESS",
"id": "decf0dc6-9e9d-4d39-8c8b-00c541699b6a",
"requestID": "cb838a8d-775b-4e71-874d-1aa22e61f9af"
}
```



CHAPTER 4

Data Telemetry from BLE Devices

Registering Topics

Register Topics - API Definition

API	Description
`/control/registration/registerTopic`	This API registers the topic with the gateway to subscribe to GATT notifications from device using ID, service UUID, and characteristic UUID. The subscriptions can be placed to various GATT attributes: • Blood Oxygen level • Heart rate • Walking rate

API Example - Register Topics

Register Topics Request:

```
{
"technology": "ble",
"ids": [
"df553860-975c-4d7f-8c1f-b2996f34e26f",
],
"controlApp": "controlApplication",
"topic": "enterprise/hospital/pulse_oximeter",
"dataFormat": "default",
"ble": {
```

```
"type": "gatt",
"serviceID": "1800",
"characteristicID": "2A5E"
}
}
```

Register Topics Response:

```
{
    "status": "SUCCESS",
    "topic": "enterprise/hospital/pulse_oximeter",
    "reason": "Gatt topics successfully registered!"
}
```

API Example - Subscribe to Critical Application Events

This subscription helps identify if the IoT Orchestrator has experienced a critical event (for example, upgrade process, uninstallation, or restart). In the current IoT Orchestrator implementation, only devices in the Onboarded state are persisted in the database. Devices in any state other than Onboarded will not be persisted; therefore, if a critical event occurs in the IoT Orchestrator, a new onboarding process is required. This API provides information about these critical events for a particular topic.

Critical Request Subscription Request:

```
{
    "technology": "ble",
    "controlApp": "controlApplication",
    "topic": "enterprise/hospital/pulse_oximeter",
    "ble": {
        "type": "application_events"
}
}
```

Critical Request Subscription Response:

```
{
  "status" : "SUCCESS",
  "topic" : "enterprise/hospital/pulse_oximeter",
  "reason" : "Application event topics successfully registered!"
}
```

Telemetry Data Format

For information, see https://github.com/iot-onboarding/non-ip-iot-control/blob/nipc-asdf-01/proto/data_app.proto.

Telemetry Message Format

The telemetry messages are in **protobuf** format. All the telemetry messages are encapsulated in the **DataSubscription** message.

Subscribing to Advertisements and Notifications

Subscribing to Advertisements and Notifications - API Definition

API	Description
/control/data/subscribe	Use this API to enable GATT notifications or indications in the BLE device using the device ID, GATT service UUID, and characteristic UUID.

API Example – Subscribing to Advertisements and Notifications

Request:

```
{
"technology": "ble",
"id": "df553860-975c-4d7f-8c1f-b2996f34e26f",
"ble": {
"serviceID": "1800",
"characteristicID": "2A5E"
},
"controlApp": "controlApplication"
}
```

Response:

```
{
"status": "SUCCESS",
"id": "df553860-975c-4d7f-8c1f-b2996f34e26f",
"requestID": "12345678-5678-1234-5578-abcdef1234"
}
```

MQTT Subscription Messages

Once the notification is enabled using the subscribe API, the data receiver application starts to receive the GATT notification. To receive the notification, you will need to execute the following command in your terminal session:

```
mosquitto_sub -h localhost -p 41883 -t
enterprise/hospital/pulse_oximeter -u
https://dataApplication --pw
c4e80e0483af0a4394dfb6e3ec5e820b
```

Telemetry Message Format for Onboarded Advertisements and Notifications

The following is the code snippet of Telemetry message format:

```
syntax = "proto3";
```

```
import "google/protobuf/timestamp.proto";

option java_package = "org.ietf.nipc.proto";
option java_multiple_files = true;

package nipc;

message DataSubscription {
    optional string device_id = 1;
    bytes data = 2;
    google.protobuf.Timestamp timestamp = 3;
    optional string ap_mac_address = 4;

    reserved 5 to 10;

    oneof subscription {
        BLESubscription ble_subscription = 11;
        BLEAdvertisement ble_advertisement = 12;
        ZigbeeSubscription zigbee_subscription = 13;
        RawPayload raw_payload = 14;
        BLEConnectionStatus ble_connection_status = 15;
        ApplicationEvent application_event = 16;
    }

    message BLESubscription {
        optional string service_uuid = 1;
        optional string characteristic_uuid = 2;
    }

    message BLEAdvertisement {
        string mac_address = 1;
        optional int32 rssi = 2;
    }

    message ZigbeeSubscription {
        optional int32 endpoint_id = 1;
        optional int32 cluster_id = 2;
        optional int32 attribute_id = 3;
        optional int32 attribute_type = 4;
    }

    message BLEConnectionStatus {
        string mac_address = 1;
        bool connected = 2;
        optional int32 reason = 3;
    }

    message RawPayload {
        optional string context_id = 1;
    }

    message ApplicationEvent {
        string message = 1;
    }
}

message DataBatch {
    repeated DataSubscription messages = 1;
}
```

Subscribing to Connection Events

Subscribing to Connection Events - API Definition

API	Description
`/control/registration/registerTopic`	Use this API to register a topic to provide notifications for device connection and disconnection events.

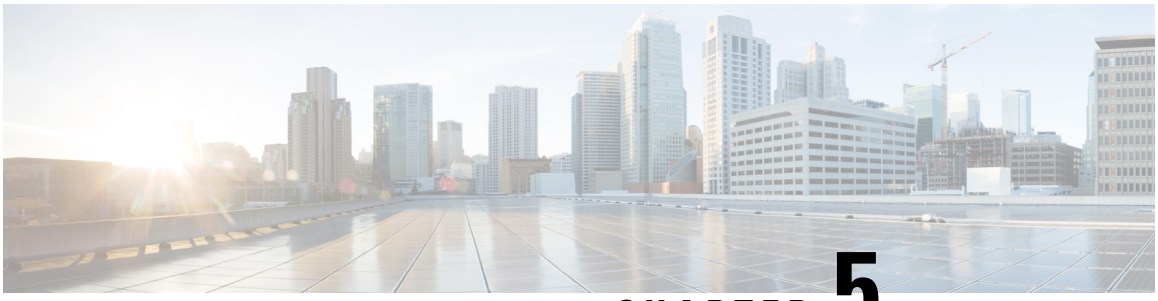
API Example – Subscribing to Connection Events

Request:

