Oracle® Cloud

Device Connectivity Guide for Oracle Fusion Cloud IoT Intelligent Applications

23.3.1
E86271-29
July 2023

ORACLE

Oracle Cloud Device Connectivity Guide for Oracle Fusion Cloud IoT Intelligent Applications, 23.3.1

E86271-29

Primary Author: Oracle Corporation

Contents

3 Integrate Oracle IoT Cloud Service with Third Party Device Management Applications

Preface

Welcome to the developer documentation for Oracle Fusion Cloud IoT Intelligent Applications. Use this documentation to learn how to access Oracle Fusion Cloud IoT Intelligent Applications, manage the devices connected to an Oracle Fusion Cloud IoT Intelligent Applications instance, deploy software to those devices, analyze data from those devices in real time, and integrate that data with enterprise applications, web services, or with other Oracle Cloud Oracle Cloud Services, such as Oracle Business Intelligence Cloud Service, JD Edwards, and Oracle Mobile Cloud Service.

Topics:

Audience

This documentation is designed for application developers who create software applications that use the Oracle Fusion Cloud IoT Intelligent Applications client software libraries and REST APIs to communicate with their devices. This document assumes a familiarity with web technologies and an intermediate understanding of the Java programming language.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our

initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

For more information, see these Oracle resources:

- Oracle Cloud at http://cloud.oracle.com
- *Getting Started with Oracle Cloud*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Learn About Device Connectivity

There are many types of Internet of Things (IoT) devices, and they can connect to an IoT network using different standards and protocols. Oracle Fusion Cloud IoT Intelligent Applications supports multiple methods to connect to the devices, and you can select a method that's compatible to your enterprise's environment and devices.

Topics:

About IoT Devices

An IoT device is any device that can connect to the internet and transmits data.

IoT enables internet connectivity to any set of physical devices that are beyond standard devices such as a desktop, laptop, mobile gadgets, or tablets. To connect to the internet, these physical devices or machines are integrated with sensors, electronic devices, or digital technology. They can also connect to each other, can transfer data, and can be remotely monitored and managed. The IoT devices can be used to automate tasks in any industry, enterprise, or a home environment.

Examples and Applications of IoT Devices

- In a production environment of a factory, machines integrated with multiple sensors can provide real-time status of the factory shop floor.
- In the fleet industry, vehicles have standardized on-board diagnostics (OBD) II devices that use wireless technology to transmit real-time status, location, and the health of vehicles.
- In the construction and mining industry, assets and protection gear are tracked using sensors that communicate over radio frequencies.
- In the agriculture industry, soil conditions can be monitored by using sensors to create optimal plans for watering and fertilizing.

About the IoT Connectivity Protocols

Besides HTTP, other protocols that are optimal and suitable for communication in IoT are Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), WebSocket, Extensible Messaging and Presence Protocol (XMPP), and Advanced Message Queuing Protocol (AMQP).

REST connectivity over the internet is used as the communication architecture for the IoT devices. Typically, the IoT devices are resource constrained, and there may be data loss or a high memory requirement in this type of communication. Alternatively, a few protocols that are effective are MQTT, CoAP, XMPP, WebSocket, and AMQP.

Description and Application of the IoT Protocols

Name	Description	Security	Use Case
MQTT	Simple and lightweight IoT protocol designed for constrained devices and low network bandwidth. See mqtt.org.	In an MQTT packet, you can pass an user name and password, but it doesn't support additional security. You can use SSL in the network for encryption, independent from the MQTT protocol.	Small medical devices with limited network connectivity, mobile apps in mobile devices, sensors in remote locations that communicate with a gateway.
CoAP	Protocol based on the REST model and is suitable for constrained devices such as a microcontroller or a constrained network because it functions with minimum resources in the device or the network.	CoAP applies datagram transport layer security (DTLS) that's equivalent to 3072-bit RSA keys.	Smart energy applications and building automation applications.
WebSocket	A full-duplex communication channel over a TCP connection.	WebSocket protocol defines a `ws://` and `wss://` prefixes indicate a WebSocket and a WebSocket secure connection, respectively.	Implement WebSocket in runtime environments or libraries that act as servers or clients. You can apply WebSockets in an IoT network where chunks of data are transmitted continuously within multiple devices.
XMPP	Uses the XML text format for communication and runs over TCP. It's not fast and uses polling to check for updates when needed. See https://xmpp.org	XMPP uses a security mechanism based on Transport Layer Security (TLS) and Simple Authentication and Security Layer (SASL).	Use XMPP to connect your home thermostat to a web server so that you can access it from your phone. It's used in consumer-oriented IoT applications.
AMQP	The message queue asynchronous protocol is for communication of transactional messages between servers. See https://www.amqp.org/	AMQP provides TLS/SSL and SASL for security.	AMQP is best used in sever-based analytical functions. It's effectively used in the banking industry.

Oracle IoT Cloud Service supports HTTP and MQTT.

Methods to Connect Devices to Oracle Fusion Cloud IoT Intelligent Applications

Connect your devices to Oracle Fusion Cloud IoT Intelligent Applications by selecting a method that works for your type of device, network, and protocol.

Typically, there are three ways to connect your device to Oracle Fusion Cloud IoT Intelligent Applications.

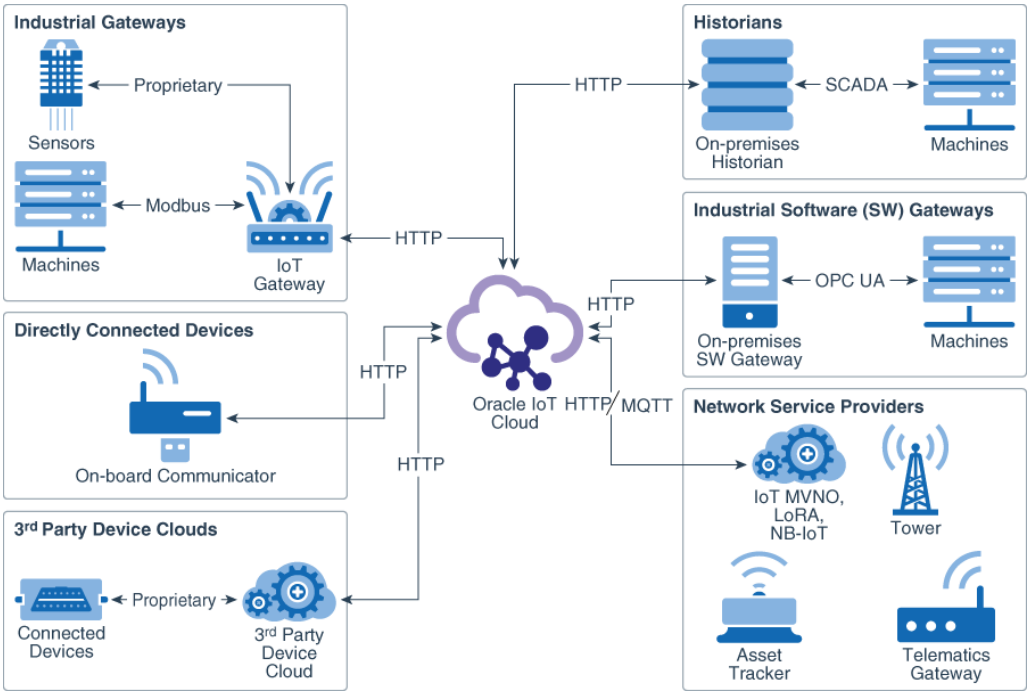
- Use Oracle IoT Client Libraries that are embedded in edge applications and enable devices to connect to Oracle Fusion Cloud IoT Intelligent Applications. They provide functionality such as security life-cycle management and bidirectional messaging. There are client libraries for different platforms such as Java, JavaScript, Android, iOS, C Posix, C, Apache Felix and Python in Windows, iOS, and Linux environments.
- Use the packaged plug-ins or adapters that acquire on-premises data, and securely communicate with Oracle Fusion Cloud IoT Intelligent Applications. These adapters enable you to quickly create prototypes and validate the connectivity. Some standard adapters are: an on-premises gateway device, an OPC UA Server, and a Historian.
- Use connectors to ingest data from existing data sources. Components of connectors allow Oracle Fusion Cloud IoT Intelligent Applications to accept data in new protocols and data formats. Connectors can perform data ingestion from packaged adapters or custom adapters, then performs data mapping and normalization to a format compliant with Oracle Fusion Cloud IoT Intelligent Applications. Some examples of packaged adapters are LoRA, M2M/telematic providers, and device partners. Custom adapters use proprietary data protocols or CSV.

About Selecting the Method to Connect Devices

You have several options to connect devices to Oracle Fusion Cloud IoT Intelligent Applications. You select the method based on the types of devices, protocols, and existing environment.

The diagram and the comparison table help you to select the best option for you to connect your devices toOracle Fusion Cloud IoT Intelligent Applications based on your setup and protocols.

Device Integration Models



Options to Select the Method

Scenario	Option to Connect the Devices	Example
Devices that support proprietary protocols.	Sensors and machines connect indirectly to Oracle Fusion Cloud IoT Intelligent Applications through a standardized industrial gateway using proprietary protocols.	A device uses the MODBUS protocol to connect with an industrial IoT gateway from vendors such as Cisco, Dell, Bosch, Intel, or Multitech which, in turn, connects to Oracle Fusion Cloud IoT Intelligent Applications using the HTTP protocol
Machine with a powerful in-built sensor that supports HTTP protocol.	Directly connect with Oracle Fusion Cloud IoT Intelligent Applications using HTTP.	A device such as an on-board communicator directly exchanges messages with Oracle Fusion Cloud IoT Intelligent Applications by using the HTTP protocol.
Devices that are already connected to an existing third-party cloud service using proprietary protocols.	Third-party clouds can connect with Oracle Fusion Cloud IoT Intelligent Applications using HTTP. This results in the devices getting indirectly connected to Oracle Fusion Cloud IoT Intelligent Applications.	A device such as an OBD II data logger connects to a third-party cloud service using its proprietary protocol. The third-party device cloud connects to Oracle Fusion Cloud IoT Intelligent Applications using the HTTP protocol.

Scenario	Option to Connect the Devices	Example
Devices and gateways that are already connected to existing network providers that provide wireless connectivity to the devices.	The network providers can transmit the device data to Oracle Fusion Cloud IoT Intelligent Applications using HTTP or MQTT.	Network providers use LoRA, NB-IoT, or IoT MVNO for wireless connectivity with devices such as an asset tracker or a telematics gateway. The network provider can transmit the device messages to Oracle Fusion Cloud IoT Intelligent Applications using HTTP or MQTT.
Existing database with device data on events, time stamps, and alarms.	The on-premises database can connect to Oracle Fusion Cloud IoT Intelligent Applications using HTTP and transfer the device messages.	Machines using the Historian service save and store messages in a database or multiple databases over the supervisory control and data acquisition (SCADA) network. An on-premises Historian service such as Pi from OSI Soft connects to Oracle Fusion Cloud IoT Intelligent Applications and provides the machine data for analysis.
Existing on-premises industrial gateway software	On-premises industrial gateway software applications manage machine data. Oracle Fusion Cloud IoT Intelligent Applications can connect to these gateway applications using HTTP to obtain the machine data.	Machines that use OPC Unified Architecture (UA), a machine-to -machine protocol to transfer machine data to industrial gateway software such as Metrikon or Kepware and that, in turn, can connect to Oracle Fusion Cloud IoT Intelligent Applications and send device messages received from the machines.

Get Started

Use this information to learn about the supported browsers, how to get support, and how to provide diagnostic information to Oracle support if you face an issue.

For details on ordering your Oracle Fusion Cloud IoT Intelligent Applications applications, activating your order, signing in to the management console, and accessing your IoT applications, refer to the *Administering Oracle Internet of Things Cloud Service* guide.

Supported Browsers

Supported Browsers

Oracle Fusion Cloud IoT Intelligent Applications supports the following web browsers:

- Google Chrome
- Microsoft Edge

- Mozilla Firefox
- Apple Safari

See Oracle Software Web Browser Support Policy for more information

Supported Platforms

The Oracle Fusion Cloud IoT Intelligent Applications gateway software and client software libraries can run on multiple platforms.

See Oracle IoT Cloud Service Client Software Certified System Configurations for the current list of supported platforms.

How to Get Support

Use these resources to resolve problems:

- Click the **Contact Us** () icon in the Oracle My Services management console and then select a support option.
- Visit the Oracle Help Center at http://docs.oracle.com/en/.
- If you're an Oracle Premier Support Customer, then visit My Oracle Support.
- Contact Oracle Technical Support. See Contacting Oracle Support in *Getting Started with Oracle Cloud*.

Locate Diagnostic Information for Oracle Support

To resolve support issues quickly, send the Oracle support team the diagnostic information listed in the table.

To locate and view diagnostic information, see Find Diagnostic Information to Help with Troubleshooting.

Diagnostic Information	Description	Example
Server Version Number	Record the version number of the Oracle Cloud instance.	17.1.5.0
Client Platform Version	Record the client platform version.	Client library information for each platform.
Java or Android Compiler Version	Record the version number of the device Java or Android compiler.	Android 7.0

Diagnostic Information	Description	Example
Error Message	Record the error message or attach a screen image.	*Operation failed. Cause: {"type":"https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html","title":"Project was not found: ProjectId{iotAppId='0-AD', serviceId='0-AD'}","status":"404"}*
Oracle Fusion Cloud IoT Intelligent Applications URL	Record the URL of the Oracle Fusion Cloud IoT Intelligent Applications instance.	`https:/targetcloudinstance.com`
Error Message Time Stamp	Record the time and date the error occurred.	Tue, Feb 7, 2017 1:07:42 PM
Log File	Provide a log file that documents the issue.	*log.txt*
Screen Images	Attach relevant screen images of the error message or the page where the error occurred.	Include a screen image with annotations.
Navigation Path	Provide the navigation path to the page where the error occurred.	*Device Model* > *Device Model Details*

2

Develop Device Software Using the Client Software Libraries

Oracle Fusion Cloud IoT Intelligent Applications client software libraries are provided to simplify how the devices work with Oracle Fusion Cloud IoT Intelligent Applications.

Topics

Device Virtualization

Virtualization is the act of creating a virtual (rather than physical) version of a computer application, storage device, or a computer network resource. Virtualization makes the logical server, storage, and network independent of the deployed physical infrastructure resources.

Oracle Internet of Things Cloud Service uses virtualization to make it easier to integrate external devices and data with Oracle Internet of Things Cloud Service. Oracle Internet of Things Cloud Service exposes every connected device as a set of resources called a device model. The use of device virtualization abstracts any complexity associated with device connectivity and standardizes device integration with the enterprise. With it, enterprise applications can directly address any device from Oracle Internet of Things Cloud Service regardless of network protocol and firewall restrictions.

A device model is a single entity that represents the interface through which Oracle Internet of Things Cloud Service can interact with a specific device type, regardless of the vendor, underlying standard, or specification that defined that device model. It can represent any object on the device side that can send messages or respond to REST requests. This object type includes devices, gateways, device adapters, and device applications. Through a device model, Oracle Internet of Things Cloud Service has access to the following:

- metadata associated with a device type
- message formats generated by a device

- exposed web resources that can be used to send commands
- device capabilities, such as device software management

Use Media with the Client Libraries

Using media with the Oracle IoT Cloud Service Device Libraries, such as images, videos, or other large binaries, involves the Oracle Storage Cloud Service.

To allow links and references in device model attributes and device model format fields, the `URI` type has been added to the list of device model attribute types. This URI type is encoded as a JSON String for communication, but adheres to the format restrictions of the Universal Resource Identifier format as defined in **RFC 3986**. A URI provides a standard syntax, while at the same time allowing flexibility in defining the semantics of what a resource is. The basic syntax of a URI is `[scheme:]scheme-specific-part[#fragment]`. With the Java SE client libraries, this may be represented as a **java.net.URI** class. For other client libraries, use an equivalent data type, or a string if none is available.

Storage Cloud Service

The Oracle Storage Cloud Service must be used to store media provided by the client and referenced within device model attributes. See the Oracle Storage Cloud Service documentation for more details. Note that the administrator may want to establish user accounts for the specific purposes of device media storage and configure these accounts to restrict access to only specific containers within the cloud service. Storage service credentials should be independent of other storage service credentials provisioned within the Oracle cloud.

The following fields need to be provisioned:

- The storage cloud service server name (for example, `a210401.storage.oraclecloud.com`)
- The storage service identity (for example, `Storage-a210401`)
- The storage service container (for example, `MediaStorageContainer`)
- The storage service username
- The storage service password

Device Virtualization in the Java Client Libraries

Using device virtualization, an application first creates an `oracle.iot.client.StorageObject` object via the `Client.createStorageObject()` method. The name parameter is used as the unique name of the object in the storage cloud REST API (i.e., the 'object' in `/v1/{account}/{container}/{object}`).

If content is being uploaded, the application sets the input path on the `StorageObject` via the `setInputPath()` method. The storage object is then set as the value of an attribute in the virtual device, and the attribute has to have type `URI` in the device model. The content is then uploaded. The `DataMessage` for the attribute, format or field is not sent until the content upload is complete. An error from uploading content to the storage cloud will result in an `onError()` callback to the virtual device.

If content is being downloaded, the application sets the output path on the `StorageObject` via the `setOutputPath()` method, and then calls `sync()` on the storage object. The `sync()` call does not block. The caller can then add a callback for sync events to the storage object.

Upload the Sample Device Models

To complete the Oracle Fusion Cloud IoT Intelligent Applications samples, the humidity and temperature sensor device models must be uploaded to your Oracle Fusion Cloud IoT Intelligent Applications instance.

The humidity sensor device model sends a humidity reading every 5 seconds. When the maximum threshold is set, the maximum threshold value and the humidity level are included in subsequent data messages. When the default maximum humidity threshold of 60% is reached or exceeded, an alert is sent.

