Instrument Applications with the IoT REST APIs

On this page:

The IoT REST APIs enables you to report instrumentation data directly to the EUM Server. You can use any platform or language that has support for HTTPS requests and JSON.

This getting started describes how to create a JSON request body, form the resource URI, and make an HTTPS request to the IoT REST APIs to report instrumentation data for the three supported events.

Follow the steps below to get your EUM App Key and use the IoT REST API:

Check the Requirements

Before you get started, make sure you meet the following requirements:

- Get an EUM App Key
- Platform/language that supports HTTPS requests.
- JSON support

Form the IoT REST URLs

To form the IoT Monitoring REST resource URL, you will need to know the IoT REST API base URL and port as well as your App Key.

IoT REST API Base URL

The IoT REST API base URL depends on your deployment:

```
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1
```

IoT Endpoints

With your App Key, you can form the IoT resource endpoints. See the Summary of the IoT endpoints for the list of supported resource endpoints and their descriptions.

Create the JSON Request Body

You report device information and events in a JSON request body. The JSON includes an array of `beacon` objects, with each beacon object containing device data and events. The array enables you to transmit data from multiple devices in one request to the EUM Server. You can transmit up to 200 beacons per request.

Save the JSON below to a file (e.g., `testBeacon.json`) and replace the values for the `timestamp` properties with integers representing the UNIX Epoch time in milliseconds. The JSON contains the three supported events `customEvents`, `networkRequest Events`, and `errorEvents` for a smart thermostat. In the next two steps, you will validate and send the JSON as a beacon to the IoT REST API.

```
[
 {
  "deviceInfo": {
   "deviceType": "Thermostat",
   "deviceId": "4e75d70d-a3f9-474b-bacf-0f4a57fa944c"
  },
  "versionInfo": {
    "hardwareVersion": "Board Rev. 13A",
    "firmwareVersion": "123.5.31",
    "softwareVersion": "9.1.3",
    "operatingSystemVersion": "Linux 13.4"
  },
  "customEvents": [
    {
      "timestamp": <UNIX_Epoch_time_in_milliseconds>,
      "eventType": "Temperature Reading",
      "eventSummary": "Temperature: 25° c",
      "doubleProperties": {
        "celsius": 25.0
      }
    }
  ],
  "networkRequestEvents": [
   {
    "timestamp": <UNIX_Epoch_time_in_milliseconds>,
    "duration": 245,
    "url": "https://api.company.com/v1/temperature",
    "statusCode": 200,
    "requestContentLength": 32,
    "responseContentLength": 0,
    "doubleProperties": {
     "reportedTemperature": 25.0
    }
   }
  ],
  "errorEvents": [
   {
    "timestamp": <UNIX_Epoch_time_in_milliseconds>,
    "name": "SQLException",
    "message": "open() failed because db is locked"
   }
  ]
 }
]
```

Transmit the Beacon Data

To send the beacon, you post the JSON request body to the `/beacons` endpoint. Again, in this cURL example, use the JSON you saved to the file `testBeacon.json` and replace <appKey> with your EUM App Key:

```
curl -v -X POST -d '@testBeacon.json'
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<appKey>/beacons
```

If the beacons were transmitted successfully, the IoT REST API will return the HTTP Status Code 202:

```
< HTTP/1.1 202 Accepted
```

Verify the Instrumentation in the Controller UI

See Confirm the IoT Application Reported Data to the Controller to verify the instrumentation.

Correlate Business Transactions with Network Requests (Optional)

To correlate business transactions (BTs) with network requests, you need to have instrumented a business application and enabled business transactions in the Controller UI. See Correlate Business Transactions for IoT Monitoring to learn more.

The steps below show you how to get the BT response headers and use them to correlate the BT with an IoT Network Request event.

1. Make a network request that includes the AppDynamics HTTP request headers `ADRUM` and ADRUM_1 to one of your business applications:

   ```
   curl -H "ADRUM: isAjax:true" -H "ADRUM_1: isMobile:true" -H "Accept:
   application/json" -H "Content-Type: application/xml" -H
   "Content-Length: 0" -X GET http://<url_to_business_app>
   ```

2. The business application will return response headers that contain information for correlating business transaction. If you were to print these BT response headers, you would see something like the following:

```
ADRUM_0: clientRequestGUID:a27ce4da-d4e6-4bf5-bbca-9b1751aa44a4
ADRUM_1:
globalAccountName:customer1_78203698-278e-428f-8726-bb381219c6cb
ADRUM_2: btId:4423
ADRUM_3: btERT:267
ADRUM_4: btDuration:368
Content-Length: 469
Server: Jetty(9.4.z-SNAPSHOT)
```