```
{
    "technology": "ble",
    "ids": [
        "df553860-975c-4d7f-8c1f-b2996f34e26f","
    ],
    "controlApp": "controlApplication",
    "topic": "enterprise/hospital/conn_events",
    "dataFormat": "default",
    "ble": {
        "type": "connection_events"
    }
}
```

Response:

```
{
  "status" : "SUCCESS",
  "topic" : "enterprise/hospital/conn_events",
  "reason" : "Successfully registered Connection event topics.",
  "registeredIDs" : [ "df553860-975c-4d7f-8c1f-b2996f34e26f" ]
}
```



CHAPTER 5

Use Case 1 - Asset Tracking

Use Case 1: Asset Tracking

If you have an asset to be tracked using a BLE tag, you need to:

- Onboard the BLE device.
- Register topics for receiving the onboarded device advertisements over MQTT using shared subscriptions.

Consider a large factory with many moving carts, each equipped with a BLE tag.

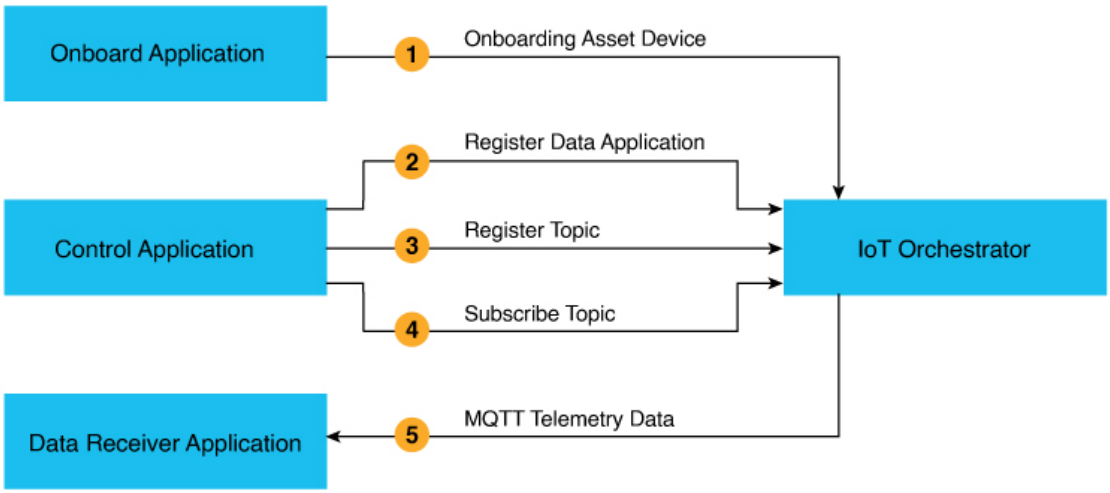
To track the location of a cart, you need to:

- Onboard the BLE device.
- Register topics for receiving the onboarded device advertisements over MQTT.

In this use case, the device to be tracked has the following details:

- Display Name: `Asset Tag`
- MAC Address Type: `Public`
- MAC Address: `EF:00:00:00:00:00`

For information on the BLE device tracking, refer to the following workflow:



The following workflow covers the sequence of operations:

1. Onboarding a Device.
2. Registering the Data Receiver Application.
3. Registering a Topic.
4. Onboarded Advertisements Subscription.

Onboarding a Device

Request Format:

```
curl -k --location 'https://173.39.84.53:8081/scim/v2/Devices' \
--header 'Content-Type: application/json' \
--header 'x-api-key: 596f0eba1668e6afeb3b25669990b7d84b8bf70408f21465a05e227664763024' \
--data '{
    "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
    ],
    "deviceDisplayName": "Asset tracking tag",
    "adminState": true,
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
        "versionSupport": [
            "5.3"
        ],
        "deviceMacAddress": "CA:2B:5C:EC:95:46",
        "isRandom": false,
        "mobility": false,
        "pairingMethods": [
            "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device",
            "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device"
        ],
        "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": null,
```

```
        "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
            "key": null
        }
    },
    "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
        "onboardingUrl": "onboardApplication",
        "deviceControlUrl": [
            "controlApplication"
        ],
        "dataReceiverUrl": []
    }
}'
```

Response Format:

```
{
  "schemas": [
    "urn:ietf:params:scim:schemas:core:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
  ],
  "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "deviceDisplayName": "Asset tracking tag",
  "adminState": true,
  "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
    "versionSupport": [
      "5.3"
    ],
    "deviceMacAddress": "CA:2B:5C:EC:95:46",
    "isRandom": false,
    "pairingMethods": [
      "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device",
      "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device"
    ],
    "mobility": false,
    "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": {},
    "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingPassKey:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingOOB:2.0:Device": {
      "key": "",
      "randNumber": 0,
      "confirmationNumber": 0
    }
  },
  "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
    "onboardingUrl": "onboardApplication",
    "deviceControlUrl": [
      "controlApplication"
    ],
    "dataReceiverUrl": []
  }
}
```

Registering the Data Receiver Application

You need to register a data receiver application with the app ID "dataApplication" to subscribe to the topic "enterprise/hospital/advertisements".

Request Format:

```
curl -k --location 'https://173.39.84.53:8081/control/registration/registerDataApp' \
--header 'Content-Type: application/json' \
--header 'x-api-key: b922958a7b030857b3e40f8d6a4db58f6a2bd65741f47fb120517ff8bb6fb0cb' \
--data '{
    "controlApp": "controlApplication",
    "topic": "enterprise/hospital/advertisements",
    "dataApps": [
        {
            "dataAppID": "dataApplication"
        }
    ]
}'
```

Response Format:

```
{
  "status" : "SUCCESS",
  "topic" : "enterprise/hospital/advertisements",
  "reason" : "Data applications successfully registered!"
}
```

Registering a Topic

You will need to register the topic "enterprise/hospital/advertisements" with the gateway to be able to subscribe to advertisements from the device with ID "79ca0f72-4ae2-436a-8bd4-24a2770faa83".

Request Format:

```
curl -k --location 'https://173.39.84.53:8081/control/registration/registerTopic' \
--header 'Content-Type: application/json' \
--header 'x-api-key: b922958a7b030857b3e40f8d6a4db58f6a2bd65741f47fb120517ff8bb6fb0cb' \
--data '{
    "technology": "ble",
    "topic": "enterprise/hospital/advertisements",
    "ids": [
        "79ca0f72-4ae2-436a-8bd4-24a2770faa83"
    ],
    "controlApp": "controlApplication",
    "ble": {
        "type": "advertisements"
    }
}'
```

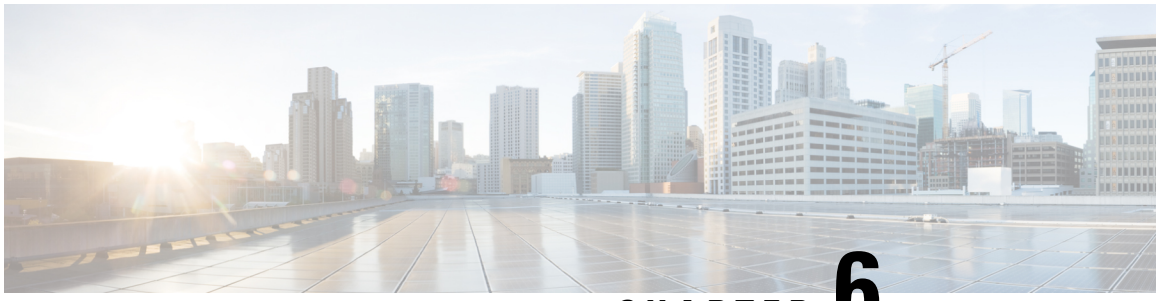
Response Format:

```
{
  "status" : "SUCCESS",
  "topic" : "enterprise/hospital/advertisements",
  "reason" : "Onboarded advertisement topics successfully registered!",
  "registeredIDs" : [ "79ca0f72-4ae2-436a-8bd4-24a2770faa83" ]
}
```

Onboarded Advertisements Subscription

```
mosquitto_sub -h localhost -p 41883 -t '$share/group/enterprise/hospital/advertisements'
-u
dataApplication --pw d5a4c7b1afd862a070514528006f22d4964f6c61ec0e2e0b6c3ebd03c2fbb507

$dbdb70f8-b02d-4733-9d1e-8875fa74bda8 dnaspaces.io/iot
"44:e8:80:00:00:00b
EF:00:00:00:00:02

$dbdb70f8-b02d-4733-9d1e-8875fa74bda8 dnaspaces.io/iot
"44:e8:80:00:00:00b
EF:00:00:00:00:02

$dbdb70f8-b02d-4733-9d1e-8875fa74bda8 dnaspaces.io/iot
"44:e8:80:00:00:00b
EF:00:00:00:00:02

$dbdb70f8-b02d-4733-9d1e-8875fa74bda8 dnaspaces.io/iot
"44:e8:80:00:00:00b
EF:00:00:00:00:02
```

Note The partner application must run multiple instances of the **mosquitto_sub** command. The number of instances is based on the Cisco Catalyst 9800 Wireless Controller platform on which the IoT Orchestrator is deployed. For more information, see the Table 1: Platforms and MQTT Subscribers (Recommended).



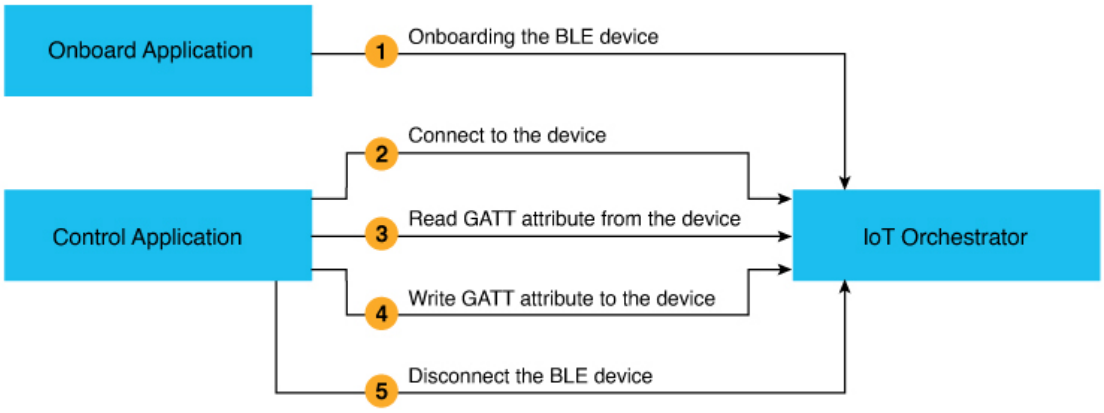
CHAPTER 6

Use Case 2 - Remote Patient Health Monitoring

Use Case 2 - Remote Patient Health Monitoring (requiring BLE connection, reading, and writing)

In this use case, you can:

1. Onboard a device.
2. Connect to the device.
3. Perform service discovery.
4. Write a characteristic.
5. Read the characteristic.
6. Disconnect from the device.



The following workflow covers the sequence of operations:

1. Onboarding a Device.
2. Connecting to a Device.
3. Writing a Characteristic to the Device.
4. Reading a Characteristic to the Device.
5. Disconnecting the Device.

Onboarding a Device

API Definition

API	Description
`POST /scim/v2/Devices`	This is the API for onboarding a device.

Request Format for Onboarding the Device without Pairing