The temperature sensor device model sends a temperature reading every 5 seconds. The minimum and maximum temperature values are included with the outgoing data message, when the device is turned on, and whenever a value changes. When the maximum or minimum temperature thresholds are set, the maximum and minimum threshold values and the current temperature are included in subsequent data messages. When the minimum or maximum temperature thresholds are reached or exceeded, a `tooCold` or `tooHot` alert is sent. Optionally, the threshold values can be included in the alert message (the default is NO). The default thresholds for the minimum and maximum temperature are 0 and 65 degrees.

When a REST call is invoked, the minimum and maximum temperature values are reset to the current temperature. When a device is turned off, data and alert messages are not sent until it is turned on.

1. Open a text editor.
2. Copy and paste this text into the text editor for the humidity sensor device model:

```
{
    "urn": "urn:com:oracle:iot:device:humidity_sensor",
    "name": "Humidity Sensor",
    "description": "Device model for sensor that measures humidity.",
    "attributes": [
        {
            "name": "humidity",
            "description": "Measures humidity between 0% and 100%",
            "type": "INTEGER",
            "range": "0,100"
        },
        {
            "name": "maxThreshold",
            "description": "Maximum humidity threshold",
            "type": "INTEGER",
            "range": "60,100",
            "writable": true,
            "defaultValue":80
        }
    ],
    "formats": [
        {
```

```
            "urn":
"urn:com:oracle:iot:device:humidity_sensor:too_humid",
            "name": "tooHumidAlert",
            "description": "Sample alert when humidity reaches the
maximum humidity threshold",
            "type": "ALERT",
            "value": {
                "fields": [
                    {
                        "name": "humidity",
                        "type": "INTEGER",
                        "optional": false
                    }
                ]
            }
        }
    ]
}
```

3. Save the file as `HumiditySensor.json` and then close the text editor.
4. Open a command prompt and then run this `curl` command to upload the humidity sensor device model to your Oracle Fusion Cloud IoT Intelligent Applications instance:

   ```
   curl -k -u "<username>:<password>" https://
   <your_Oracle_IoT_CloudService_instance>:<instance_port_number>/iot/a
   pi/v2/deviceModels -X POST -d @HumiditySensor.json -H "Content-
   Type:application/json" -H "Accept:application/json"
   ```

   Replace *<username>*, *<password>*, and *<your_Oracle_IoT_CloudService instance>*, and *<instance_port_number>* variables with the values for your environment. For example:

   ```
   curl -k -u "joeuser@xyz.com:password" https://
   myiotserver.com:7105/iot/api/v2/deviceModels -X POST -d
   @HumiditySensor.json -H "Content-Type:application/json" -H
   "Accept:application/json"
   ```

5. Open a text editor.
6. Copy and then paste this text into the text editor for the temperature sensor device model:

   ```
   {
       "urn": "urn:com:oracle:iot:device:temperature_sensor",
       "name": "Temperature Sensor",
       "description": "Device model for sensor that measures
   temperature.",
       "attributes": [
           {
               "name": "temp",
               "description": "Measures temperature value between -20
   and +80 Cel",
               "range": "-20,80",
   ```

```
            "type": "NUMBER"
        },
        {
            "name": "unit",
            "description": "Measurement unit, such as Cel for Celsius.",
            "type": "STRING",
            "defaultValue":"C"
        },
        {
            "name": "minTemp",
            "alias": "minimumTemperature",
            "description": "The minimum value measured by the sensor
since power ON or reset",
            "type": "NUMBER"
        },
        {
            "name": "maxTemp",
            "alias": "maximumTemperature",
            "description": "The maximum value measured by the sensor
since power ON or reset",
            "type": "NUMBER"
        },
        {
            "name": "minThreshold",
            "description": "The minimum temperature threshold",
            "type": "INTEGER",
            "range": "-20,0",
            "writable": true,
            "defaultValue":0
        },
        {
            "name": "maxThreshold",
            "description": "The maximum temperature threshold",
            "type": "INTEGER",
            "range": "65,80",
            "writable": true,
            "defaultValue":70
        },
        {
            "name": "startTime",
            "description": "The time (measured in EPOCH) at which the
system was powered ON or reset",
            "type": "DATETIME"
        }
    ],
    "actions": [
        {
            "name": "reset",
            "description": "Reset the minimum and maximum measured values
to current value"
        },
        {
            "name": "power",
            "description": "Turns system ON or OFF",
            "argType": "BOOLEAN"
```

```
            }
        ],
        "formats": [
            {
                "urn":
"urn:com:oracle:iot:device:temperature_sensor:too_hot",
                "name": "tooHotAlert",
                "description": "Temperature has reached the maximum
temperature threshold",
                "type": "ALERT",
                "value": {
                    "fields": [
                        {
                            "name": "temp",
                            "type": "NUMBER",
                            "optional": false
                        },
                        {
                            "name": "unit",
                            "type": "STRING",
                            "optional": false
                        },
                        {
                            "name": "maxThreshold",
                            "type": "NUMBER",
                            "optional": true
                        }
                    ]
                }
            },
            {
                "urn":
"urn:com:oracle:iot:device:temperature_sensor:too_cold",
                "name": "tooColdAlert",
                "description": "Temperature has reached the minimum
temperature threshold",
                "type": "ALERT",
                "value": {
                    "fields": [
                        {
                            "name": "temp",
                            "type": "NUMBER",
                            "optional": false
                        },
                        {
                            "name": "unit",
                            "type": "STRING",
                            "optional": false
                        },
                        {
                            "name": "minThreshold",
                            "type": "NUMBER",
                            "optional": true
                        }
                    ]
```

```
                }
            }
        ]
}
```

7. Save the file as `TemperatureSensor.json` and then close the text editor.
8. Open a command prompt and then run this `curl` command to upload the temperature sensor device model to your Oracle Fusion Cloud IoT Intelligent Applications:

   ```
   curl -k -u "<username>:<password>" https://
   <your_Oracle_IoT_CloudService_instance>:<instance_port_number>/iot/api/v2/
   deviceModels -X POST -d @TemperatureSensor.json -H "Content-
   Type:application/json" -H "Accept:application/json"
   ```

   Replace the *<username>*, *<password>*, and *<your_Oracle_IoT_CloudService instance>*, and *<instance_port_number>* variables with the values for your environment. For example:

   ```
   curl -k -u "joeuser@abc.com:password" https://
   myiotserver.com:7105/iot/api/v2/deviceModels -X POST -d
   @TemperatureSensor.json -H "Content-Type:application/json" -H
   "Accept:application/json"
   ```

Client Library Best Practices

Make sure that your application using the Oracle Fusion Cloud IoT Intelligent Applications client library APIs performs a well-defined set of tasks for using the Client Library APIs.

> **Note:**
>
> When publishing an application, a new provisioning file must be created for the application. You can't use the same provisioning file for multiple applications.

Before you write the application, you need to make certain decisions or obtain certain information so that your application performs the necessary tasks:

- Determine the resources you will use to exchange data between your device and the server
- Determine the message formats you will use
- Determine how to communicate securely between the server and the client (using the device ID and other properties)

Once you assemble this information, you can write an application that performs the following tasks:

- Instantiates the application class
- Retrieves any properties needed by the application
- Initializes a device client

- Obtains or creates a private key for secure communication with the server
- Registers request handlers for all resources that will be used
- Retrieves resource values from the device and sends messages to the server about them, or sets them as needed
- Tracks message delivery
- At the end, releases resources by unregistering the request handlers and closing the device client.

Use the Java SE Client Software Libraries

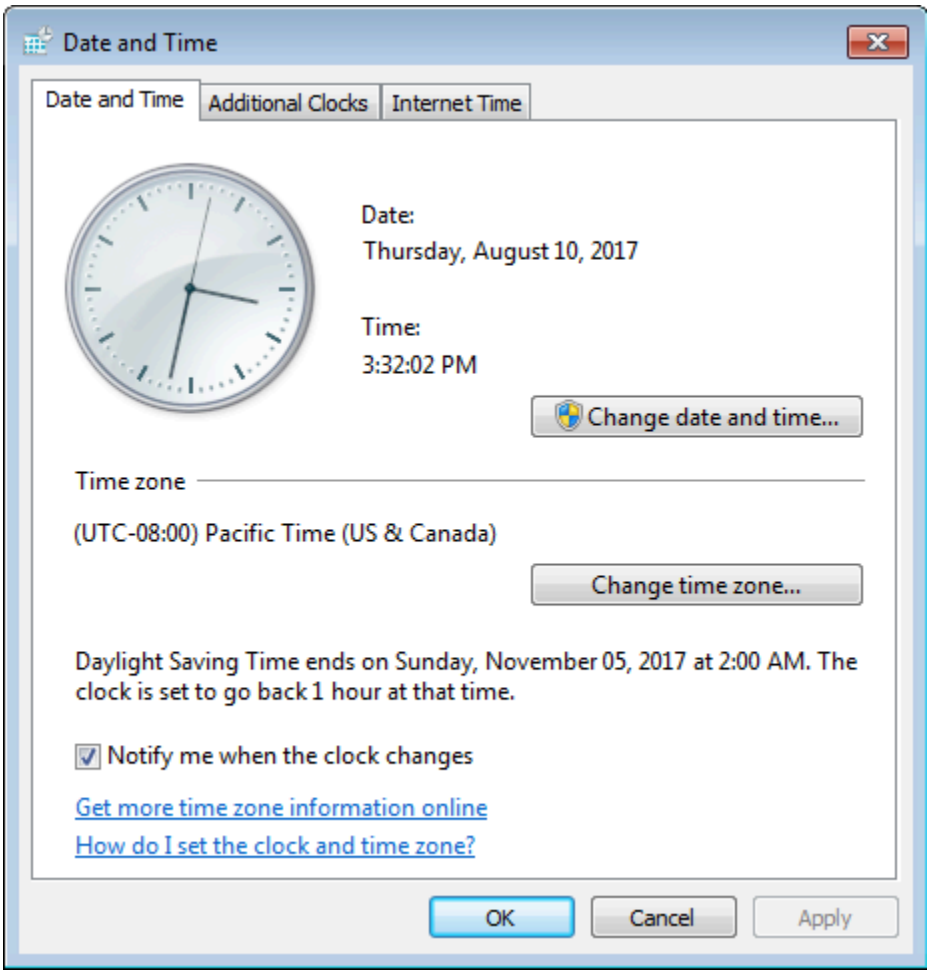
You can develop applications using the Oracle IoT Cloud Service Client Software Libraries for the Java SE platform by downloading the binary provided with the libraries. To run the examples that use the Client Software Library APIs, you can download the samples bundle provided. To customize the Client Software Libraries for your specific development environment, you can download and build the source files.

Topics

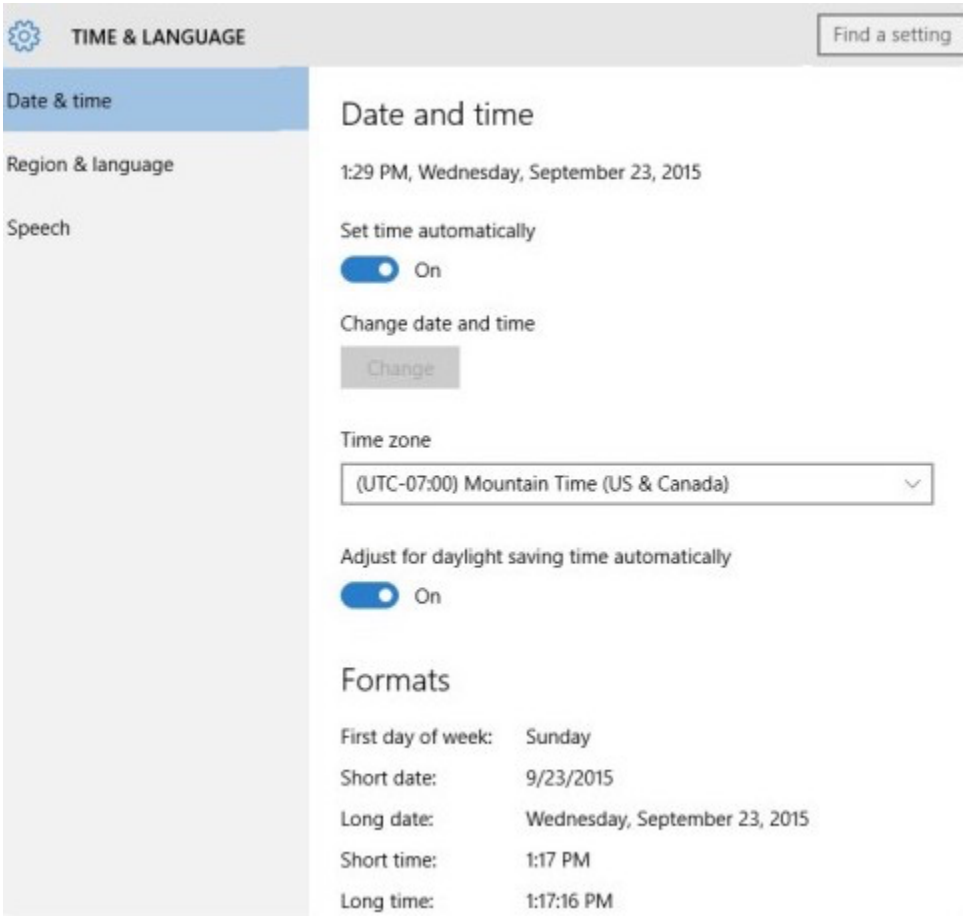
Set Up Your Development Environment to Use the Java SE Client Software Libraries

Before you can develop applications that let your devices to communicate with Oracle Fusion Cloud IoT Intelligent Applications, you first download, install, and configure the Java SE client software libraries.

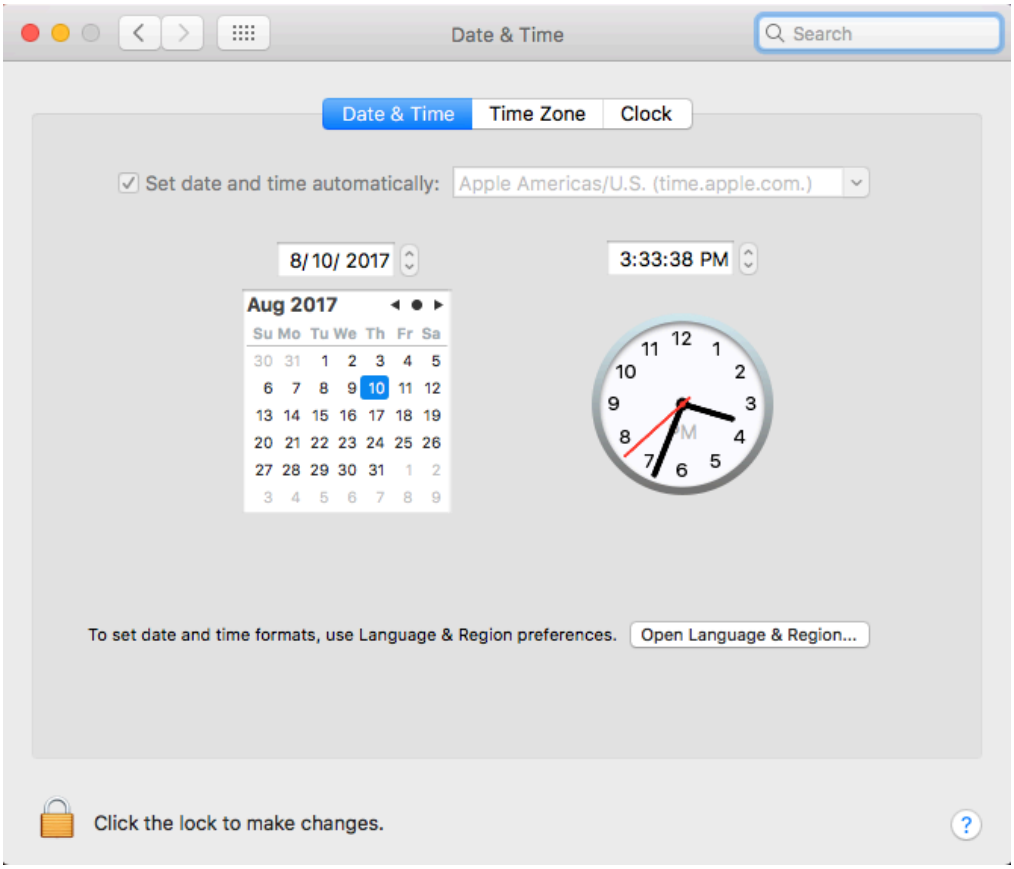
1. Ensure that the time in your system is current. If the date and time must be adjusted, do the following:
    a. For the Windows 7 and 8 platform, click the time and date in the lower right corner of the desktop. When the resulting popup window opens, select **Change Date and Time Settings...** at the bottom. Use the **Date and Time** dialog, shown below, to reset the date and time.



b. For the Windows 10 platform, right click the time and date in the lower right corner of the desktop, and select **Adjust Date and Time** from the popup menu. Use the **Time and Language** settings panel, shown below, to reset the date and time.



c. For the Mac platform, select **System Preferences...** from the Apple menu in the upper left corner of the desktop, then choose **Date and Time**. Reset the date and time accordingly.



d. For the Linux platform, open a command prompt and use the `date` command, such as the following:

```
date -s "19 APR 2017 11:14:00"
```

2. Download and install Gradle. Instructions on installation for various platforms can be found on the Gradle website.
3. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
4. Scroll to **Java SE Client Software Library** and download the **Binaries**, **Source Code** and **Samples** zip files from the **Java SE Client Software Library** table. This table describes the contents of each zip file:

Filename	Description
`iotcs-csl-javase-bin-<version>.zip`	Contains the Java SE client software library binary files.
`iotcs-csl-javase-src-<version>.zip`	Contains the Java SE client software library binary file source code.
`iotcs-csl-javase-samples-<version>.zip`	Contains the Java SE sample applications.