3. Create a beacon file `btCorrelation.json` with the returned BT response headers (only those headers that include `ADRUM_*`). You assign the returned `ADRUM_*` response headers from the network event request to the business application to the object `responseHeaders` in the beacon as shown below.

```
[
      {
         'deviceInfo':{
            'deviceId':'1111',
            'deviceName':'AudiS3',
            'deviceType':'SmartCar'
         },
         'versionInfo':{
            'hardwareVersion':'1.0',
            'firmwareVersion':'1.0',
            'softwareVersion':'1.0',
            'operatingSystemVersion':'1.0'
         },
         'networkRequestEvents':[
            {
               'url':'<url_to_business_app>',
               'statusCode':200,
               'responseHeaders':{
                  'ADRUM_0':[
                     '<value_returned_from_business_app>'
                  ],
                  'ADRUM_1':[
                     '<value_returned_from_business_app>'
                  ],
                  'ADRUM_2':[
                     '<value_returned_from_business_app>'
                  ],
                  'ADRUM_3':[
                     '<value_returned_from_business_app>'
                  ]
               },
               'timestamp':1525226857000,
               'duration':0,
               'requestContentLength':0,
               'responseContentLength':457
            }
         ]
      }
]
```

4. Send the beacon containing the BT headers to the EUM Server with a cURL command similar to the one here:

```
curl -I -H "Content-Type: application/json" -H "Accept:
application/json" -X POST -d @btCorrelation.json -H
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/
<appKey>/beacons
```

5. For a successful call, the response headers should be similar to the following:

```
HTTP/1.1 202 Accepted
Cache-Control: private, no-cache, no-store, must-revalidate,
max-age=0, proxy-revalidate, s-maxage=0
Expires: 0
Pragma: no-cache
Vary: *
Transfer-Encoding: chunked
Via: 1.1 sjc12-dmz-wsa-5.cisco.com:80 (Cisco-WSA/X)
Connection: keep-alive
```

6. In the Controller UI, you should be able to view the correlated business transaction in the **Device Details** dialog.

Customize the IoT REST API Instrumentation (Optional)

You can further customize the IoT instrumentation using the IoT REST API. See the latest IoT REST API documentation or the previous versions listed below:

- https://docs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.0/
- https://docs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.1/
- https://docs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.2/
- https://docs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.4/

Run the Sample Python Application

The sample Python application uses the IoT REST API to send sample data for Custom, Network Request, and Error events. The Network Request events include correlated business transactions. The data mocks a smart car application, capturing usage information, network performance, and errors.

To run the sample app, follow the instructions given in the GitHub repository iot-rest-api-sample-apps.

Troubleshoot the IoT REST API Instrumentation

The sections below provide instructions for troubleshooting your IoT REST API Instrumentation.

Verify Your IoT App Has Been Enabled

Using your App Key, verify that your IoT app has been enabled:

```
curl -v -X GET
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<appKey>/enabled
```

If your App Key has been enabled, you should get the following response:

```
< HTTP/1.1 200 OK
< Cache-Control: private, no-cache, no-store, must-revalidate,
max-age=0, proxy-revalidate, s-maxage=0
< Date: Sat, 19 Aug 2017 01:20:39 GMT
< Expires: 0
< Pragma: no-cache
< Vary: *
< Content-Length: 0
< Connection: keep-alive
```

If the App Key does not exist:

```
< HTTP/1.1 403 Forbidden
```

Validate Beacons

You can use the validate beacon endpoint (`/validate-beacons`) to verify that the beacon's JSON request body complies with the REST API schema.

> You are not required or recommended to validate beacons before transmitting them. You should only use this endpoint in development for testing and troubleshooting.

In this cURL example, you are verifying that the JSON given in the file `testBeacon.json` is valid. Replace `<appKey>` with your EUM App Key.

```
curl -v -X POST -d '@testBeacon.json'
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<appK
ey>/validate-beacons
```

If the JSON request body containing the beacon data is valid, the IoT Monitoring REST API will return the HTTP Status 200:

```
HTTP/1.1 200 OK
```

If the JSON request body is invalid, the IoT REST API will return the HTTP Status 422 and a response body with the description of the error message.

```
< HTTP/1.1 422 Unprocessable Entity
```