```
curl -k --location 'https://173.39.84.53:8081/scim/v2/Devices' \
--header 'Content-Type: application/json' \
--header 'x-api-key: 596f0eba1668e6afeb3b25669990b7d84b8bf70408f21465a05e227664763024' \
--data '{
    "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
    ],
    "deviceDisplayName": "BLE Heart Monitor",
    "adminState": true,
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
        "versionSupport": [
            "5.3"
        ],
        "deviceMacAddress": "CA:2B:5C:EC:95:46",
        "isRandom": false,
```

```
        "mobility": false,
        "pairingMethods": [
            "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device"
        ],
        "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": null,
        "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
            "key": null
        }
    },
    "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
        "onboardingUrl": "onboardApplication",
        "deviceControlUrl": [
            "controlApplication"
        ],
        "dataReceiverUrl": []
    }
}'
```

Response Format for Onboarding the Device without Pairing

```
{
  "schemas": [
    "urn:ietf:params:scim:schemas:core:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
  ],
  "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "deviceDisplayName": "BLE Heart Monitor",
  "adminState": true,
  "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
    "versionSupport": [
      "5.3"
    ],
    "deviceMacAddress": "CA:2B:5C:EC:95:46",
    "isRandom": false,
    "pairingMethods": [
      "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device"
    ],
    "mobility": false,
    "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": {},
    "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingPassKey:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingOOB:2.0:Device": {
      "key": "",
      "randNumber": 0,
      "confirmationNumber": 0
    }
  },
  "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
    "onboardingUrl": "onboardApplication",
    "deviceControlUrl": [
      "controlApplication"
    ],
    "dataReceiverUrl": []
  }
}
```

Request Format for Onboarding the Device with Pairing

```
curl -k --location 'https://173.39.84.53:8081/scim/v2/Devices' \
--header 'Content-Type: application/json' \
--header 'x-api-key: 596f0eba1668e6afeb3b25669990b7d84b8bf70408f21465a05e227664763024' \
--data '{
    "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
    ],
    "deviceDisplayName": "BLE Heart Monitor",
    "adminState": true,
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
        "versionSupport": [
            "5.3"
        ],
        "deviceMacAddress": "CA:2B:5C:EC:95:46",
        "isRandom": false,
        "mobility": true,
        "pairingMethods": [
            "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device"
        ],
        "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": null,
        "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
            "key": null
        }
    },
    "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
        "onboardingUrl": "onboardApplication",
        "deviceControlUrl": [
            "controlApplication"
        ],
        "dataReceiverUrl": []
    }
}'
```

Response Format for Onboarding the Device with Pairing

```
{
  "schemas": [
    "urn:ietf:params:scim:schemas:core:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
  ],
  "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "deviceDisplayName": "BLE Heart Monitor",
  "adminState": true,
  "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
    "versionSupport": [
      "5.3"
    ],
    "deviceMacAddress": "CA:2B:5C:EC:95:46",
    "isRandom": false,
    "pairingMethods": [
      "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device"
    ],
    "mobility": true,
    "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": {},
    "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
      "key": 0
    },
```

```
    "urn:ietf:params:scim:schemas:extension:pairingPassKey:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingOOB:2.0:Device": {
      "key": "",
      "randNumber": 0,
      "confirmationNumber": 0
    }
  },
  "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
    "onboardingUrl": "onboardApplication",
    "deviceControlUrl": [
      "controlApplication"
    ],
    "dataReceiverUrl": []
  }
}
```

Connecting to a Device

To connect to a BLE device and discover its services, perform the following:

1. Use the Device ID:

   The device ID is `"f837c89a-e1dd-4b90-a446-06016c0d2b75"`.

2. Connect the BLE device from the AP:

   Use the device ID to establish a connection with the BLE device from the AP.

3. Discover Services:

   Discover the services with the following service UUIDs:

   - 1800
   - 2002
   - 1822
   - 180F

API Definition

API	Description
`/control/connectivity/connect`	This API is used to connect to a device.

Request Format

```
{
    "technology": "ble",
    "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
    "ble": {
        "services": [
            {
                "serviceID": "1800"
```

```
            },
            {
                "serviceID": "2022"
            },
            {
                "serviceID": "1822"
            },
            {
                "serviceID": "180F"
            }
        ]
    },
    "controlApp": "controlApplication"
}
```

Response Format