5. Extract the contents of the zip files. The files are saved to these directories:

Filename	Directory
iotcs-csl-javase-bin-<version>.zip	iotcs/csl/javase/bin
iotcs-csl-javase-src-<version>.zip	iotcs/csl/javase/src
iotcs-csl-javase-samples-<version>.zip	iotcs/csl/javase/samples

6. Set the `CL_HOME`, `GRADLE_HOME`, and, `JAVA_HOME` environment variables to their appropriate values. Sample values for the Linux platform are listed in the following table.

System Variable Name	Example
CL_HOME	/home/<user>/iotcs/csl/javase/
GRADLE_HOME	/opt/gradle-<version>
JAVA_HOME	/usr/java/jdk<version>

   a. For the Windows 7 platform, open the Control Panel and select System. Then, choose Advanced System Settings in the upper left of the dialog. In the System Properties dialog, choose the Advanced tab, then press the Environment Variables button. Add the new environment variables to either the user or system section.

   b. For the Windows 10 platform, open Settings and select System. Then, select the About tab on the left side, and choose Advanced System Settings in the upper right of the dialog. In the System Properties dialog, choose the Advanced tab, then press the Environment Variables button. Add the new environment variables to either the user or system section.

   c. For the Mac platform, create or edit the file `~/.bash_profile` and add one line for each entry, such as the following examples:

```
export CL_HOME=~/iotcs/csl/javase/
export GRADLE_HOME=/opt/local/share/java/gradle
export JAVA_HOME=/Library/Java/JavaVirtualMachines/
jdk1.8.0_65.jdk/Contents/Home
```

   d. For the Linux platform, create or edit the file `~/.profile` (or equivalent for the command line shell) and add one for each entry, as in the Mac example.

7. Modify the `PATH` variable to include the path to the Java and Gradle executables. Use the instructions for each platform in the previous step to access the environment variables for each operating system.

   For example, in the Linux environment, you can add the following to the existing `PATH` variable.

```
export PATH=$JAVA_HOME/bin:$GRADLE_HOME/bin:$PATH
```

8. If your computer is on a Virtual Private Network, or behind a firewall:

a. Open the `gradle.properties` file, located in the Gradle user home directory and add these lines:

```
systemProp.http.proxyHost=<your_proxy_server.com>
systemProp.http.proxyPort=<your_proxy_port>
systemProp.https.proxyHost=<your_proxy_server.com>
systemProp.https.proxyPort=<your_proxy_port>
```

b. Save your changes and close the `gradle.properties` file.

9. Start your favorite IDE and install the Gradle Support plugin.
10. Register your device, record the password, and download the provisioning file. See Registering and Activating Devices.
11. Using the web interface, create a new application named **JavaSECLApp**. See Creating a New Application.
12. Associate the humidity and temperature sensor device models with the **JavaSECLApp** application. See Assign a Device Model to a Cloud Service.
13. Add an integration named **JavaSECLApp** to the application. See Integrating Enterprise Applications with Oracle IoT Cloud Service.
14. Download the provisioning file for the integration:
    a. Log in to your Oracle Fusion Cloud IoT Intelligent Applications instance.
    b. Click the **Menu** (☰) icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.
    c. Click **Applications** and then **Browse Applications**.
    d. Click **JavaSECLApp** and then **Integration**.
    e. Select the **JavaSECLApp** integration and click the **Edit** (✎) icon.
    f. Click the **Overview** tab.

       If you are using Oracle Fusion Cloud IoT Intelligent Applications version 16 4.1 or earlier, record the **ID** and **Shared Secret** values. These values are required when you run the provisioning tool to create the trusted assets store .
    g. Enter a password in the **File Protection Password** field to encrypt the provisioning file that contains the configuration and credentials to activate your integration.
    h. Enter the password again in the **Confirm Password** field.
    i. Click **Download Provisioning File**.
    j. Click **Save File**.
    k. Browse to a directory and then click **Save**.
15. Prepare your device for the installation of the Java SE client software libraries. See Prepare an Embedded Device to Use the Java SE Client Software Library.
16. Run the Java SE client software library sample applications. See Create the Java SE Client Software Library Sample Applications.

Prepare an Embedded Device to Use the Java SE Client Software Library

An embedded device running Linux must be configured correctly and meet the minimum hardware requirements to successfully install the Java SE client software library.

1. Make sure the hardware prerequisites are met before you install the Java SE client software library on your device. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System Configurations.
2. Open a web browser and browse to Oracle Java SE Embedded Downloads page.
3. Download Oracle Java SE Embedded JDK to your device. These are the available versions:

Platform	Hardware	File
Linux/x86 desktop	x86 Linux Small Footprint – Headless	`ejdk-<version>-linux-i586.tar.gz`
ARM v6/v7 Linux — Hard Floating Point	ARM v6/v7 Linux - VFP, HardFP ABI, Little Endian 1	`ejdk-<version>-linux-armv6-vfp-hflt.tar.gz`
ARM v5/v6/v7 Linux — Soft Floating Point	ARMv5/ARMv6/ARMv7 Linux - SoftFP ABI, Little Endian 2.	`ejdk-<version>-linux-arm-sflt.tar.gz`

4. Run this command to extract the contents of the Oracle Java SE Embedded JDK file. using the example 8u101 version:

```
cd <eJDK-download-folder-location>/
tar xzvf ejdk-8u101-linux-armv6-vfp-hflt.tar.gz
```

5. Verify that the `ejdk1.8.0_101` folder was created.
6. Run this command to create the Java SE Embedded Compact 2 profile:

```
cd ejdk1.8.0_73/bin
./jrecreate.sh -d /home/<user>/ejre1.8.0_73_compact2_minimal_vm -p
compact2 --vm minimal
```

The output should appear similar to this example:

```
Building JRE using Options {
ejdk-home: /home/janeuser/ejdk1.8.0_101
        dest: /home/janeuser/ejre1.8.0_101_compact2_minimal_vm
        target: linux_arm_vfp_hflt
        vm: minimalruntime: compact2 profile
        debug: false
        keep-debug-info: false
        no-compression: false
        dry-run: false
```

```
verbose: false
    extension: []}
```

7. Install the JDK Compact 2 profile on your device. See Create Your JRE with `jrecreate` in the *Oracle Java SE Embedded Developer's Guide*.

Create the Java SE Client Software Library Sample Applications

The Java SE client software library sample applications must be created before they can run on your device.

1. Set up your Java SE development environment. See Set Up Your Development Environment to Use the Java SE Client Software Libraries.
2. Run one of these commands to create the sample applications:

   To build the samples from `iotcs-csl-javase-bin-<version>.zip`:

   ```
   cd $CL_HOME/samples
   gradle
   ```

   To build binary JAR files from `iotcs-csl-javase-src-<version>.zip`:

   ```
   cd $CL_HOME
   gradle
   cd $CL_HOME/samples
   gradle
   ```

3. Run the sample applications. See Run the Sample Java SE Directly Connected Device Applications.

Use the Provisioning Tool to Create the Truststore

If you are using Oracle Fusion Cloud IoT Intelligent Applications version 16.4.1 or earlier, use the provisioning tool to create the trusted assets truststore. The truststore contains the Oracle Fusion Cloud IoT Intelligent Applications authentication certificate and the device `ID` and `Shared Secret` values.

These are the prerequisites for this procedure:

- You have uploaded the humidity and temperature sensor device models.
- You have created the JavaSECLApp application. See Set Up Your Development Environment to Use the Java SE Client Software Libraries.
- You have created a `iotcs/csl/javase/samples` directory.
- You know the URL and port number for the Oracle Fusion Cloud IoT Intelligent Applications instance.

1. Run the following command to create a trusted assets store:

   ```
   java -cp $CL_HOME/lib/device-library.jar
   com.oracle.iot.client.impl.trust.TrustedAssetsProvisioner -serverHost
   <iotserver_url> -serverPort <port> -sharedSecret
   <device_id_shared_secret> -deviceId  <device_ID>
   ```

A trusted assets store is created in the current directory. The trusted assets store contains the Oracle Fusion Cloud IoT Intelligent Applications SSL/TLS authentication certificate, the client ID and shared secret values, and the certificates for code bundle verification.

2. Run the following command to create a trust store for the enterprise application integration:

```
java -cp $CL_HOME/lib/enterprise-library.jar
com.oracle.iot.client.impl.trust.TrustedAssetsProvisioner -
serverHost <Oracle_IoT_Cloud_Service_instance_url> -serverPort
<Oracle_IoT_Cloud_Service_instance_port> -sharedSecret
<Enterprise_app_integration_Shared_Secret> -endpointId
<Enterprise_app_integtration_ID>
```

Replace `endpointID` with the enterprise application integration ID you recorded when you were Set Up Your Development Environment to Use the Java SE Client Software Libraries. A trusted assets truststore named `Enterprise_app_integtration_ID.jks` is created . The trusted assets truststore contains the Oracle Fusion Cloud IoT Intelligent Applications SSL/TLS authentication certificate, the client ID and shared secret values.

> **Note:**
>
> Provisioning fails if the required certificate is not available in the trusted assets truststore. To correct this issue, create a `SERVER_ROOT_CERTIFICATE` environment variable with the location of the `pem` file containing the root certificate . A connectivity attempt will be made to the server using this certificate when the truststore is being created. If the connection cannot be made, the assets are created with a warning message.

Run the Sample Java SE Directly Connected Device Applications

Run the sample Java SE directly connected device applications to learn how to use the client software library APIs. The sample directly connected device applications use software to simulate temperature and humidity sensors. The sample directly connected device applications periodically send temperature, humidity, and alert messages to Oracle Fusion Cloud IoT Intelligent Applications.

About the Sample Java SE Directly Connected Device Applications

Two directly connected device sample applications are available. The first sample application is located in the `com.oracle.iot.sample` package and uses virtualization. The second sample application is located in the `com.oracle.iot.sample.ext` package and it uses a messaging API to provide direct control over the client software library.

1. Create the sample applications. See Create the Java SE Client Software Library Sample Applications.
2. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.

3. Register the device and download the provisioning file. See Registering and Activating Devices.

> **Note:**
>
> If you are using Oracle Fusion Cloud IoT Intelligent Applications version 16.4.1 or earlier, use the provisioning tool to create the trusted assets truststore. See Use the Provisioning Tool to Create the Truststore.

4. Run this command to start the `DirectlyConnectedDeviceSample` application:

```
java -cp $CL_HOME/samples/build/libs/iotcs-csl-samples.jar:$CL_HOME/lib/
device-library.jar:$CL_HOME/lib/ json-20200518.jar
com.oracle.iot.sample.DirectlyConnectedDeviceSample activation_id-
provisioning-file.conf Password123
```

Replace `activation_id` , `-provisioning-file.conf`, and `Password123` with the values you recorded when you registered your device.

To run the application with device policies, add this parameter to the command:

```
-Dcom.oracle.iot.sample.use_policy=true
```

The command to start the application with device policies:

```
java -cp -Dcom.oracle.iot.sample.use_policy=true $CL_HOME/samples/build/
libs/iotcs-csl-samples.jar:$CL_HOME/lib/device-library.jar:$CL_HOME/lib/
json-20200518.jar com.oracle.iot.sample.DirectlyConnectedDeviceSample
activation_id-provisioning-file.conf Password123
```

Output similar to the following appears:

```
Created virtual humidity sensor 6E6BD2A4-65A8-4482-869D-325D9E5291F2
Tue Feb 9 16:13:53 EST 2016 : 6E6BD2A4-65A8-4482-869D-325D9E5291F2 :
Set : "humidity"=81,"maxThreshold"=90

        Press enter to exit.

Tue Feb 9 16:13:53 EST 2016 : 6E6BD2A4-65A8-4482-869D-325D9E5291F2 :
Set : "humidity"=86
Tue Feb 9 16:13:58 EST 2016 : 6E6BD2A4-65A8-4482-869D-325D9E5291F2 :
Set : "humidity"=89
Tue Feb 9 16:14:03 EST 2016 : 6E6BD2A4-65A8-4482-869D-325D9E5291F2 :
Set : "humidity"=91
Tue Feb 9 16:14:03 EST 2016 : 6E6BD2A4-65A8-4482-869D-325D9E5291F2 :
Alert : "humidity"=91 (tooHumidAlert)
Tue Feb 9 16:14:09 EST 2016 : 6E6BD2A4-65A8-4482-869D-325D9E5291F2 :
Set : "humidity"=89
Tue Feb 9 16:14:14 EST 2016 : 6E6BD2A4-65A8-4482-869D-325D9E5291F2 :
Set : "humidity"=86
```

Run the Sample Java SE Gateway Application

Run the sample Java SE gateway application to learn how to use the client software library APIs. The sample Java SE gateway application simulates a gateway that polls humidity and temperature sensors and sends sensor data to the Oracle Fusion Cloud IoT Intelligent Applications instance.

1. Create the sample applications. See Create the Java SE Client Software Library Sample Applications.
2. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
3. Register the gateway device and download the provisioning file.

   Do not reuse the device from the *Running the Sample Java SE Directly Connected Device Applications* procedure. This procedure requires a device with indirect activation capability.

   > **Note:**
   >
   > If you are using Oracle Fusion Cloud IoT Intelligent Applications version 16 4.1 or earlier, use the provisioning tool to create the trusted assets truststore. See Use the Provisioning Tool to Create the Truststore.

4. Run this command to start the `GatewayDeviceSample` application:

```
 java -cp $CL_HOME/samples/build/libs/iotcs-csl-
samples.jar:$CL_HOME/lib/device-library.jar:$CL_HOME/lib/
json-20200518.jar com.oracle.iot.sample.GatewaydDeviceSample MY-GW-
SAMPLE-provisioning-file.conf Password123
```

   To run the application with device policies, add this parameter to the command:

```
-Dcom.oracle.iot.sample.use_policy=true
```

   The command to start the application with device policies:

```
 java -cp -Dcom.oracle.iot.sample.use_policy=true $CL_HOME/samples/
build/libs/iotcs-csl-samples.jar:$CL_HOME/lib/device-
library.jar:$CL_HOME/lib/json-20200518.jar
com.oracle.iot.sample.GatewaydDeviceSample MY-GW-SAMPLE-
provisioning-file.conf Password123
```

   Output similar to the following appears:

```
Creating the gateway instance...
Created virtual temperature sensor 0-HA
Created virtual humidity sensor 0-HE

        Press enter to exit.

Tue Feb 9 16:17:53 EST 2016 : 0-HE : Set : "humidity"=81
```

```
Tue Feb 9 16:17:52 EST 2016 : 0-HA : Set :
"power"=true,"temp"=58.5,"unit"="°C,"minTemp"=58.5,"maxTemp"=58.5,"minThres
hold"=0,"maxThreshold"=65
Tue Feb 9 16:17:58 EST 2016 : 0-HE : Set : "humidity"=86
Tue Feb 9 16:17:58 EST 2016 : 0-HA : Set : "temp"=62.25,"maxTemp"=62.25
Tue Feb 9 16:18:03 EST 2016 : 0-HE : Set : "humidity"=89
Tue Feb 9 16:18:03 EST 2016 : 0-HA : Set : "temp"=64.99,"maxTemp"=64.99
Tue Feb 9 16:18:08 EST 2016 : 0-HE : Set : "humidity"=91
Tue Feb 9 16:18:08 EST 2016 : 0-HE : Alert : "humidity"=91 (tooHumidAlert)
Tue Feb 9 16:18:08 EST 2016 : 0-HA : Set : "temp"=66.0,"maxTemp"=66.0
Tue Feb 9 16:18:08 EST 2016 : 0-HA : Alert :
"temp"=66.0,"unit"="°C,"maxThreshold"=65
Tue Feb 9 16:18:13 EST 2016 : 0-HE : Set : "humidity"=89
Tue Feb 9 16:18:13 EST 2016 : 0-HA : Set : "temp"=64.99
```

Run the Sample Java SE Gateway Application Using Apache Felix

Run the sample Java SE gateway application using Apache Felix to learn how to use the client software library APIs. The sample Java SE gateway application simulates a gateway that polls humidity and temperature sensors, and sends sensor data to the Oracle Fusion Cloud IoT Intelligent Applications instance.

1. Set up your Java SE development environment. See Set Up Your Development Environment to Use the Java SE Client Software Libraries.
2. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
3. Download and extract the content of the latest version of the Apache Felix framework.
4. Copy the following definition of the gateway device application and create a file named `SampleActivator.java` in the `$CL_HOME/samples/src/main/java/com/oracle/iot/sample` directory.

```
/*
* Copyright (c) 2015, 2017, Oracle and/or its affiliates. All rights
reserved.
*
* This software is dual-licensed to you under the MIT License (MIT) and
* the Universal Permissive License (UPL). See the LICENSE file in the root
* directory for license terms. You may choose either license, or both.
*/

package com.oracle.iot.sample;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;

/**
 * Activator class for OSGi platform. Refer to the on-line, Oracle IoT
Cloud Service
 * Java Client Library documentation for details on how to run a client
library
 * sample from an OSGi bundle.
 */
```

```
public class SampleActivator implements BundleActivator
{
    /**
     * Implements BundleActivator.start().
     *
     * Note: expects following system properties to be set:
'sample.name', 'sample.args'
     *
     * For example: {@code -Dsample.name=GatewayDeviceSample -
Dsample.args="activationId.jks password"}
     * @param context the framework context for the bundle.
     **/
    public void start(BundleContext context) throws Exception
    {
        Thread executeSample = new Thread(new
ExecuteSample(context));
        executeSample.setDaemon(true);
        executeSample.start();
    }

   /**
     * Implements BundleActivator.stop().
     * @param context the framework context for the bundle.
     **/
    public void stop(BundleContext context) throws Exception
    {
        System.err.println("SampleActivator.stop called");
        String sampleName = System.getProperty("sample.name",
"GatewayDeviceSample");
        String classname = "com.oracle.iot.sample."+sampleName;
        Class<?> cls = Class.forName(classname);
        java.lang.reflect.Field f = cls.getDeclaredField("exiting");
        f.set(null, true);
    }
private class ExecuteSample implements Runnable {
        private BundleContext context;
        public ExecuteSample(BundleContext context) { this.context
= context; }
        @Override
        public void run() {
            String sampleName = System.getProperty("sample.name",
"GatewayDeviceSample");
            String classname = "com.oracle.iot.sample."+sampleName;
            try {
                Class<?> cls = Class.forName(classname);
                java.lang.reflect.Field f =
cls.getDeclaredField("isUnderFramework");
                f.set(null, true);

                f = cls.getDeclaredField("exiting");
                f.set(null, false);

                String[] arguments =
System.getProperty("sample.args","").split("\\s+");
```

```
                @SuppressWarnings("unchecked")
                final java.lang.reflect.Method meth =
cls.getMethod("main", String[].class);
                meth.invoke(null, (Object) arguments); // static method
doesn't have an instance

            } catch(Exception e) {
                try {
                    context.getBundle().stop();
                } catch(BundleException be) {
                }
            }
        }
    }
}
```

5. Run the following command to create the `build` directory:

```
cd $CL_HOME/samples
mkdir -p build/classes
```

6. Run the following command to build the classes:

```
javac -d ./build/classes -cp Apache_Felix_directory/bin/felix.jar:../lib/
device-library.jar
src/main/java/com/oracle/iot/sample/GatewayDeviceSample.java
src/main/java/com/oracle/iot/sample/SampleActivator.java
src/main/java/com/oracle/iot/sample/TemperatureSensor.java
src/main/java/com/oracle/iot/sample/HumiditySensor.java
```

   The output classes appear in the `$CL_HOME/samples/build/classes/` directory.

7. Create the `MANIFEST.MF` file in the`$CL_HOME/samples` directory using the values in the following table:

Sample Name	MANIFEST.MF

`GatewayDeviceSample` and `DirectlyConnectedDeviceSample`	`Bundle-ManifestVersion: 2`
	`Bundle-Name: GatewayDevice Sample`
	`Bundle-SymbolicName: com.oracle.iot.sample`
	`Bundle-Version: 1.0.0`
	`Bundle-Activator: com.oracle.iot.sample.SampleActivator`
	`Import-Package: org.osgi.framework,oracle.iot.client.device,oracle.iot.client`

Sample Name	MANIFEST.MF
EnterpriseClientSample	Bundle-ManifestVersion: 2 Bundle-Name: GatewayDevice Sample Bundle-SymbolicName: com.oracle.iot.sample Bundle-Version: 1.0.0 Bundle-Activator: com.oracle.iot.sample.SampleActivat or Import-Package: org.osgi.framework,oracle.iot.clien t.enterprise,oracle.iot.client
ext.DirectlyConnectedDeviceSample and ext.GatewayDeviceSample	Bundle-ManifestVersion: 2 Bundle-Name: GatewayDevice Sample Bundle-SymbolicName: com.oracle.iot.sample Bundle-Version: 1.0.0 Bundle-Activator: com.oracle.iot.sample.SampleActivat or Import-Package: org.osgi.framework,org.json,com.ora cle.iot.client.device,com.oracle.io t.client.message,com.oracle.iot.cli ent.device.util

> **Note:**
>
> Make sure there is a line break after the last line of the `MANIFEST.MF` file.

8. Run this command to create the bundle jar file:

```
jar -cmf MANIFEST.MF osgi-sample.jar -C ./build/classes com
```

9. Register the gateway device and download the provisioning file.
10. Change directories to `Apache_Felix_directory`.
11. Run the following command to start Apache Felix, and replace the *sample_name* and the *sample_args* using the values in the table:

```
java -Dsample.name=sample_name -Dsample.args="sample_args" -jar
FELIX_PATH/bin/felix.jar
```

To run the application with device policies, add this parameter to the command:

```
-Dcom.oracle.iot.sample.use_policy=true
```

Run this command to start the application with device policies:

```
java -Dcom.oracle.iot.sample.use_policy=true -Dsample.name=sample_name -
Dsample.args="sample_args" -jar FELIX_PATH/bin/felix.jar
```

sample_name	sample_args
DirectlyConnectedDeviceSample	*provisioning_file_name provisioning_file_password*
GatewayDeviceSample	*provisioning_file_name provisioning_file_passwordprovisioning_file_name provisioning_file_password temperature_sensor_endpointID humidity_sensor_endpointID*
EnterpriseClientSample	*provisioning_file_name provisioning_file_passwordprovisioning_file_name provisioning_file_password deviceID* [,*deviceID*]*provisioning_file_name provisioning_file_password deviceID*][reset | on | off] java com.oracle.iot.sample.EnterpriseClientSample java com.oracle.iot.sample.EnterpriseClientSample *provisioning_file_name provisioning_file_password deviceID maxThreshold minThreshold*
ext.DirectlyConnectedDeviceSample	*provisioning_file_name provisioning_file_password*
ext.GatewayDeviceSample	*provisioning_file_name provisioning_file_passwordprovisioning_file_name provisioning_file_password temperature_sensor_endpointID humidity_sensor_endpointID*

12. Install the `device-library.jar`, `json-20160212.jar`, `osgi-sample.jar` files, and then start your bundle using the assigned ID.

    The output should be similar to the following example:

```
Welcome to Apache Felix Gogo
g! install CL_HOME/lib/device-library.jar
Bundle ID: 5
g! install CL_HOME/lib/json-20200518.jar
Bundle ID: 6
g! install CL_HOME/samples/osgi-sample.jar
Bundle ID: 7
g! start 7
Starting to listen for service events.
Creating the gateway instance...
Created virtural temperature sensor 0-2UJQ
Created virtural humidity sensor 0-2YJQ
    Press enter to exit.
Tue Aug 02 17:18:18 EDT 2016 : 0-2YJQ : Set :
"humidity"=81,"maxThreshold"=90
Tue Aug 02 17:18:17 EDT 2016 : 0-2UJQ : Set :
"power"=true,"temp"=58.5,"unit"="°C,"minTemp"=58.5,"maxTemp"=58.5,"minThres
hold"=0,"maxThreshold"=65
```

> **Note:**
>
> The Bundle ID values may vary.

Run the Sample Java SE Enterprise Applications

Run the Java SE sample enterprise applications to learn how to use the client software library APIs. The sample enterprise application reads humidity and temperature values of directly connected or gateway devices. The sample enterprise applications can also change device attributes by sending commands through Oracle Fusion Cloud IoT Intelligent Applications.

1. Create the sample applications. See Create the Java SE Client Software Library Sample Applications.
2. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
3. Register the device and download the provisioning file.

   > **Note:**
   >
   > If you are using Oracle Fusion Cloud IoT Intelligent Applications version 16 4.1 or earlier, use the provisioning tool to create the trusted assets truststore. See Use the Provisioning Tool to Create the Truststore.

4. Run the `DirectlyConnectedDevice` or the `GatewayDevice` sample applications. See Run the Sample Java SE Directly Connected Device Applications or Run the Sample Java SE Gateway Application.
5. Run this command to return a list of device IDs that are associated with the humidity sensor:

   ```
   java -cp $CL_HOME/samples/build/libs/iotcs-csl-
   samples.jar:$CL_HOME/lib/enterprise-library.jar:$CL_HOME/lib/
   json-20200518.jar com.oracle.iot.sample.EnterpriseClientSample
   app_integration_ID-provisioning-file.conf Password123
   ```

   Replace `app_integration_ID` and `-provisioning-file.conf Password123` with the values that you recorded when adding the integration and registering the device.

   Output similar to the following appears:

   ```
   0-HA [Temperature Sensor]
   0-BQ [Humidity Sensor]
   0-HE [Humidity Sensor]
   ```

6. Record the device ID, **0-BQ** that you used to run the `DirectlyConnectedDeviceDeviceSample` sample and the device IDs **0-HE** and **0-HA** of the indirectly connected devices that you used to run the `GatewayDeviceSample` .

7. Run this command to monitor the humidity and temperature sensors:

```
java -cp $CL_HOME/samples/build/libs/iotcs-csl-samples.jar:$CL_HOME/lib/
enterprise-library.jar:$CL_HOME/lib/ json-20200518.jar
com.oracle.iot.sample.EnterpriseClientSample app_integration_ID-
provisioning-file.conf Password123 0-HA,0-HE,0-BQ
```

> **Note:**
>
> Replace `app_integration_ID-provisioning-file.conf` with the name of the enterprise integration provisioning file and `app_integration2_ID-provisioning-file.conf` with the integration provisioning file for the control feature of the `EnterpriseClientSample` application. Replace `Password123` with the password used to secure the provisioning files.

Output similar to the following appears:

```
        Press enter to exit.

Tue Feb 9 16:34:09 EST 2016 : 0-HE : onChange : "humidity"=89
Tue Feb 9 16:34:13 EST 2016 : 0-BQ : onChange : "humidity"=89
Tue Feb 9 16:34:16 EST 2016 : 0-HE : onChange : "humidity"=91
Tue Feb 9 16:34:16 EST 2016 : 0-BQ : onChange : "humidity"=86
Tue Feb 9 16:34:16 EST 2016 : 0-HI : onChange : "temp"=66.0
Tue Feb 9 16:34:16 EST 2016 : 0-HI : onAlert :
"temp"=66.0,"unit"="C,"maxThreshold"=65.0 (tooHotAlert)
Tue Feb 9 16:34:16 EST 2016 : 0-HE : onAlert : "humidity"=91
(tooHumidAlert)
Tue Feb 9 16:34:19 EST 2016 : 0-HE : onChange : "humidity"=89
Tue Feb 9 16:34:19 EST 2016 : 0-HI : onChange : "temp"=64.99
Tue Feb 9 16:34:23 EST 2016 : 0-BQ : onChange : "humidity"=81
Tue Feb 9 16:34:26 EST 2016 : 0-HE : onChange : "humidity"=86
Tue Feb 9 16:34:26 EST 2016 : 0-BQ : onChange : "humidity"=76
Tue Feb 9 16:34:26 EST 2016 : 0-HI : onChange : "temp"=62.25
Tue Feb 9 16:34:29 EST 2016 : 0-HE : onChange : "humidity"=81
Tue Feb 9 16:34:29 EST 2016 : 0-HI : onChange : "temp"=58.5
```

8. Run this command to set the maximum humidity threshold of the `DirectlyConnectedDeviceSample` device:

```
java -cp $CL_HOME/samples/build/libs/iotcs-csl-samples.jar:$CL_HOME/lib/
enterprise-library.jar:$CL_HOME/lib/ json-20200518.jar
com.oracle.iot.sample.EnterpriseClientSample app_integration2_ID-
provisioning-file.conf Password123 0-BQ 67
```

This message appears:

```
Tue Feb 9 18:24:56 EST 2016 : 0-BQ : Set : "maxThreshold"=67 ..
... [Humidity readings] ...
Done.
```

Output similar to the following appears on the device console:

```
Tue Feb 9 18:25:00 EST 2016 : 0-BQ : Set : "humidity"=73
Tue Feb 9 18:25:00 EST 2016 : 0-BQ : onChange : "maxThreshold"=67
Tue Feb 9 18:25:05 EST 2016 : 0-BQ : Set : "humidity"=57
Tue Feb 9 18:25:10 EST 2016 : 0-BQ : Set : "humidity"=60
Tue Feb 9 18:25:15 EST 2016 : 0-BQ : Set : "humidity"=64
Tue Feb 9 18:25:20 EST 2016 : 0-BQ : Set : "humidity"=67
Tue Feb 9 18:25:20 EST 2016 : 0-BQ : Alert : "humidity"=67
(tooHumidAlert)
Tue Feb 9 18:25:25 EST 2016 : 0-BQ : Set : "humidity"=68
Tue Feb 9 18:25:30 EST 2016 : 0-BQ : Set : "humidity"=67
```

9. Run this command to reset the `EnterpriseClientSample` device temperature sensor:

```
java -cp $CL_HOME/samples/build/libs/iotcs-csl-
samples.jar:$CL_HOME/lib/enterprise-library.jar:$CL_HOME/lib/
json-20200518.jar com.oracle.iot.sample.EnterpriseClientSample
app_integration2_ID-provisioning-file.conf Password123 0-HA reset
```

Output similar to the following appears on the device console:

```
...
Tue Feb 9 18:18:06 EST 2016 : 0-HA : Call : reset
Tue Feb 9 18:18:10 EST 2016 : 0-HA : Set :
"temp"=58.5,"minTemp"=58.5,"maxTemp"
...
```

10. Run this command to turn the `EnterpriseClientSample` device power on or off:

```
java -cp $CL_HOME/samples/build/libs/iotcs-csl-
samples.jar:$CL_HOME/lib/enterprise-library.jar:$CL_HOME/lib/
json-20200518.jar com.oracle.iot.sample.EnterpriseClientSample
app_integration2_ID-provisioning-file.conf Password1230-HA off  (or
on to turn on sensor, in this case it is the temperature sensor)
```

Temperature data stops displaying on the device console:

```
...
Tue Feb 9 18:21:38 EST 2016 : 0-HA : Call : "power"=false
...
```

11. Run this command to set the maximum and minimum threshold values for the `EnterpriseClientSample` device:

```
java -cp $CL_HOME/samples/build/libs/iotcs-csl-
samples.jar:$CL_HOME/lib/enterprise-library.jar:$CL_HOME/lib/
json-20200518.jar com.oracle.iot.sample.EnterpriseClientSample
app_integration2_ID-provisioning-file.conf Password123 0-HA 65 -10
```

Output similar to the following appears on the device console:

```
...
Tue Feb 9 18:28:15 EST 2016 : 0-HA : Alert :
"temp"=66.0,"unit"="°C,"maxThreshold"=65

Tue Feb 9 18:28:15 EST 2016 : 0-HA : onChange :
"minThreshold"=-10,"maxThreshold"=65

...
```

Output similar to the following appears on the `EnterpriseClientSample` console:

```
Tue Feb 9 18:28:14 EST 2016 : 0-HA : Set :
"maxThreshold"=65,"minThreshold"=-10 ..
Tue Feb 9 18:28:14 EST 2016 : 0-HA : onChange : "startTime"=Tue Feb 9
18:27:55 EST
2015,"maxTemp"=64.99,"unit"="°C,"minThreshold"=0,"maxThreshold"=65,"temp"=6
4.99,"minTemp"=58.49
Tue Feb 9 18:28:14 EST 2016 : 0-HA : onAlert :
"temp"=66.0,"unit"="°C,"maxThreshold"=65.0 (tooHotAlert)
Tue Feb 9 18:28:14 EST 2016 : 0-HA : onAlert :
"temp"=66.0,"unit"="°C,"maxThreshold"=65.0 (tooHotAlert)
Tue Feb 9 18:28:14 EST 2016 : 0-HA : onAlert :
"temp"=66.0,"unit"="°C,"maxThreshold"=65.0 (tooHotAlert)
Tue Feb 9 18:28:14 EST 2016 : 0-HA : onAlert :
"temp"=66.0,"unit"="°C,"maxThreshold"=65.0 (tooHotAlert)
Tue Feb 9 18:28:14 EST 2016 : 0-HA : onAlert :
"temp"=66.0,"unit"="°C,"maxThreshold"=65.0 (tooHotAlert)
Done.
```

12. Run this command to start the `advanced/.../DirectlyConnectedDeviceSample` application:

```
java -cp $CL_HOME/samples/advanced/build/libs/iotcs-samples-
advanced.jar$CL_HOME/lib/device-library.jar:$CL_HOME/lib/
json-20200518.jar com.oracle.iot.sample.DirectlyConnectedDeviceSample
activation_ID-provisioning-file.conf Password123
```

Output similar to the following appears:

```
Created virtual humidity sensor 0-BQ
Tue Feb 9 18:34:44 EST 2015 : 0-BQ : Set : "humidity"=81,"maxThreshold"=90

        Press enter to exit.

Tue Feb 9 18:34:44 EST 2015 : 0-BQ : Set : "humidity"=86
Tue Feb 9 18:34:49 EST 2015 : 0-BQ : Set : "humidity"=89
Tue Feb 9 18:34:54 EST 2015 : 0-BQ : Set : "humidity"=91
Tue Feb 9 18:34:54 EST 2015 : 0-BQ : Alert : "humidity"=91 (tooHumidAlert)
Tue Feb 9 18:34:59 EST 2015 : 0-BQ : Set : "humidity"=89
Tue Feb 9 18:35:04 EST 2015 : 0-BQ : Set : "humidity"=86
```

13. Run this command to start the `advanced/.../GatewayDeviceSample` application:

```
java -cp $CL_HOME/samples/advanced/build/libs/iotcs-samples-
advanced.jar$CL_HOME/lib/device-library.jar:$CL_HOME/lib/
json-20200518.jar com.oracle.iot.sample.GatewayDeviceSample MY-GW-
SAMPLE-provisioning-file.conf Password123
```

Output similar to the following appears:

```
Creating the gateway instance...

Created virtural temperature sensor 0-HY
Created virtural humidity sensor 0-H4

        Press enter to exit.

Tue Feb 9 18:32:34 EST 2015 : 0-H4 : Set : "humidity"=81
Tue Feb 9 18:32:33 EST 2015 : 0-HY : Set :
"power"=true,"temp"=58.5,"unit"="°C,"minTemp"=58.5,"maxTemp"=58.5,"mi
nThreshold"=0,"maxThreshold"=65
Tue Feb 9 18:32:39 EST 2015 : 0-H4 : Set : "humidity"=86
Tue Feb 9 18:32:39 EST 2015 : 0-HY : Set :
"temp"=62.25,"maxTemp"=62.25
Tue Feb 9 18:32:44 EST 2015 : 0-H4 : Set : "humidity"=89
Tue Feb 9 18:32:44 EST 2015 : 0-HY : Set :
"temp"=64.99,"maxTemp"=64.99
```

14. Run the following command to start the `advanced/.../GatewayDeviceSample` application:

```
java -cp $CL_HOME/samples/build/libs/iotcs-csl-
samples.jar:$CL_HOME/lib/device-library.jar:$CL_HOME/lib/
json-20200518.jar com.oracle.iot.sample.GatewayDeviceSample MY-GW-
SAMPLE- provisioning-file.conf Password123 0-HY 0-H4
```

Output similar to the following appears:

```
Creating the gateway instance...

Created virtural temperature sensor 0-HY
Created virtural humidity sensor 0-H4

        Press enter to exit.

Tue Feb 9 18:32:34 EST 2015 : 0-H4 : Set : "humidity"=81
Tue Feb 9 18:32:33 EST 2015 : 0-HY : Set :
"power"=true,"temp"=58.5,"unit"="°C,"minTemp"=58.5,"maxTemp"=58.5,"mi
nThreshold"=0,"maxThreshold"=65
Tue Feb 9 18:32:39 EST 2015 : 0-H4 : Set : "humidity"=86
Tue Feb 9 18:32:39 EST 2015 : 0-HY : Set :
"temp"=62.25,"maxTemp"=62.25
Tue Feb 9 18:32:44 EST 2015 : 0-H4 : Set : "humidity"=89
```

```
Tue Feb 9 18:32:44 EST 2015 : 0-HY : Set : "temp"=64.99,"maxTemp"=64.99
```

Build the Java SE Client Software Libraries

Build the client software libraries from the downloaded source files to customize the functionality of the libraries and change the size of the library JAR files.