```
{
  "status": "SUCCESS",
  "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "services": [
    {
      "serviceID": "1800",
      "characteristics": [
        {
          "characteristicID": "6367",
          "descriptors": [
            {
              "descriptorID": "7751"
            }
          ],
          "flags": [
            "read",
            "write",
            "notify"
          ]
        }
      ]
    },
    {
      "serviceID": "2022",
      "handle": 1,
      "characteristics": [
        {
          "characteristicID": "3670",
          "descriptors": [
            {
              "descriptorID": "7929"
            }
          ],
          "flags": [
            "read",
            "write",
            "notify"
          ]
        }
      ]
    },
    {
      "serviceID": "1822",
      "handle": 2,
      "characteristics": [
        {
```

```
            "characteristicID": "2198",
            "descriptors": [
              {
                "descriptorID": "5622"
              }
            ],
            "flags": [
              "read",
              "write",
              "notify"
            ]
          }
        ]
      },
      {
        "serviceID": "180F",
        "handle": 2,
        "characteristics": [
          {
            "characteristicID": "2198",
            "descriptors": [
              {
                "descriptorID": "5622"
              }
            ],
            "flags": [
              "read",
              "write",
              "notify"
            ]
          }
        ]
      }
    ],
    "requestID": "251ea636-8708-43b0-b8f8-438332098c9d",
    "reason": "NO_INFO"
}
```


Writing a Characteristic to the Device

API Definition

API	Description
`/control/data/write`	This API is used to write a characteristic to the device.

Request Format


```
{
    "technology": "ble",
    "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
    "value": "0001",
    "ble": {
        "serviceID": "1800",
        "characteristicID": "2A00"
    },
    "controlApp": "controlApplication"
}
```

Response Format

```
{
  "status" : "SUCCESS",
  "id" : "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "requestID" : "621f4b8c-c1eb-461f-ac2d-818a9193f10e"
}
```

Reading a Characteristic to the Device

API Definition

API	Description
/control/data/read	This API is used to read a characteristic to the device.

Request Format

```
{
    "technology": "ble",
    "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
    "ble": {
        "serviceID": "1800",
        "characteristicID": "2A00"
    },
    "controlApp": "controlApplication"
}
```

Response Format

```
{
  "status" : "SUCCESS",
  "id" : "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "value" : "2342432",
  "requestID" : "7245ed08-11ff-4a3d-a023-9fe8c8fec89f"
}
```

Disconnecting the Device

API Definition

API	Description
/control/connectivity/disconnect	This API is used to disconnect the device.

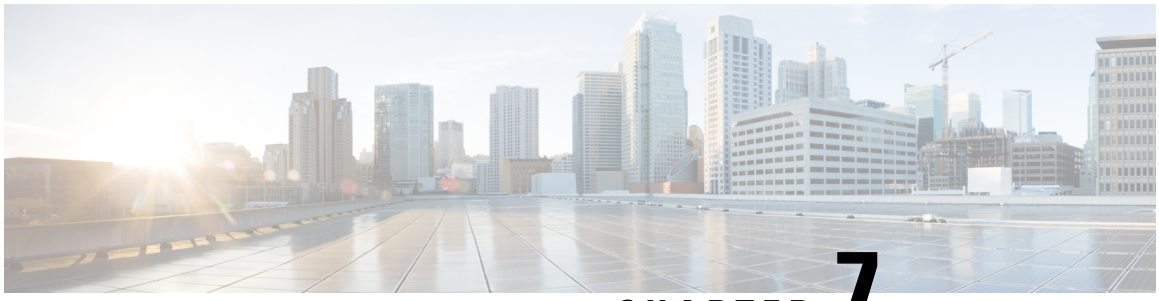
Request Format

```
{
    "technology": "ble",
```

```
    "id" : "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
    "controlApp" : "controlApplication"
}
```

Response Format

```
{
  "status" : "SUCCESS",
  "id" : "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "requestID" : "d1ef1fdf-d70b-4674-bb09-ff85870ad18d"
}
```



CHAPTER 7

Use Case 3 - BLE Notification-based Use Cases

Use Case 3 - BLE Notification-Based Use Cases

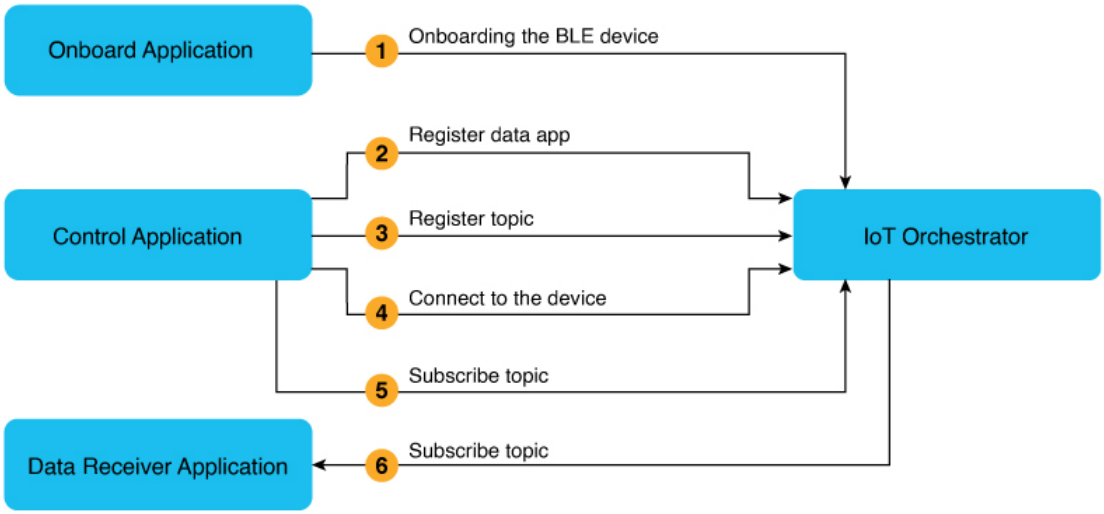
A BLE device can have multiple services and each service can have multiple characteristics. For BLE notification, you can register the topic and subscribe for a specific service or characteristic ID for which the notification needs to be received. In this case, the best example is to subscribe for BLE GATT notification.

In this use case, you will be able to onboard a device, register a topic, connect to a device, and subscribe for notification.

Here, the sample use case is that of a Pulse Oximeter. A Pulse Oximeter is a medical device that non-invasively measures the oxygen saturation level (SpO2) of a person's blood and pulse rate. The Pulse Oximeter is a small portable device commonly used in hospitals, clinics, and home to monitor patients with respiratory or cardiovascular conditions.

The following are the GATT attributes associated with the Pulse Oximeter:

1. Oxygen level
2. Heart rate



The following workflow covers the sequence of operations:

1. Onboarding the BLE Device.
2. Registering the Data Receiver Application
3. Registering a Topic for Application Events.
4. Registering a Topic for GATT Notifications.
5. Connecting to a BLE Device from an AP.
6. Starting Notifications.
7. MQTT Subscription Messages.
8. Application Events.

Onboarding the BLE Device

API Definition

API	Description
`POST /scim/v2/Devices`	This API is used to onboard the BLE device.

Request Format for Onboarding the Device without Pairing

```
curl -k --location 'https://173.39.84.53:8081/scim/v2/Devices' \
--header 'Content-Type: application/json' \
--header 'x-api-key: 596f0eba1668e6afeb3b25669990b7d84b8bf70408f21465a05e227664763024' \
--data '{
    "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
```

```
        "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
    ],
    "deviceDisplayName": "BLE Heart Monitor",
    "adminState": true,
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
        "versionSupport": [
            "5.3"
        ],
        "deviceMacAddress": "CA:2B:5C:EC:95:46",
        "isRandom": false,
        "mobility": false,
        "pairingMethods": [
            "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device"
        ],
        "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": null,
        "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
            "key": null
        }
    },
    "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
        "onboardingUrl": "onboardApplication",
        "deviceControlUrl": [
            "controlApplication"
        ],
        "dataReceiverUrl": []
    }
}'
```

Response Format for Onboarding the Device without Pairing

```
{
  "schemas": [
    "urn:ietf:params:scim:schemas:core:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
  ],
  "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "deviceDisplayName": "BLE Heart Monitor",
  "adminState": true,
  "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
    "versionSupport": [
      "5.3"
    ],
    "deviceMacAddress": "CA:2B:5C:EC:95:46",
    "isRandom": false,
    "pairingMethods": [
      "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device"
    ],
    "mobility": false,
    "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": {},
    "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingPassKey:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingOOB:2.0:Device": {
      "key": "",
      "randNumber": 0,
      "confirmationNumber": 0
    }
  },
  "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
```

```
    "onboardingUrl": "onboardApplication",
    "deviceControlUrl": [
      "controlApplication"
    ],
    "dataReceiverUrl": []
  }
}
```

Request Format for Onboarding the Device with Pairing

```
curl -k --location 'https://173.39.84.53:8081/scim/v2/Devices' \
--header 'Content-Type: application/json' \
--header 'x-api-key: 596f0eba1668e6afeb3b25669990b7d84b8bf70408f21465a05e227664763024' \
--data '{
    "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
        "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
    ],
    "deviceDisplayName": "BLE Heart Monitor",
    "adminState": true,
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
        "versionSupport": [
            "5.3"
        ],
        "deviceMacAddress": "CA:2B:5C:EC:95:46",
        "isRandom": false,
        "mobility": true,
        "pairingMethods": [
            "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device"
        ],
        "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": null,
        "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
            "key": null
        }
    },
    "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
        "onboardingUrl": "onboardApplication",
        "deviceControlUrl": [
            "controlApplication"
        ],
        "dataReceiverUrl": []
    }
}'
```

Response Format for Onboarding the Device with Pairing

```
{
  "schemas": [
    "urn:ietf:params:scim:schemas:core:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:ble:2.0:Device",
    "urn:ietf:params:scim:schemas:extension:endpointapps:2.0:Device"
  ],
  "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "deviceDisplayName": "BLE Heart Monitor",
  "adminState": true,
  "urn:ietf:params:scim:schemas:extension:ble:2.0:Device": {
    "versionSupport": [
      "5.3"
    ],
    "deviceMacAddress": "CA:2B:5C:EC:95:46",
```

```
    "isRandom": false,
    "pairingMethods": [
      "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device"
    ],
    "mobility": true,
    "urn:ietf:params:scim:schemas:extension:pairingNull:2.0:Device": {},
    "urn:ietf:params:scim:schemas:extension:pairingJustWorks:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingPassKey:2.0:Device": {
      "key": 0
    },
    "urn:ietf:params:scim:schemas:extension:pairingOOB:2.0:Device": {
      "key": "",
      "randNumber": 0,
      "confirmationNumber": 0
    }
  },
  "urn:ietf:params:scim:schemas:extension:endpointAppsExt:2.0:Device": {
    "onboardingUrl": "onboardApplication",
    "deviceControlUrl": [
      "controlApplication"
    ],
    "dataReceiverUrl": []
  }
}
```

Registering the Data Receiver Application

You need to register a data receiver application with the app ID "dataApplication" to subscribe to the topic "enterprise/hospital/pulse_oximeter".

Request Format:

```
curl -k --location 'https://173.39.84.53:8081/control/registration/registerDataApp' \
--header 'Content-Type: application/json' \
--header 'x-api-key: b922958a7b030857b3e40f8d6a4db58f6a2bd65741f47fb120517ff8bb6fb0cb' \
--data '{
    "controlApp": "controlApplication",
    "topic": "enterprise/hospital/pulse_oximeter",
    "dataApps": [
        {
            "dataAppID": "dataApplication"
        }
    ]
}'
```

Response Format:

```
{
  "status" : "SUCCESS",
  "topic" : "enterprise/hospital/pulse_oximeter",
  "reason" : "Data applications successfully registered!"
}
```

Registering a Topic for Application Events

You can register the topic "`enterprise/hospital/app_events`" with the gateway to subscribe to application events.

API Definition

API	Description
`/control/registration/registerTopic`	This API is used to register a topic with the gateway to subscribe to GATT notifications from the device.

Request Format

```
{
"technology": "ble",
"controlApp": "controlApplication",
"topic": "enterprise/hospital/app_events",
"ble": {
"type": "application_events"
}
}
```

Response Format

```
{
"status" : "SUCCESS",
"topic" : "enterprise/hospital/app_events",
"reason" : "Application event topics successfully registered!"
}
```

Registering a Topic for GATT Notifications

You can register the topic "`enterprise/hospital/pulse_oximeter`" with the gateway to subscribe to GATT notifications from the device ID "`df553860-975c-4d7f-8c1f-b2996f34e26f`", service UUID "`1800`", and characteristic UUID "`2A5E`".

API Definition

API	Description
`/control/registration/registerTopic`	This API is used to register a topic with the gateway to subscribe to GATT notifications from the device.

Request Format

```
{
    "technology": "ble",
    "ids": [
```

```
        "79ca0f72-4ae2-436a-8bd4-24a2770faa83"
    ],
    "controlApp": "controlApplication",
    "topic": "enterprise/hospital/pulse_oximeter",
    "dataFormat": "default",
    "ble": {
        "type": "gatt",
        "serviceID": "1800",
        "characteristicID": "2A5E"
    }
}
```

Response Format

```
{
  "status" : "SUCCESS",
  "topic" : "enterprise/hospital/pulse_oximeter",
  "reason" : "Gatt topics successfully registered!",
  "registeredIDs" : [ "79ca0f72-4ae2-436a-8bd4-24a2770faa83" ]
}
```

Connecting to a BLE Device from an AP

API Definition

API	Description
/control/connectivity/connect	This API is used to connect to the BLE device from the AP and discover the services with service UUIDs 1800, 2022, 1822, and 180F.

Request Format

```
{
    "technology": "ble",
    "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
    "ble": {
        "services": [
            {
                "serviceID": "1800"
            },
            {
                "serviceID": "2022"
            },
            {
                "serviceID": "1822"
            },
            {
                "serviceID": "180F"
            }
        ]
    },
    "controlApp": "controlApplication"
}
```

Response Format


```
{
  "status": "SUCCESS",
  "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "services": [
    {
      "serviceID": "1800",
      "characteristics": [
        {
          "characteristicID": "6367",
          "descriptors": [
            {
              "descriptorID": "7751"
            }
          ],
          "flags": [
            "read",
            "write",
            "notify"
          ]
        }
      ]
    },
    {
      "serviceID": "2022",
      "handle": 1,
      "characteristics": [
        {
          "characteristicID": "3670",
          "descriptors": [
            {
              "descriptorID": "7929"
            }
          ],
          "flags": [
            "read",
            "write",
            "notify"
          ]
        }
      ]
    },
    {
      "serviceID": "1822",
      "handle": 2,
      "characteristics": [
        {
          "characteristicID": "2198",
          "descriptors": [
            {
              "descriptorID": "5622"
            }
          ],
          "flags": [
            "read",
            "write",
            "notify"
          ]
        }
      ]
    },
    {
```

```
      "serviceID": "180F",
      "handle": 2,
      "characteristics": [
        {
          "characteristicID": "2198",
          "descriptors": [
            {
              "descriptorID": "5622"
            }
          ],
          "flags": [
            "read",
            "write",
            "notify"
          ]
        }
      ]
    }
  ],
  "requestID": "251ea636-8708-43b0-b8f8-438332098c9d",
  "reason": "NO_INFO"
}
```

Starting Notifications

API Definition

API	Description
/control/data/subscribe	This API enables the GATT notifications in the device using the device ID, GATT service UUID, and characteristic UUID.

Request Format

```
{
    "technology": "ble",
    "id": "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
    "ble": {
        "serviceID": "1800",
        "characteristicID": "2A5E"
    },
    "controlApp": "controlApplication"
}
```

Response Format

```
{
  "status" : "SUCCESS",
  "id" : "79ca0f72-4ae2-436a-8bd4-24a2770faa83",
  "requestID" : "5aa21160-24be-4bdb-8206-9de0f5c24898",
  "serviceID" : "1800",
  "characteristicID" : "2a5e"
}
```

MQTT Subscription Messages

Once the notification is enabled using the subscribe API, the data receiver application starts receiving the GATT notifications once it subscribes to the registered topic.