1. Download and install Gradle. Versions 2.2.1 to 2.13 are supported.
2. Download and unzip `iotcs-csl-javase-src-<VERSION>.zip`.
3. Open a command prompt and use the `cd` command to browse to `iotcs/csl/javase`.
4. If your computer is on a Virtual Private Network, or behind a firewall:
   a. Open the `gradle.properties` file, located in the Gradle user home directory and add these lines:

      ```
      systemProp.http.proxyHost=<your_proxy_server.com>
      systemProp.http.proxyPort=<your_proxy_port>
      systemProp.https.proxyHost=<your_proxy_server.com>
      systemProp.https.proxyPort=<your_proxy_port>
      ```

      The default value for the Gradle user home directory is `USER_HOME/.gradle`. To use a different directory, set the `GRADLE_USER_HOME` environment variable.

   b. Save your changes and close the `gradle.properties` file.
5. Run one of these commands to build the Client Software Libraries and documentation:

Command	Description
`gradle`	Builds the `device-library.jar` and `enterprise-library.jar` library files and places them in the `build/libs` folder.
`gradle deviceClientJar`	Compiles the code specific to a device client and creates `build/libs/device-library.jar`
`gradle enterpriseClientJar`	Compiles the code specific to an enterprise client and creates `build/libs/enterprise-library.jar`
`gradle –PWITH_VIRTUALIZATION=false`	Compiles the code specific to a device client without virtualization support. This reduces the size of the `device-library.jar`
`gradle -PWITH_ENUMERATION=false`	Compiles the code specific to an enterprise client without resource and message enumeration support. This reduces the size of the `enterprise-library.jar` file.
`gradle doc`	Generates the javadoc files and places them in the `build/docs` folder.

Use the JavaScript Client Software Libraries

Two JavaScript client software libraries are available. To run sample applications, download the Samples library. To create a client software library for your specific development environment, download the Source Code library.

Topics

Set Up Your Development Environment to Use the JavaScript Client Software Libraries

Before you can develop applications that let your devices to communicate with Oracle Fusion Cloud IoT Intelligent Applications, you first download and extract the JavaScript client software libraries.

1. Register your device, record the password, and download the provisioning file.
2. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
3. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
4. Scroll to **JavaScript Client Software Library** and download the **Binaries**, **Source Code**, and **Samples** zip files.
5. Extract the contents of the zip files. The files are saved to these directories:

Filename	Directory
`iotcs-csl-js-bin-<version>.zip`	`iotcs/csl/js/bin`
`iotcs-csl-js-src-<version>.zip`	`iotcs/csl/js/src`
`iotcs-csl-js-samples-<version>.zip`	`iotcs/csl/js/samples`

6. Create new applications named **JavaScriptCLapp** and **JavaScriptELapp** and then record the application IDs for each application. See Creating a New Application.
7. Associate the humidity and temperature sensor device models with the **JavaScriptCLapp** and **JavaScriptELapp** applications.
8. Add integrations named **JavaScriptCLapp** and **JavaScriptELapp** to the applications. See Integrating Enterprise Applications with Oracle IoT Cloud Service.

9. Download the provisioning file for the **JavaScriptCLapp** integration:
    a. Log in to your Oracle Fusion Cloud IoT Intelligent Applications instance.
    b. Click the **Menu** ( ) icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.
    c. Click **Applications** and then **Browse Applications**.
    d. Click **JavaScriptCLapp** and then **Integration**.
    e. Select the **JavaScriptCLapp** integration and click the **Edit** ( ) icon.
    f. Click the **Overview** tab.

       If you are using Oracle Fusion Cloud IoT Intelligent Applications version 16 4.1 or earlier, record the **ID** and **Shared Secret** values. These values are required when you run the provisioning tool to create the trusted assets store.
    g. Enter a password in the **File Protection Password** field to encrypt the provisioning file that contains the configuration and credentials to activate your integration.
    h. Enter the password again in the **Confirm Password** field.
    i. Click **Download Provisioning File**.
    j. Click **Save File**.
    k. Browse to a directory and then click **Save**.
10. Repeat the previous step to download the provisioning file for the **JavaScriptELapp** integration.
11. Prepare your device for the installation of the JavaScript client software libraries. See Prepare Your Device to Use the JavaScript Client Software Library.
12. Run the sample applications.

Prepare Your Device to Use the JavaScript Client Software Library

Ensure that the hardware and software prerequisites are met prior to installing the Oracle IoT Cloud Service Client Library Software for the JavaScript platform on your device. You need to configure your device with the supported operating system and the latest version of the required software.

1. Make sure the hardware prerequisites are met before you install the JavaScript client software library on your device. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System Configurations.
2. Install npm and Node.js.
3. Download and install gradle.
4. Run this command to install the debug module: `npm install -g debug`.
5. Run this command to install the sqlite3 module: `npm install -g sqlite3`.
6. Run this command to install the node-forge module: `npm install -g node-forge`.
7. To optionally build documentation from sources, run this command: `npm install -g jsdoc`.
8. Run this command to set the `NODE_PATH` environment variable: `SET NODE_PATH "%C:\username\AppData\Roaming\npm\node_modules%"`.

9. Run this command to move to the `node-forge` directory: `cd C:\Users\username\AppData\Roaming\npm\node_modules\node-forge`.

   If you have trouble running the `npm run bundle` command in a Windows environment, run this command: `node node_modules\requirejs\bin\r.js -o minify.js optimize=none out=js/forge.bundle.js`

10. Run this command to move to the `js` directory: `cd C:\Users\username\Documents\iot\csl\js`.
11. Create a new folder named **External**.
12. Copy the file `forge.bundle.js` from `C:\Users\username\AppData\Roaming\npm\node_modules\node-forge\js` to `C:\Users\username\Documents\iot\csl\js\external` directory.
13. Copy the provisioning file that you downloaded in Set Up Your Development Environment to Use the JavaScript Client Software Libraries to the `C:\Users\username\Documents\iot\csl\js\external` directory.
14. Open a text editor and the open the `EnterpriseClient.html` file located at this path: `C:\Users\username\Documents\iot\csl\js\samples`.
15. Locate the `iotcs.oracle.iot.tam.storePassword` value and enter the password used to protect the provisioning file you downloaded when you registered the device.
16. Locate the `src` value and make sure the value is `"../modules/enterprise-library.web.js"`.
17. Save and close the `EnterpriseClient.html` file.
18. Continue with either Run the Sample JavaScript Directly Connected Device Applications or Run the Sample JavaScript Enterprise Applications.

Run the Sample JavaScript Directly Connected Device Applications

Run the sample JavaScript directly connected device applications to learn how to use the client software library APIs. The sample directly connected device applications use software to simulate temperature and humidity sensors. The sample directly connected device applications periodically send temperature, humidity, and alert messages to Oracle Fusion Cloud IoT Intelligent Applications.

1. Download and install Node.js with npm. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System .
2. Open a command prompt and run this command to move to the `samples` directory: `cd C:\Users\username\Documents\iot\csl\js\samples`.
3. Run this command to activate a registered device:

   ```
   run-device-node-sample.bat [sample js file] [[id].conf file path]
   [File Protection Password used for the .conf file]
   ```

4. Confirm the device is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications:
   a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

The URL format to access the IoT management console is: `<iot instance name>.<domain name>/ui`

For example: `https://myiotcs.mydomain.oraclecloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

b. Click the **Menu** () icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.

c. Click **Devices**.

d. Click **Alerts and Messages**.

e. Click the **Messages** tab and confirm there are incoming messages.

Run the Sample JavaScript Gateway Application

Run the sample JavaScript gateway application to learn how to use the client software library APIs. The sample JavaScript gateway application simulates a gateway that polls humidity and temperature sensors and sends sensor data to the Oracle Fusion Cloud IoT Intelligent Applications instance.

1. Download and install Node.js with npm. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System .
2. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
3. Register the gateway device and download the provisioning file.

   Do not reuse the device from the *Running the Sample Java SE Directly Connected Device Applications* procedure. This procedure requires a device with indirect activation capability.

   > **Note:**
   >
   > If you are using Oracle Fusion Cloud IoT Intelligent Applications version 16 4.1 or earlier, use the provisioning tool to create the trusted assets truststore. See Use the Provisioning Tool to Create the Truststore.

4. Run this command to move to the `samples` directory:

   ```
   cd C:\Users\username\Documents\iot\csl\js\samples
   ```

5. Run this command to start the GatewayDeviceSample application:

   ```
   run-device-node-sample.bat GatewayDeviceSample.js [[id].conf file path]
   [File Protection Password used for the .conf file]
   ```

6. Confirm the device is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications:

   a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

      The URL format to access the IoT management console is: `<iot instance name>.<domain name>/ui`

For example: `https://myiotcs.mydomain.oracleiotcloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

b. Click the **Menu** () icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.

c. Click **Devices**.

d. Click **Alerts and Messages**.

e. Click the **Messages** tab and confirm there are incoming messages.

Run the Sample JavaScript Enterprise Applications

Run the JavaScript sample enterprise applications to learn how to use the client software library APIs. The sample enterprise application reads humidity and temperature values of directly connected or gateway devices. The sample enterprise applications can also change device attributes by sending commands through Oracle Fusion Cloud IoT Intelligent Applications.

1. Run this command to install the HTTP server: `npm install -g http-server`.

   If you do not want to install the `http-server node module`, you can install an alternate HTTP server.

2. Host the HTTP server in this directory: `C:\Users\username\Documents\iot\csl\js` .

3. Open a command prompt and run this command to move to the `js` directory: `cd C:\Users\username\Documents\iot\csl\js`.

4. Run this command to run the HTTP server: `js http-server`.

5. Run the enterprise application:

   a. Open a web browser and enter this URL in the address bar: `http://127.0.0.1:8080/samples/EnterpriseClient.html?trustStore=/external/[id].conf`.

   b. Press **Enter**.

   c. Select your JavaScript application.

   d. Select the device types to monitor.

   e. Select devices to monitor.

   f. Set the minimum and maximum temperature and humidity thresholds, reset the device, or turn it on or off.

6. Confirm the device is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications:

   a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

   The URL format to access the IoT management console is: *<iot instance name>.<domain name>*`/ui`

   For example: `https://myiotcs.mydomain.oracleiotcloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

   b. Click the **Menu** () icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.

c. Click **Devices**.

d. Click **Alerts and Messages**.

e. Click the **Messages** tab and confirm there are incoming messages.

Use the Android Client Software Libraries

Three Android client software libraries are available. To develop applications, download the Binaries library. To run sample applications, download the Samples library. To create a client software library for your specific development environment, download and build the Source Code library.

Topics

Set Up Your Development Environment to Use the Android Client Software Libraries

Before you can develop applications that let your devices communicate with Oracle Fusion Cloud IoT Intelligent Applications, you first download, install, and configure the Android client software libraries.

1. Register your device, record the password, and download the provisioning file.
2. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
3. Download and install Gradle. Version 4.*x* is supported.
4. Download and install Java SE Development Kit (JDK) 8.0 or later.
5. Download and install Android Studio.
6. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
7. Scroll to **Android Client Software Library** and download the **Binaries**, **Source Code** and **Samples** zip files from the **Android Client Software Library** table.
8. Extract the contents of the zip files. The files are saved to these directories:

Filename	Directory
`iotcs-csl-android-bin-<version>.zip`	`iotcs/csl/<version>/android/bin`
`iotcs-csl-android-src-<version>.zip`	`iotcs/csl/<version>/android/src`
`iotcs-csl-android-samples-<version>.zip`	`iotcs/csl/<version>/android/samples`

9. Create a new application named **AndroidELAppln**. See Creating a New Application.
10. Associate the humidity and temperature sensor device models with the **AndroidELAppln** application.
11. Add an integration named **EnterpriseClientSample** to the application. See Integrating Enterprise Applications with Oracle IoT Cloud Service.
12. Download the provisioning file for the **EnterpriseClientSample** integration:
    a. Log in to your Oracle Fusion Cloud IoT Intelligent Applications instance.
    b. Click the **Menu** ( ) icon next to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.
    c. Click **Applications** and then **Browse Applications**.
    d. Click **AndroidELAppln** and then **Integration**.
    e. Select the **EnterpriseClientSample** integration and click the **Edit** ( ) icon.
    f. Click the **Overview** tab.

       If you are using Oracle Fusion Cloud IoT Intelligent Applications version 16 4.1 or earlier, record the **ID** and **Shared Secret** values. These values are required when you run the provisioning tool to create the trusted assets store.
    g. Enter a password in the **File Protection Password** field to encrypt the provisioning file that contains the credentials required to activate your integration.
    h. Enter the password again in the **Confirm Password** field.
    i. Click **Download Provisioning File**.
    j. Click **Save File**.
    k. Browse to a directory and then click **Save**.
13. Prepare your device to run the Android client software libraries. See Prepare Your Device to Use the Android Client Software Libraries .
14. Run the sample applications.

Prepare Your Device to Use the Android Client Software Libraries

A device must be configured correctly and meet the minimum hardware requirements to successfully install the Android client software libraries.

1. Make sure the hardware prerequisites are met before you install the Android client libraries on your device. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System Configurations.
2. Verify device registration:
    a. Open the Oracle Fusion Cloud IoT Intelligent Applications management console.
    b. Click the **Menu** ( ) icon.
    c. Click **Devices** and then **Management**.
    d. Select a device in the device list, or use the **Property** and **Value** fields to search for a device.

e. Confirm **Registered** is displayed in the **State** column.

3. Confirm the location of your Android software development kit (SDK). Typically, it is located at this path: `C:\users\user-name\AppData\Local\Android\sdk`.
4. Set the path for the Android SDK:
   a. Browse to `C:\Users\user-name\Documents\iotcs\csl\<version>\android\samples`.
   b. Open the `local.properties` file in a source code editor.
   c. Enter the Android SDK path as the `sdk.dir` property value.
   d. Enter the user name used to access the Android device as the `user-name` property value.
   e. Save and close the `local.properties` file.
5. Browse to `C:\Users\user-name\Documents\iotcs\csl\<version>\android\lib` and make sure the `device-library.aar` and `enterprise-library.aar` files are in the folder.
6. Set the `device-library.aar` path in the `build.gradle` file:
   a. Browse to `C:\Users\user-name\Documents\iotcs\csl\android\samples\dcd\app`.
   b. Open the `build.gradle` file in a source code editor.
   c. Confirm the path for the `flatDir` property value is correct.
   d. Save and close the `build.gradle` file.
7. Set the Android SDK version in the build files:
   a. Browse to `C:\Users\user-name\AppData\Local\Android\sdk\build-tools`.
   b. Record the Android SDK version.
   c. Browse to `C:\Users\user-name\Documents\iotcs\csl\<version>\android\samples`.
   d. Open the `local.properties` file in a source code editor.
   e. Enter the installed Android SDK version as the `androidBuildToolsVersion` property value.
   f. Save and close the `local.properties` file.
   g. Browse to `C:\Users\user-name\Documents\iotcs\csl\<version>\android\samples\dcd\app`.
   h. Open the `build.gradle` file in a source code editor.
   i. Enter the installed Android SDK version as the `buildToolsVersion` property value.
   j. Save and close the `build.gradle` file.

Create the Android Client Software Library Sample Applications

The Android client library sample applications must be created before they can run on your device.

1. Build the directly connected device sample application:
   a. Open a command prompt.
   b. Browse to `C:\Users\user-name\Documents\iotcs\csl\<version>\android\samples`.
   c. Run this command: `gradle build`.
2. Confirm the Android application package (APK) was created:
   a. Browse to `C:\Users\user-name\Documents\iotcs\csl\<version>\android\samples\dcd\app\build\outputs\apk`.
   b. Confirm the `app-debug.apk` file is in the folder
3. Build the Enterprise sample application:
   a. Open a command prompt.
   b. Browse to `C:\Users\user-name\Documents\iotas\cal\<version>\android\samples\EC`.
   c. Run this command: `gradle assemble`.
4. Confirm the Android application package (APK) was created:
   a. Browse to `C:\Users\user-name\Documents\iotcs\csl\<version>\android\samples\EnterpriseClientSample\app\build\outputs\apk`.
   b. Confirm the `app-debug.apk` file is in the folder
5. Run the sample applications. See Run the Sample Android Directly Connected Device Application or Run the Sample Android Enterprise Application.