```
mosquitto_sub -h localhost -p 41883 -t '$share/group/enterprise/hospital/pulse_oximeter'
-u
dataApplication --pw d5a4c7b1afd862a070514528006f22d4964f6c61ec0e2e0b6c3ebd03c2fbb507

$dbdb70f8-b02d-4733-9d1e-8875fa74bda03baf13f
18002a5e

$dbdb70f8-b02d-4733-9d1e-8875fa74bda03baf13f
18002a5e

$dbdb70f8-b02d-4733-9d1e-8875fa74bda03baf13f
18002a5e

$dbdb70f8-b02d-4733-9d1e-8875fa74bda03baf13f
18002a5e

$dbdb70f8-b02d-4733-9d1e-8875fa74bda03baf13f
18002a5e
```

Note The partner application must run multiple instances of the **mosquitto_sub** command. The number of instances is based on the Cisco Catalyst 9800 Wireless Controller platform on which the IoT Orchestrator is deployed. For more information, see the Table 1: Platforms and MQTT Subscribers (Recommended).

Application Events

Application events apply to use cases that require BLE device connection. If a BLE device is connected and any of the following scenarios occur:

- IoT Orchestrator application restarts
- Cisco Catalyst 9800 Wireless Controller reloads
- Cisco Catalyst 9800 Wireless Controller HA switchover

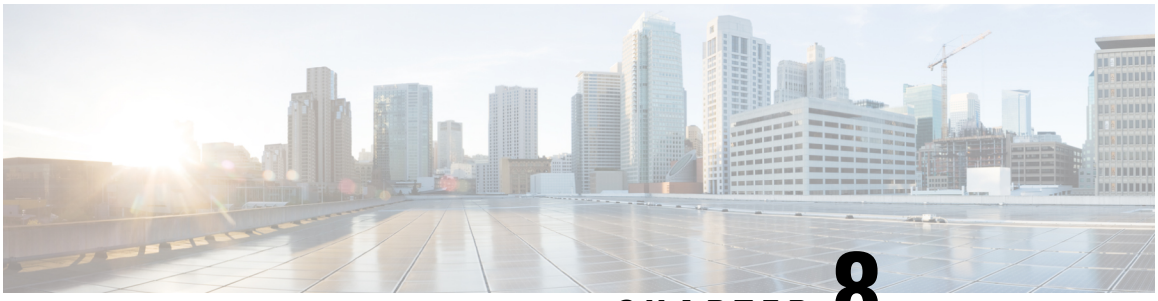
Then the partner application must register for the **application_events** topic to receive notifications such as *Application Restarted* or *Application Stopped*.

- *Application Restarted*: The IoT Orchestrator application has successfully restarted.
- *Application Down*: The IoT Orchestrator application has been gracefully shutdown.

Upon receiving an *Application Restarted* notification message, the partner application must issue the "NIPC Connect Request API" for devices that were connected prior to the IoT Orchestrator application restart. Without this action, the connection to those BLE devices will be lost.

Once the application events are enabled using the `registerTopic` API, the data receiver application starts receiving the application events once it subscribes to the registered topic.

```
mosquitto_sub -h localhost -p 41883 -t '$share/group/enterprise/hospital/app_events' -u
dataApplication --pw d5a4c7b1afd862a070514528006f22d4964f6c61ec0e2e0b6c3ebd03c2fbb507

APPLICATION RESTARTED
```



CHAPTER 8

Troubleshooting

Troubleshooting

Issue	Solution	Verification
If the Cisco Wireless AP does not show up as Connected in the AP Inventory page in IoT Orchestrator application.	Verify, if the **Configure 9800 WLC** displays as Success. If not, see the **Day 0 - Deploying IoT Orchestrator Application on Cisco Catalyst 9800 Wireless Controller** section in *Cisco Sensor Connect for IoT Services Configuration Guide*.	Verify, if the Cisco Wireless AP is connected to the Cisco Catalyst 9800 Wireless Controller in the IoT Orchestrator running as an IOx application. Verify, the network reachability from the Cisco Wireless AP to the IoT Orchestrator application IP address.
If the API call is rejected by the IoT Orchestrator when the on-premise application is registered using the API key.	Verify, if the application issuing the API call is registered in the IoT Orchestrator application.	For applications with API key-based registration, you will need to verify: • If the application issuing the API call has the application name registered in the IoT Orchestrator. • If the IoT Orchestrator application uses the same API key when registering the application.

Issue	Solution	Verification
If the API call is rejected by the IoT Orchestrator when the on-premise application is registered using the certificate name.	Verify, if the application issuing the API call is registered in the IoT Orchestrator application.	For applications with Certificate-based registration, you will need to verify: • If the application issuing the API call has the application name registered in the IoT Orchestrator. • If the IoT Orchestrator application uses the same certificate whose canonical name is registered.
If the Connect API call is rejected with the “Control app is not authorized” message.	Verify, if the control application issuing the connect API call for a specific BLE device is specified in the "deviceControlUrl" field of the "endpoints" section in the Onboarding API request for that device.	
If the IoT Orchestrator application is unable to connect to the BLE device.	Verify, if the BLE device can connect and send out BLE advertisements. Most BLE devices stop advertising when connected to another mobile application. To reconnect, the user must first disconnect the device, which will then advertise for reconnection upon issuing the connect API.	If the BLE device does not need pairing, you will need to check, if the onboarded device used the pairing method as “PairingNull”. If the BLE device needs pairing, you will need to check, if the onboarded device used the pairing method as “PairingJustWorks”.
If the MQTT receiver in the application does not receive streaming data from the IoT Orchestrator.	Verify, if the data application is registered using the control application. Verify, if the topic is registered for the data of interest using the control application. Verify, if the subscription for the topic is done using the control application.	

Issue	Solution	Verification
How to determine the real-time logs from the IoT Orchestrator application.	Refer to the **Logs** section in *Cisco Sensor Connect for IoT Services Configuration Guide*.	
How can I capture the logs from the Cisco Wireless AP that are connected to the IoT Orchestrator application.		
How can I capture the logs for a specific BLE device from the IoT Orchestrator application and Cisco Wireless AP.		

Troubleshooting