Run the Sample Android Directly Connected Device Application

Run the sample Android directly connected device application to learn how to use the client software library APIs. The sample directly connected device applications use software to simulate temperature and humidity sensors. The sample directly connected device applications periodically send temperature and humidity data to Oracle Fusion Cloud IoT Intelligent Applications. You can run the application on an emulator (Android SDK), or on an Android device running Android KitKat 4.4 or later.

> **Note:**
>
> If you have previously installed and configured Android Studio, you do not need to set the Gradle installation path. If you have an existing emulator, you do not need to create a new one.

1. Open **Android Studio** and then select **Open an existing Android Studio project**.
2. (Optional) Set the Gradle installation path:
   a. Click **Cancel**.
   b. Click the **Ellipsis** () button.

c. Browse to the location of your Gradle installation and then click **OK**.

3. (Optional) Create an emulator:
   a. Click **Tools**, **Android**, and then **AVD Manager**.
   b. Click **Create Virtual Device** to create a new emulator.
   c. Select **Phone** in the **Category** list, select a phone model, and then click **Next**.
   d. Select a system image, click **Next**, and then click **Finish**.
4. Open a command prompt and browse to `C:\Users\user-name\AppData\Local\Android\sdk\platform-tools`.
5. Run this command to copy the device provisioning file to the emulator SD card:

   ```
   adb.exe push C:\Users\user-name\Documents\iotcs\csl\android\bin\provisioning-file.conf /sdcard/
   ```

6. Provision the application:
   a. Click the **Run** (▶) icon on the Android Studio toolbar and then select an emulator in the list.
   b. Click **ALLOW**, click **SELECT**, enter the provisioning file password, and then click **PROVISION APPLICATION**.

      To run the application with device policies, select the **Use Device Policy** check box that appears on the emulator screen.
   c. Review the messages displayed on the emulator screen.
7. Confirm data is being sent from the application to Oracle Fusion Cloud IoT Intelligent Applications:
   a. Open the Oracle Fusion Cloud IoT Intelligent Applications management console.
   b. Click the **Menu** (☰) icon.
   c. Click **Devices** and then **Alerts and Messages**.
   d. Confirm temperature and humidity data is being sent to Oracle Fusion Cloud IoT Intelligent Applications.

Run the Sample Android Enterprise Application

Run the sample Android enterprise application to learn how to use the client software library APIs. The sample enterprise application reads humidity and temperature values of directly connected or gateway devices. The sample enterprise applications can also change device attributes by sending commands through Oracle Fusion Cloud IoT Intelligent Applications. You can run the application on an emulator (Android SDK), or on an Android device running Android KitKat 4.4 or later.

> **Note:**
>
> If you have previously installed and configured Android Studio, you do not need to set the Gradle installation path. If you have an existing emulator, you do not need to create a new one.

1. Open **Android Studio** and then select **Open an existing Android Studio project**.
2. (Optional) Set the Gradle installation path:
   a. Click **Cancel**.
   b. Click the **Ellipsis** () button.
   c. Browse to the location of your Gradle installation and then click **OK**.
3. (Optional) Create an emulator:
   a. Click **Tools**, **Android**, and then **AVD Manager**.
   b. Click **Create Virtual Device** to create a new emulator.
   c. Select **Phone** in the **Category** list, select a phone model, and then click **Next**.
   d. Select a system image, click **Next**, and then click **Finish**.
4. Browse to `C:\users\user-name\documents\iotcs\csl\android\samples`, select **EnterpriseSampleApplication**, and click then click **OK**.
5. Open a command prompt and browse to `C:\Users\user-name\AppData\Local\Android\sdk\platform-tools`.
6. Run this command to copy the device provisioning file to the emulator SD card:

   ```
   adb.exe push C:\Users\user-name\Documents\iotcs\csl\android\bin\IntegrationID-provisioning-file-apps.conf\sdcard\
   ```

7. Provision the application:
   a. Click the **Run** () on the Android Studio toolbar and then select an emulator in the list.
   b. Click **ALLOW**, click **SELECT**, enter the provisioning file password, and then click **PROVISION APPLICATION**.
   c. Select **Humidity Sensor** and **Temperature Sensor** and then click **NEXT**.
   d. Select a device and then click **NEXT**.
8. Confirm data is being sent from the application to Oracle Fusion Cloud IoT Intelligent Applications:
   a. Open the Oracle Fusion Cloud IoT Intelligent Applications management console.
   b. Click the **Menu** () icon.
   c. Click **Devices** and then **Alerts and Messages**.

      Use the **Device_ID Temperature Sensor** page to:

      - View the start time for the device.
      - Monitor the current temperature, the minimum and maximum recorded temperatures, and the minimum and maximum thresholds for the device.
      - Control the device. Use the seek bar to change the minimum and maximum temperature thresholds, the **RESET** button to reset

temperature, the **ON**, **OFF** slide button to switch the device on or off, the **BACK** button to return to the **Select Device to Monitor** page.

- Compare the monitoring information with the information displayed in Oracle Fusion Cloud IoT Intelligent Applications.

Use the Python Client Software Libraries

You can develop IoT applications using the Oracle Fusion Cloud IoT Intelligent Applications Python Client Software Libraries. Use the library by downloading the binary provided with the libraries. To run the examples that use the Python Client Software Libraries APIs, you can download the samples bundle provided. To customize the Python Client Software Libraries for your specific development environment, you can download and build the source files.

Topics

Set Up Your Development Environment to Use the Python Client Software Library

Before you can develop applications that let your devices to communicate with Oracle Fusion Cloud IoT Intelligent Applications, you first download and extract the Python client software library.

1. Log in to your Oracle Internet of Things Cloud Service instance.
2. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
3. Create new a application named **PythonCLapp**. See Creating a New Application.
4. Associate the humidity sensor device model with the **PythonCLapp**, record the password, and download the provisioning file. See Register a Single Device.
5. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
6. Scroll to **Python Client Software Library** and download the **Binaries** and **Samples** zip files, `iotcs-csl-python-bin-`*`release`*`.zip` and `iotcs-csl-python-samples-`*`release`*`.zip`, where *release* is the release and version number.
7. Download and install.

Prepare Your Device to Use the Python Client Software Library

Ensure that the software prerequisites are met prior to installing the Oracle IoT Cloud Service Client Library Software for the Python platform on your device. You need to configure your device with the supported operating system and the specific version of the required software.

1. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System . Download and install the required version of Python release software.

2. Open a command prompt and browse to the directory where you downloaded the Python Client Libraries zip files.
3. Run this command to create a Python Virtual Environment (PVE): `python3 -m venv [python virtual environment directory].`
4. Run the activation script to setup the shell environment: `[python virtual environment directory]\Scripts\activate.bat .`
5. Perform this command line option in the same PVE directory in which activate was invoked, to upgrade the Python's `pip` setup tool:
6. If your computer is on a Virtual Private Network, or behind a firewall: `python3 -m pip install --proxy=https://proxy:port --upgrade pip`
7. Download and extract the `requirements.txt` file from: requirements.zip into the same directory where the PVE is located.
8. Run this command to install the requirements in the PVE by using the requirements.txt file you extracted: `pip install -r requirements.txt`.
9. If your computer is on a Virtual Private Network, or behind a firewall: `pip install --proxy=https://proxy:port -r requirements.txt`
10. Run these commands to install the libraries:

    ```
    pip install iotcs-csl-python-bin-release.zip
    pip install iotcs-csl-python-samples-release.zip
    ```

11. Continue with Run the Sample Python Directly Connected Device Application.

Run the Sample Python Directly Connected Device Application

Run the sample Python directly connected device application to learn how to use the client software library APIs. The sample directly connected device application periodically send humidity messages to Oracle Fusion Cloud IoT Intelligent Applications.

1. Copy the provisioning file obtained in Set Up Your Development Environment to Use the Python Client Software Library to the current directory.
2. Run the following command in the same directory:

    ```
    python3 -m iotcs.sample.DirectlyConnectedDeviceSample <provisioner file> <passphrase>
    ```

3. Confirm the device is sending messages to Oracle Fusion Cloud IoT Intelligent Applications:
    a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

    The URL format to access the IoT management console is: *<iot instance name>.<domain name>*`/ui`

    For example: `https://myiotcs.mydomain.oraclecloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

    b. Click the **Menu** (☰) icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.

c. Click **Applications**.

d. Click **PythonCLapp**.

e. Click **Data and Explorations**.

f. Click **Data** tab.

g. In **Filter By**, select Type.

h. In **Message Type** select Data.

i. Confirm there are incoming messages.

4. Deactivate the virtual environment by using this command:

```
[python virtual environment directory]\Scripts\deactivate.bat
```

Run the Sample Python Gateway Application

Run the sample Python gateway application to learn how to use the client software library APIs. The sample gateway application periodically sends humidity and temperature sensor messages to Oracle Fusion Cloud IoT Intelligent Applications.

1. Ensure that you install Python. Follow the steps at Prepare Your Device to Use the Python Client Software Library.
2. Register a new device. See Set Up Your Development Environment to Use the Python Client Software Library and copy the obtained provisioning file to the current directory.
3. Run the following command in the same directory:

```
python3 -m iotcs.sample.GatewayDeviceSample <provisioner file>
<passphrase>
```

4. Confirm the device is sending messages to Oracle Fusion Cloud IoT Intelligent Applications:

   a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

      The URL format to access the IoT management console is: `<iot instance name>.<domain name>/ui`

      For example: `https://myiotcs.mydomain.oraclecloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

   b. Click the **Menu** (☰) icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.

   c. Click **Devices**.

   d. Click **Alerts and Messages**.

   e. Click the **Messages** tab.

   f. Confirm there are incoming messages.

5. Deactivate the virtual environment by using this command:

```
[python virtual environment directory]\Scripts\deactivate.bat
```

Use the C POSIX Client Software Libraries

The C POSIX Client Libraries are designed to enable development of client software across a variety of platforms and operating systems. Two examples are using a Linux operating system on an AMD platform, and an Apple Macintosh OS X operating system running on an AMD platform. Three C POSIX client software libraries are available. To develop applications, download the Binaries library. To run sample applications, download the Samples library. To create a client software library for your specific development environment, download and build the Source Code library.

Topics

Prepare Your Device to Use the C POSIX Client Software Libraries

A device must be configured correctly and meet the minimum hardware requirements to successfully install the Oracle Fusion Cloud IoT Intelligent Applications C POSIX client software libraries.

1. Make sure the hardware prerequisites are met before you install the C POSIX client software library on your target device. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System Configurations.
2. Set up your Raspberry Pi device to use Oracle Fusion Cloud IoT Intelligent Applications. See Set Up a Raspberry Pi Device for Connecting to Oracle IoT Cloud Service.
3. Open a command prompt on your Raspberry Pi device and run the `date` command to make sure the time and date are correct. If the time and date are incorrect:
   a. Run the `sudo raspi-config` command.
   b. Select **Internationalization Options**.
   c. Select **I2 Change Timezone**.
   d. Select your geographical area.
   e. Select a city nearest to your location.
   f. Select **Finish**.
   g. Select **Yes**.
4. Run this command to install dh-autoreconf: `sudo apt-get install dh-autoreconf`.

5. Run this command to instal llibssl-dev: `sudo apt-get install libssl-dev`.
6. Run the `startx` command to open the Raspberry Pi graphical user interface (GUI).
7. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software library download site.
8. Scroll to **C Client Software Libraries** and download the **Binaries** and **Samples** zip files from the POSIX table to the Raspberry Pi home directory (`/home/pi`).
9. Run this command to unzip the binary files: `unzip iotcs-csl-posix-bin-<release number>.zip`.
10. Run this command to unzip the sample files: `unzip iotcs-csl-posix-samples-<release number>.zip`

Build the C POSIX Client Software Library Sample Applications

The C POSIX client software library sample applications must be built before they can run on your device.

1. Open a command prompt on the Raspberry Pi device and run this command to move to the `make` directory: `cd /home/pi/iotcs/csl/posix/samples/make`.
2. Run this command to create the directly connected device sample application: `make clean all LIB_CFG=ts_md_vs`.

   An application named `directly_connected_device_sample.out` is created in the `../build/sample/arm/ts_md_vs` directory.
3. Run this command to create the gateway sample application: `make clean all LIB_CFG=ts_md_vs_gw`.

   An application named `gateway_device_sample.out` is created in the `../build/sample/arm/ts_md_vs_gw` directory.

Run the C POSIX Sample Applications

Run the C POSIX sample applications to learn how to use the client software library APIs. The device client samples use software to simulate temperature and humidity sensors. The device samples periodically send temperature, humidity, and alert messages to Oracle Fusion Cloud IoT Intelligent Applications. The gateway sample sets a threshold value, resets the device, and then switches the device on or off. The attributes, actions, and alerts of the sample temperature and humidity sensors are specified in device models which you upload to Oracle Fusion Cloud IoT Intelligent Applications.

1. Build the samples. See Build the C POSIX Client Software Library Sample Applications.
2. Register the device and download the provisioning file.
3. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
4. Open a command prompt on the Raspberry Pi device and run this command to set the operating system variable : `export IOTCS_OS_NAME=" Raspbian GNU/Linux"`.
5. Run this command to set the operating system version variable: `export IOTCS_OS_VERSION="8"`.
6. Run this command to move to the `build/sample` directory: `cd /home/pi/iotcs/csl/posix/build/sample/arm/ts_md_vs`.

7. Run this command to run the directly connected device sample application: `./directly_connected_device_sample.out <path_to_your_provisioning_file> <your_provisioning_file_password>`.
8. Confirm the device is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications:
    a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

       The URL format to access the IoT management console is: *<iot instance name>*.*<domain name>*`/ui`

       For example: `https://myiotcs.mydomain.oraclecloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

    b. Click the **Menu** () icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.
    c. Click **Devices**.
    d. Click **Alerts and Messages**.
    e. Click the **Messages** tab and confirm there are incoming messages.
9. Run this command to move to the directory containing the sample gateway application: `cd /home/pi/iotcs/csl/posix/build/sample/arm/ts_md_vs_gw`.
10. Run this command to run the gateway sample application: `./gateway_device_sample.out <path_to_your_provisioning_file> <your_provisioning_file_password>`.
11. Repeat step 8 to confirm the gateway is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications.

Run the C POSIX Sample Gateway Application

Run the C POSIX sample gateway application to learn how to use the client software library APIs. The gateway sample sets a threshold value, resets the device, and then switches the device on or off. The attributes, actions, and alerts of the sample temperature and humidity sensors are specified in device models which you upload to Oracle Fusion Cloud IoT Intelligent Applications.

1. Build the samples. See Build the C POSIX Client Software Library Sample Applications.
2. Register the device and download the provisioning file.
3. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
4. Open a command prompt on the Raspberry Pi device and run this command to set the operating system variable : `export IOTCS_OS_NAME=" Raspbian GNU/Linux"`.
5. Run this command to set the operating system version variable: `export IOTCS_OS_VERSION="8"`.
6. Run this command to move to the directory containing the sample gateway application: `cd /home/pi/iotcs/csl/posix/build/sample/arm/ts_md_vs_gw`

7. Run this command to run the gateway sample application: `./gateway_device_sample.out <path_to_your_provisioning_file> <your_provisioning_file_password>`.
8. Confirm the device is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications:
   a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

      The URL format to access the IoT management console is: *<iot instance name>.<domain name>*`/ui`

      For example: `https://myiotcs.mydomain.oraclecloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

   b. Click the **Menu** (☰) icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.
   c. Click **Devices**.
   d. Click **Alerts and Messages**.
   e. Click the **Messages** tab and confirm there are incoming messages.

Build the C POSIX Client Software Libraries

The C POSIX binary file contains libraries for ARM and x86 platforms with Device Library (DL) multi threaded implementation with virtualization support and indirect activation support. These libraries can be used to run the directly connected device sample and gateway device sample applications. To customize and run the C POSIX sample applications, you use the libraries in the source code file.

1. Open a web browser and browse to theOracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
2. Scroll to **C Client Software Libraries** and download the **Source Code** zip file from the POSIX table to the Raspberry Pi home directory (`/home/pi`).
3. Run this command to unzip the binary files:`unzip iotcs-csl-posix-src-<release number>.zip`.
4. Run this command to install Doxygen: `sudo apt-get install doxygen`.
5. Run this command to move to the `make` directory: `cd /home/pi/iotcs/csl/posix/make`.
6. Run this command to compile the source code and build the `libdeviceclient.a` file: `make clean all LIB_CFG=ts`.
7. Run this command to move to the `samples/make` directory: `cd /home/pi/iotcs/csl/posix/samples/make`.
8. Run this command to create the advanced directly connected device sample: `make clean all LIB_CFG=ts`.

   An application named `directly_connected_device_sample.out` is created in the `../build/sample/arm/ts` directory.
9. To build additional sample applications, repeat steps 5 to 8 and replace `make clean all LIB_CFG=ts` with the command listed in the **Command** column. The **Required Libraries** column lists the Oracle client software libraries that are required to create the sample application.

Command	Required Libraries	Description
`make LIB_CFG=ts`	Samples and Source Code	Creates a device library named `directly_connected_device_sample.out` for a single threaded implementation without virtualization.
`make LIB_CFG=ts_md_gw`	Samples and Source Code	Compiles the code specific to a device client and creates the `gateway_device_sample.out` device library with a multi-threaded implementation and a asynchronous message dispatcher. Use this command to create gateway support and without virtualization.
`make LIB_CFG=ts_md_vs`	Samples and Binaries	Creates a `directly_connected_device_sample.out` device library for a multi-threaded implementation with an asynchronous message dispatcher and virtualization.
`make LIB_CFG=ts_md_vs_gw`	Samples and Binaries	Creates a `gateway_device_sample.out` device library for a multi-threaded implementation with an asynchronous message dispatcher, gateway support and virtualization.

Set Up Your Development Environment to use Mac OS X

Before you can develop applications using the C POSIX Client Libraries on a Mac OS X platform, you first download, install, and configure the C client software libraries and set up you Mac OS X environment.

1. Register your device, record the password, and download the provisioning file.
2. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
3. Scroll to **C Client Software Libraries** and download the **Binaries** and **Samples** zip files from the **POSIX** table.
4. Extract the contents of the **Binaries** and **Samples** zip files.
5. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
6. Open a terminal window and change directories to the `iotcs/csl/posix` directory.
7. Install Xcode version 8.1 or Xcode command line tools.
8. Install `HomeBrew`
9. Run this command to install `openssl`: `brew install openssl`
10. Run this command to install `doxygen`: `brew install doxygen`

11. Edit or create `~/.bashrc` add the following lines and save the file:

    ```
    PATH=/usr/local/opt/openssl/bin:$PATH
    export PATH
    ```

12. Source the `~/.bashrc` file with this command: `source ~/.bashrc`
13. Run this command to check that the correct version of openssl is in the path: `openssl version`

    The result should be `OpenSSL x.x.x*`, where `x.x.x` is the latest version number and `*` is a lower case letter.

Build the C POSIX Client Software Libraries on Mac OS X

To run the C POSIX samples on the Mac OS X platform, you must first build the libraries from source code files. You can change which features are supported by the library you build through a configuration option passed to the build command.

1. Set up your Mac OS X development environment. See Set Up Your Development Environment to use Mac OS X.
2. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
3. Scroll to **C Client Software Libraries** and download the **Source Code** zip file from the POSIX table to your development computer.
4. Open a terminal window and change directories to the `iotcs/csl/posix/make` directory.
5. Run this command to build the libraries and create the documentation: `make LIB_CFG=<library option> CPP_OPTS="-I/usr/local/opt/openssl/include" LD_OPTS="-L/usr/local/opt/openssl/lib"`

   Use the following table to determine which value to enter for `<config option>`:

Library Option	Messaging Thread Safety	Message Dispatcher	Virtualization Support	Gateway
`nots`	false	false	false	false
`nots_gw`	false	false	false	true
`ts`	true	false	false	false
`ts_gw`	true	false	false	true
`ts_md`	true	true	false	false
`ts_md_gw`	true	true	false	true
`ts_md_vs`	true	true	true	false
`ts_md_vs_gw`	true	true	true	true

Build the C POSIX Client Software Library Sample Applications

The C POSIX client software library sample applications must be built before they can run on your Mac OS X platform.

1. Set up the environment on your Mac OS X platform. See Set Up Your Development Environment to use Mac OS X

2. Open a terminal window and change directories to: `iotcs/csl/posix/samples/make`.
3. Run this command to create the directly connected device sample application: `make LIB_CFG=<library option> CPP_OPTS="-I/usr/local/opt/openssl/include" LD_OPTS="-L/usr/local/opt/openssl/lib"`

   An application named `directly_connected_device_sample.out` is created in the `../build/sample/x86/ts_md_vs` directory.

   The value of `<library option>` must be one of the following:

   - `nots`
   - `ts`
   - `ts_md`
   - `ts_md_vs`
4. Run this command to create the gateway sample application: `make LIB_CFG=<library option> CPP_OPTS="-I/usr/local/opt/openssl/include" LD_OPTS="-L/usr/local/opt/openssl/lib"`

   An application named `gateway_device_sample.out` is created in the `../build/sample/x86/ts_md_vs_gw` directory.

   The value of `<library option>` must be one of the following:

   - `nots_gw`
   - `ts_gw`
   - `ts_md_gw`
   - `ts_md_vs_gw`

Run the C POSIX Sample Applications on Mac OS X

Run the C POSIX sample applications to learn how to use the client software library APIs. The gateway sample sets a threshold value, resets the device, and then switches the device on or off. The attributes, actions, and alerts of the sample temperature and humidity sensors are specified in device models which you upload to Oracle Fusion Cloud IoT Intelligent Applications.

1. Set up your development environment on Mac OS X. See Set Up Your Development Environment to use Mac OS X.
2. Build the libraries. See Build the C POSIX Client Software Libraries on Mac OS X.
3. Build the samples. See Build the C POSIX Client Software Library Sample Applications.
4. Open a terminal window and run these commands:

   ```
   export IOTCS_OS_NAME=OSX
   export IOTCS_OS_VERSION=10.1
   ```

5. Change directories to the `build/sample/x86/<library option>` directory, where `<library option>` is the library used to build the sample. For example, `ts_md_vs`.

6. Run this command to run the directly connected device sample application: `./directly_connected_device_sample.out <path_to_your_provisioning_file> <your_provisioning_file_password>`.
7. Confirm the device is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications:
   a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

      The URL format to access the IoT management console is: *<iot instance name>.<domain name>*`/ui`

      For example: `https://myiotcs.mydomain.oraclecloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

   b. Click the **Menu** (☰) icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.
   c. Click **Devices**.
   d. Click **Alerts and Messages**.
   e. Click the **Messages** tab and confirm there are incoming messages.
8. Change directories to the `iotcs/csl/posix/build/sample/x86/<library option>` directory, where `<library option>` is the library used to build the gateway sample. For example, `ts_md_vs_gw`.
9. Run this command to run the gateway sample application: `./gateway_device_sample.out <path_to_your_provisioning_file> <your_provisioning_file_password>`.
10. Repeat step 7 to confirm the gateway is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications.

Use the Windows Client Software Libraries

Three Windows client software libraries are available. To develop applications, download the Binaries library. To run sample applications, download the Samples library. To create a client software library for your specific development environment, download and build the Source Code library.

Topics

Set Up Your Development Environment to Use the Windows Client Software Libraries

Before you can develop applications that let your devices to communicate with Oracle Fusion Cloud IoT Intelligent Applications, you first download, install, and configure the Windows client software libraries.

1. Prepare your device for the installation of the Windows client software libraries. See Prepare Your Device to Use the Windows Client Software Library.
2. Register your device, record the password, and download the provisioning file.
3. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
4. Scroll to **C Client Software Libraries** and download the **Binaries** and **Samples** zip files from the **Windows** table.
5. Extract the contents of the **Binaries** and **Samples** zip files.
6. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
7. Open a command prompt and run this command to move to the `iotcs/csl/windows/bin` directory: `cd iotcs/csl/windows/bin`.
8. Download the latest version of OpenSSL from https://www.openssl.org/source/. or run this command to download the OpenSSL file: `wget https://www.openssl.org/source/openssl-x.x.x*.tar.gz` where `x.x.x` is the latest version number and `*` is a lower case letter.

   This command example assumes the `wget` utility is installed.
9. Run this command to extract the `openssl-x.x.x*.tar.gz` file: `tar -xvf openssl-x.x.x*.tar.gz` where `x.x.x` is the latest version number and `*` is a lower case letter.
10. Run this command to move to the `openssl-x.x.x*` folder: `cd <filepath>/openssl-x.x.x*` where `x.x.x` is the latest version number and `*` is a lower case letter.
11. Run this command to determine the path to the Visual Studio environment variable: `set| findstr -i comntools`.

    The command should return a response similar to:`VS150COMNTOOLS=C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\Tools\`
12. Run these batch scripts to set the path and environment variables for command-line builds:

    ```
    $(VS150COMNTOOLS)\VsDevCmd.bat
    $(VS150COMNTOOLS)\..\..\VC\Auxiliary\Build\vcvars32.bat
    ```
13. Run this command to configure OpenSSL for Windows: `perl Configure VC-WIN32 no-asm .`
14. Run this command to build the OpenSSL libraries: `nmake /nologo .`
15. Run this command to copy the generated libraries to the `iotcs/csl/windows/bin` folder: `copy /y *.lib ...`

Prepare Your Device to Use the Windows Client Software Library

A device must be configured correctly and meet the minimum hardware requirements to successfully install the Windows client software library.

1. Make sure the hardware prerequisites are met before you install the Windows client software library on your device. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System Configurations.
2. Make sure the time and date on your device is current. If it isn't, open a command prompt and run these commands:
   a. Run the `date` command and update the date.
   b. Run the `time` command and update the time.
3. Download and install Microsoft Visual Studio 2013 or later on your Windows computer.
4. Download and install the Cygwin (version 2.6 or later) `make`, `wget`, `perl`, `tar`, `zip`, `unzip`, `xxd`, and `curl` packages on the device.
5. Run this command to set the PATH system variable to the location of the Cygwin installation directory: `setx path "%path%;c:\cygwin\bin"`.
6. Set up your development environment to use the Windows client software libraries. See Set Up Your Development Environment to Use the Windows Client Software Libraries.

Create the Windows Client Software Library Sample Applications

The Windows client software library sample applications must be created before they can run on your device.

1. Set up your development environment. See Set Up Your Development Environment to Use the Windows Client Software Libraries.
2. Open a command prompt and run this command to move to the `make` directory: `cd iotcs/csl/windows/samples/make`.
3. Run this command to build a sample application: `make build [CC_CFG=<CC_CFG_Option>][LIB_CFG=<LIB_CFG_Option>][PROXY=<your-company-proxy-server>]`.

   Replace `<CC_CFG_Option>` and `<LIB_CFG_Option>` with one of the commands listed in the table. Replace `<your-company-proxy-server>` with the IP address of your company server if you are running the sample applications behind a corporate firewall.

Command	Where Used	Description
`cl`	`<CC_CFG_Option>`	Builds the client software library on Windows computers. This is the default compiler.
`gcc`	`<CC_CFG_Option>`	Builds the client software library on Linux computers and adds the GNU Compiler Collection (GCC).
`nots`	`<LIB_CFG_Option>`	Builds the directly connected device sample application with these options: messaging thread safety=false, messaging dispatcher=false, and virtualization support=false.
`ts`	`<LIB_CFG_Option>`	Builds the directly connected device sample application with these options: messaging thread safety=true, messaging dispatcher= false, and virtualization support=false.
`ts_md_gw`	`<LIB_CFG_Option>`	Builds the gateway device sample application with these options: messaging thread safety=true, messaging dispatcher= true, virtualization support=false, and indirect activation=true.

Command	Where Used	Description
ts_md_vs	<LIB_CFG_Option>	Builds a directly connected device sample application with these options: messaging thread safety=true, messaging dispatcher= true, and virtualization support=true.
ts_md_vs_gw	<LIB_CFG_Option>	Builds a gateway device sample application with these options: messaging thread safety=true, messaging dispatcher= true, virtualization support=true, and indirect activation=true. This is the default value for the LIB_CFG parameter.

4. (Optional) Run this command to clean the sample applications: `make clean [LIB_CFG=<LIB_CFG_Option>]`.
5. Run the sample applications. See Run the Windows Sample Applications.

Run the Windows Sample Applications

Run the Windows sample applications to learn how to use the client software library APIs. The device client samples use software to simulate temperature and humidity sensors. The device samples periodically send temperature, humidity, and alert messages to Oracle Fusion Cloud IoT Intelligent Applications. The gateway sample sets a threshold value, resets the device, and then switches the device on or off. The attributes, actions, and alerts of the sample temperature and humidity sensors are specified in device models which you upload to Oracle Fusion Cloud IoT Intelligent Applications.

1. Create the sample applications. See Create the Windows Client Software Library Sample Applications.
2. Register the device and download the provisioning file.
3. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
4. Open a command prompt and run this command to set the `IOTCS_OS_NAME` environment variable: `SET IOTCS_OS_NAME "%Windows%"`.
5. Run this command to set the `IOTCS_OS_VERSION` environment variable: `SET IOTCS_OS_Version "%7%"`.
6. Run this command to move to the `sample` directory: `cd iotcs/csl/windows / build/sample/x86/<sample_directory>`. Replace `<sample_directory>` with the name of the folder where you created the sample applications.
7. Run this command to run the directly connected device sample application: `sample_name.exe ./trusted_asset_store password`. Replace `sample_name.exe` with the name of the sample application you want to run, `trusted_asset_store` with the path to the provisioning file you downloaded when registering your device, and `password` with the password used to protect the provisioning file.
8. Confirm the device is sending humidity and temperature data to Oracle Fusion Cloud IoT Intelligent Applications:
   a. Open the Oracle Fusion Cloud IoT Intelligent Applications Management Console.

      The URL format to access the IoT management console is: *<iot instance name>.<domain name>*/**ui**

For example: `https://myiotcs.mydomain.oraclecloud.com/ui` or `https://myiotcs.mydomain.oracleiotcloud.com/ui`.

b. Click the **Menu** () icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.

c. Click **Devices**.

d. Click **Alerts and Messages**.

e. Click the **Messages** tab and confirm there are incoming messages.

Build the Windows Client Software Libraries

To customize the Windows sample applications, you use the libraries in the source code file.

1. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
2. Scroll to **C Client Software Libraries** and download the **Source Code** zip file from the Windows table to your development computer.
3. Open a command prompt and run this command to move to the directory where you saved the source code zip file: `cd C:\users\yourname\downloads`.
4. Run this command to unzip the source code files:`unzip iotcs-csl-windows-src-<release number>.zip`

   This command sample assumes you have unzip.exe or a similar utility installed.
5. Run this command to move to the `make` folder: `cd iotcs\csl\windows\samples\make`.
6. Run this command to build a sample application: `make build [CC_CFG=<CC_CFG_Option>][LIB_CFG=<LIB_CFG_Option>][PROXY=<your-company-proxy-server>]`.

   Replace `<CC_CFG_Option>` and `<LIB_CFG_Option>` with one of the commands listed in the table. Replace `<your-company-proxy-server>` with the IP address of your company server if you are running the sample applications behind a corporate firewall.

Command	Where Used	Description
`cl`	`<CC_CFG_Option>`	Builds the client software library on Windows computers. This is the default compiler.
`gcc`	`<CC_CFG_Option>`	Builds the client software library on Linux computers and adds the GNU Compiler Collection (GCC).
`all`	`<LIB_CFG_Option>`	Builds all of the application samples, including: nots, nots_gw, ts, ts_gw, ts_md, ts_md_gw, ts_md_vs, ts_md_vs_gw.
`nots`	`<LIB_CFG_Option>`	Builds the directly connected device sample application with these options: messaging thread safety=false, messaging dispatcher=false, and virtualization support=false.
`ts`	`<LIB_CFG_Option>`	Builds the directly connected device sample application with these options: messaging thread safety=true, messaging dispatcher= false, and virtualization support=false.

Command	Where Used	Description
ts_md_gw	<LIB_CFG_Option>	Builds the gateway device sample application with these options: messaging thread safety=true, messaging dispatcher= true, virtualization support=false, and indirect activation=true.
ts_md_vs	<LIB_CFG_Option>	Builds a directly connected device sample application with these options: messaging thread safety=true, messaging dispatcher= true, and virtualization support=true.
ts_md_vs_gw	<LIB_CFG_Option>	Builds a gateway device sample application with these options: messaging thread safety=true, messaging dispatcher= true, virtualization support=true, and indirect activation=true. This is the default value for the LIB_CFG parameter.

7. (Optional) Run this command to clean the sample applications: `make clean [LIB_CFG=<LIB_CFG_Option>]`.

Use the iOS Client Software Libraries

Three iOS client software libraries are available. To develop applications, download the Binaries library. To run sample applications, download the Samples library. To create a client software library for your specific development environment, download and build the Source Code library.

Topics

Set Up Your Development Environment to Use the iOS Client Software Libraries

Before you can develop applications that let your devices to communicate with Oracle Fusion Cloud IoT Intelligent Applications, you first download and extract the iOS client software libraries.

These items are required to complete this procedure:

- An Apple Macintosh computer running Mac OS X version 10.12 or later.
- Xcode version 8.3.2 or later installed on the development computer.
- iOS version 10.3 or later installed on the device.

1. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
2. Scroll to **iOS Client Software Libraries** and download the **Binaries** and **Samples** zip files from the **iOS** table. This table describes the contents of each zip file:

Filename	Description
iotcs-csl-ios-bin-<version>.zip	Contains the iOS client software library binary files.
iotcs-csl-ios-samples-<version>.zip	Contains the iOS sample applications.

3. Extract the contents of the zip files. The files are saved to these directories:

Filename	Directory
iotcs-csl-ios-bin-<version>.zip	iotcs/csl/ios/bin
iotcs-csl-ios-samples-<version>.zip	iotcs/csl/ios/samples

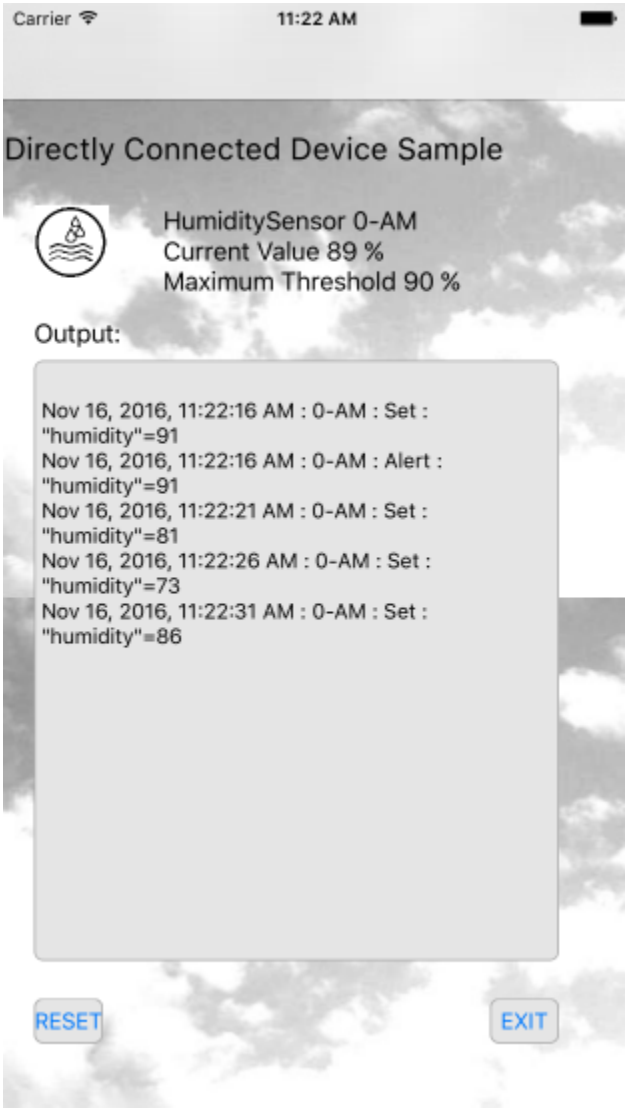
Run the Sample Directly Connected Device Application

Run the iOS sample directly connected device application to learn how to use the client software library APIs. The device client samples use software to simulate temperature and humidity sensors. The class `HumiditySensor` is used to simulate data points on a sine wave. The directly connected device sample applications periodically send temperature, humidity, and alert messages to Oracle Fusion Cloud IoT Intelligent Applications. The attributes, actions, and alerts of the sample temperature and humidity sensors are specified in device models which you upload to Oracle Fusion Cloud IoT Intelligent Applications.

About the Sample Directly Connected Device Application

The sample application is located in the `iotcs/csl/ios/samples` directory and it uses a high-level, virtual device abstraction that hides the details of sending and receiving data from Oracle Fusion Cloud IoT Intelligent Applications.

1. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
2. Register the device and download the provisioning file.
3. Run the `DirectlyConnectedDeviceSample` application:
   a. Open Xcode and then open the `DirectlyConnectedDeviceSample.xcodeproj` file in the `iotcs/csl/ios/samples` directory.
   b. Right-click the project name and select **Add Files to "DirectlyConnectedDeviceSample"**.
   c. Browse to the location of the provisioning file and then click **Add**.
   d. Expand the `DirectlyConnectedDeviceSample` folder and select the `TrustStore.plist` file.
   e. Click the `filename` field and enter the name of the provisioning file without the file extension.
   f. Click the `fileextension` field and enter the file extension of the provisioning file.
   g. Click the `password` field and enter the provisioning file password.
   h. Run the `DirectlyConnectedDeviceSample` application. Output similar to this image appears:



Run the Sample Gateway Application

Run the iOS sample gateway application to learn how to use the client software library APIs. The gateway device samples use software to simulate temperature and humidity sensors. The class `TemperatureSensor` and the class `HumiditySensor` are used to simulate data points on a sine wave. The gateway sample applications periodically send temperature, humidity, and alert messages to Oracle Fusion Cloud IoT Intelligent Applications. The gateway sample sets a threshold value, resets the device, and then switches the device on or off. The attributes, actions, and alerts of the sample temperature and humidity sensors are specified in device models which you upload to Oracle Fusion Cloud IoT Intelligent Applications.

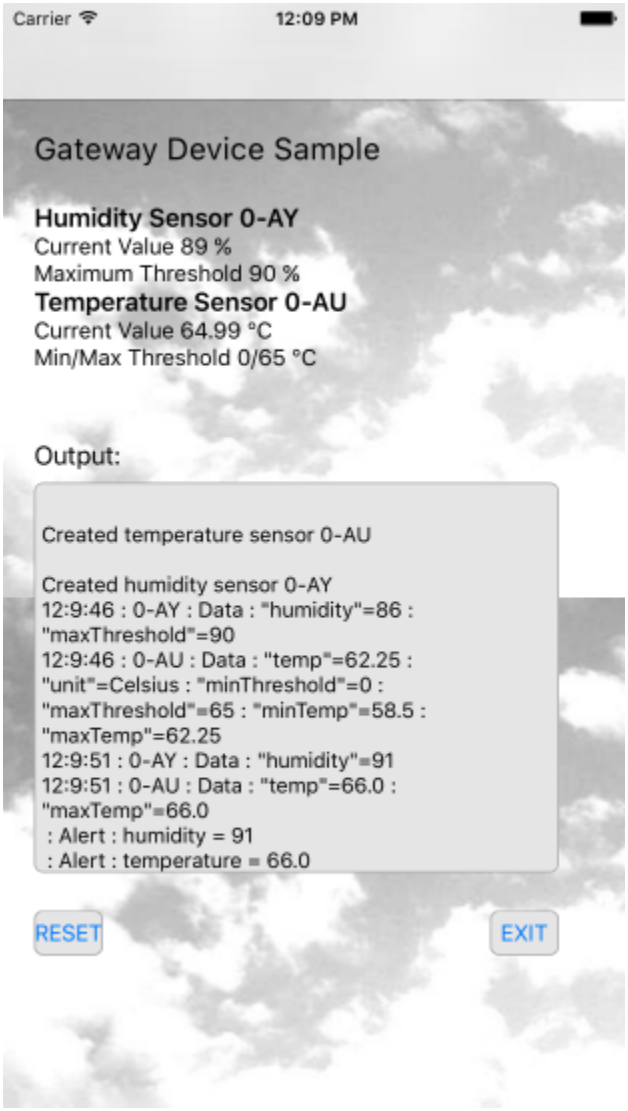
About the Sample Gateway Application

The sample application is located in the `iotcs/csl/ios/samples` directory and it uses a high-level, virtual device abstraction that hides details of sending and receiving data from Oracle Fusion Cloud IoT Intelligent Applications.

1. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
2. Register the device and download the provisioning file.

> **Note:**
>
> Do not reuse the device from the *Running the Sample Directly Connected Device Applications* procedure. This procedure requires a device with indirect activation capability.

3. Run the `GatewayDeviceSample` application:
   a. Open Xcode and then open the `GatewayDeviceSample.xcodeproj` file located in the `iotcs/csl/ios/samples` directory.
   b. Right-click the project name and select **Add Files to "GatewayDeviceSample"**.
   c. Browse to the location of the provisioning file and then click **Add**.
   d. Expand the `GatewayDeviceSample` folder and select the `TrustStore.plist` file.
   e. Click the `filename` field and enter the name of your provisioning file without the file extension.
   f. Click the `fileextension` field and enter file the extension of the provisioning file.
   g. Click the `password` field and enter the provisioning file password.
   h. Run the `GatewayDeviceSample` application. Output similar to this image appears:



Run the Sample Enterprise Applications

Run the iOS sample enterprise applications to learn how to use the client software library APIs. The sample enterprise applications simulates software that communicates with and controls directly connected or gateway devices. The sample enterprise application reads humidity and temperature values of directly connected or gateway devices. The sample enterprise applications can also change device attributes by sending commands through Oracle Fusion Cloud IoT Intelligent Applications.

About the Sample Enterprise Applications

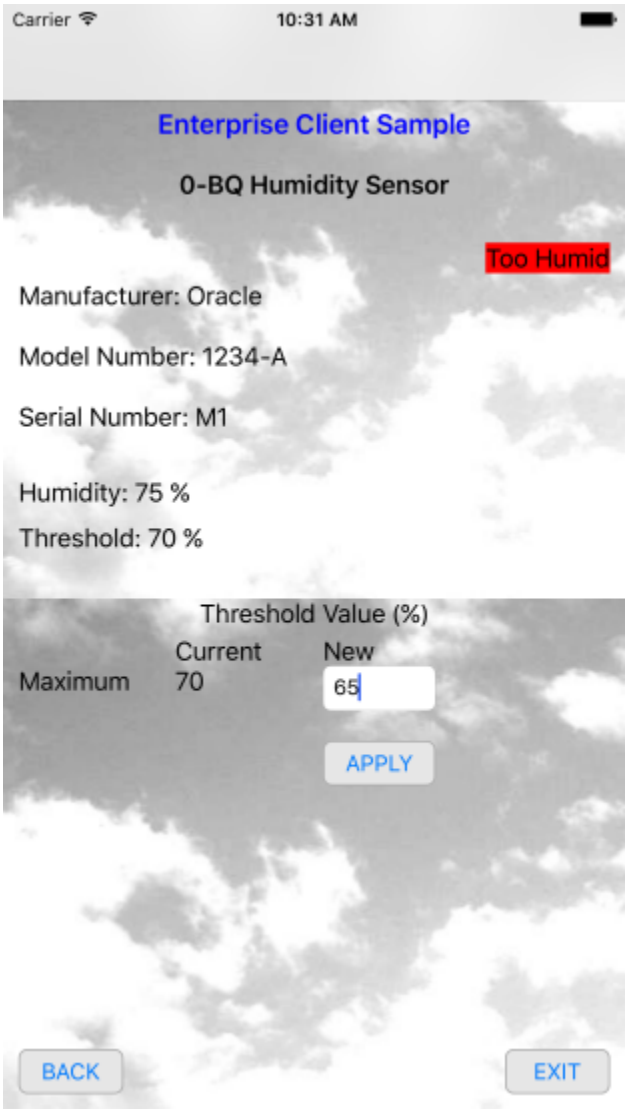
Two sample enterprise applications are available. one that demonstrates a gateway connecting using virtualization and a sample that demonstrates a gateway connecting using direct messaging. The first sample is located in the `iotcs/csl/ios/samples` directory and it uses a high-level, virtual device abstraction that hides details of sending and receiving data from Oracle Fusion Cloud IoT Intelligent Applications. The second sample is located in the `iotcs/csl/ios/samples/advanced` folder and it uses a send and receive model to provide direct control over the client software library.

1. Upload the humidity and temperature sensor device models to Oracle Fusion Cloud IoT Intelligent Applications. See Upload the Sample Device Models.
2. Create a new application named **iOS Device**. See Creating a New Application.
3. Associate the humidity and temperature sensor device models with the **iOS Device** application.
4. Add an integration named **iOS Device** to the application. See Integrating Enterprise Applications with Oracle IoT Cloud Service.
5. Download the provisioning file for the integration:
   a. Log in to your Oracle Fusion Cloud IoT Intelligent Applications instance.
   b. Click the **Menu** () icon adjacent to the Oracle Fusion Cloud IoT Intelligent Applications title on the Management Console.
   c. Click **Applications** and then **Browse Applications**.
   d. Click **iOS Device** and then **Integration**.
   e. Select the **iOS Device** integration and click the **Edit** () icon.
   f. Click the **Overview** tab.

      If you are using Oracle Fusion Cloud IoT Intelligent Applications version 17.3.3 or earlier, record the **ID** and **Shared Secret** values. These values may be required when you run the provisioning tool to create the trusted assets store.
   g. Enter a password in the **File Protection Password** field to encrypt the provisioning file that contains the configuration and credentials to activate your integration.
   h. Enter the password again in the **Confirm Password** field.
   i. Click **Download Provisioning File**.
   j. Click **Save File**.
   k. Browse to a directory and then click **Save**.
6. Run the `DirectlyConnectedDevice` or the `GatewayDevice` sample applications. See Run the Sample Directly Connected Device Application or Run the Sample Gateway Application.
7. Run the `EnterpriseClientSample` application:
   a. Open Xcode and then open the `EnterpriseClientSample.xcodeproj` file located in the `iotcs/csl/ios/samples` directory.
   b. Right-click the project name and select **Add Files to "GatewayDeviceSample"**.
   c. Browse to the location of the provisioning file and then click **Add**.
   d. Expand the `EnterpriseClientSample` folder and select the `TrustStore.plist` file.
   e. Click the `filename` field and enter the name of the provisioning file without the file extension.
   f. Click the `fileextension` field and enter the file extension of the provisioning file.
   g. Click the `password` field and enter the provisioning file password.
   h. Run the `EnterpriseClientSample` application.
   i. On the first screen enter **iOS Device** and click **Next**.
   j. Select **Humidity Sensor**.

k. Select a device to monitor and control.

l. Click in the **New** field and enter a value between 65 and 100 and click **Apply**. Output similar to this image appears:



Note that the `maxThreshold` value is changed on the device.

Build the iOS Client Software Libraries

Build the client software libraries from the provided source files.

You can use an automated build script and Gradle or Xcode to build the client software libraries.

1. Open a web browser and browse to the Oracle Fusion Cloud IoT Intelligent Applications client software libraries download site.
2. Scroll to **iOS Client Software Libraries** and download the **Source Code** zip file from the **iOS** table.
3. Extract the contents of the zip file.
4. Download and install Gradle. Versions 2.5 to 2.13 are supported.

5. If your computer is on a Virtual Private Network, or behind a firewall:
   a. Open the `gradle.properties` file, located in the Gradle user home directory and add these lines:

      ```
      systemProp.http.proxyHost=<your_proxy_server.com>
      systemProp.http.proxyPort=<your_proxy_port>
      systemProp.https.proxyHost=<your_proxy_server.com>
      systemProp.https.proxyPort=<your_proxy_port>
      ```

      The default value for the Gradle user home directory is `USER_HOME/.gradle`. To use a different directory, set the `GRADLE_USER_HOME` environment variable.
   b. Save your changes and close the `gradle.properties` file.
6. Build the iOS libraries using an automated script:
   a. Open a terminal window and change directories to `iotcs/csl/ios`.
   b. Run the command `gradle build` to build the iOS device and enterprise client software libraries.

      To build the device and enterprise libraries individually run one of these commands:

Command	Description
`build device`	Builds the iOS device library. The library is created in the `iotcs/csl/ios/lib/DeviceLib.framework` directory.
`build enterprise`	Builds the iOS enterprise library. The library is created in the `iotcs/csl/ios/lib/EnterpriseLib.framework` directory.

7. Build the iOS libraries using Xcode:
   a. Open Xcode, open `DeviceLib.xcodeproj` in the `iotcs/csl/ios/src/device/DeviceLib.xcodeproj` directory, and then build the project.
   b. Open Xcode, open `EnterpriseLib.xcodeproj` in the `iotcs/csl/ios/src/enterprise/` directory, and then build the project.

Network Provisioning Support in Client Libraries

Network provisioning enables your IoT application to dynamically provide provisioning information over the network for your devices. IoT applications can use network provisioning for seamless onboarding and activation of trusted assets.

When registering new devices with Oracle IoT Cloud Service, you need to provision the devices with information like the server name, port, device ID, and certificate before they can connect securely with Oracle IoT Cloud Service. Network provisioning enables you to dynamically provision your devices over the network.

After a device is registered, it waits for the provisioning information from a network provisioner. Once it receives the provisioning information, the device can perform activation and start communicating messages with Oracle IoT Cloud Service.

Network provisioning support enables you to build applications that can dynamically register and provision your devices. A service technician with the required privileges/roles to register a device, for example, can use the Oracle Asset Monitoring mobile application to scan the

device details QR code, register the device using these details, and dynamically provision the device over the network.

To enable network provisioning, the client libraries include the bootstrapper and network provisioning utilities:

- Bootstrapper: The client libraries include the bootstrapper utility. When the device is switched on, the bootstrapper looks into the specified trusted assets store to check if provisioning is complete for the device. If the device isn't provisioned, then the bootstrapper waits for it to get provisioned before handing over the control to the device application that completes activation. On subsequent reboots, the bootstrapper detects that the device is already provisioned and directly launches the device application.
- Network Provisioner: The network provisioner is a stand-alone application that discovers available clients waiting to be provisioned, and sends them the provisioning information required to complete provisioning. The provisioning information includes information like the server name, port, device ID, and certificate required for the device to securely connect to the Oracle IoT Cloud Service server.

  The network provisioner uses UDP multicast messages to discover registered clients waiting to be provisioned. The network provisioner returns this list to the user, who then selects the device to be provisioned. The network provisioner next unicasts the provisioning information to the selected device. The device stores the provisioning information in its trusted assets store, and uses the information to perform direct activation.

Network Provisioner

Network Provisioner Usage:

```
networkProvisioner.[sh|bat] [client_host provisioning_file]
```

Where:

client_host is the IP address of the gateway.

provisioning_file is the name of the file containing the provisioning information in the unified provisioner format (upf).

For example:

```
./bin/networkProvisioner 10.0.0.1 ~/Downloads/mytas.ufp
```

Java Client Library Bootstrapper and Network Provisioner

The Bootstrapper class files are contained in the `lib/bootstrapper.jar` file.

- Bootstrapper Usage:

```
java oracle.iot.client.util.Bootstrapper trust_assets_file
trust_assets_password
   application_class_name
[application_argument_0...application_argument_8]
```

Where:

trust_assets_file is the relative or fully-qualified filename of the trusted assets store.

trust_assets_password is the password of the trusted assets store. It must match the file protection password of the provisioning file.

application_class_name is the name of the application to start.

application_argument_0 .... application_argument_8 are any optional arguments to be passed to the application.

So, for example:

```
java -cp ../lib/bootstrapper.jar:./build/libs/iotcs-csl-samples.jar
   oracle.iot.client.util.Bootstrapper MyFile MyPassword123
com.oracle.iot.sample.DirectlyConnectedDeviceSample
```

- Network Provisioner Usage:

```
java -jar network-provisioner.jar [client_host Provisioning_file]
```

Where:

client_host is the IP address of the device where the bootstrapper is running.

provisioning_file is the name of the file containing the provisioning information.

If no arguments are provided, then the network provisioner discovers the available clients. Else, the network provisioner provisions the specified client.

JavaScript Client Library Bootstrapper and Network Provisioner

- Bootstrapper Usage:

> **Note:**
>
> Make sure that the `NODE_PATH` environment variable points to the location where the required node modules are installed.

```
bootstrapper.[sh|bat] provisioned_file password app_class_name
[app_arg_0...app_arg_8]
```

Where:

provisioned_file is the relative or fully-qualified filename of the trusted assets store.

password is the password of the trusted assets store. It must match the file protection password of the provisioning file.

app_class_name is the name of the application to start.

app_arg_0 .... app_arg_8 are any optional arguments to be passed to the application.

- Network Provisioner Usage:

> **Note:**
>
> Make sure that the `NODE_PATH` environment variable points to the location where the required node modules are installed.

```
networkProvisioner.[sh|bat] [client_host provisioning_file]
```

Where:

client_host is the IP address of the device where the bootstrapper is running.

provisioning_file is the name of the file containing the provisioning information in the unified provisioner format (upf).

If no arguments are provided, then the network provisioner discovers the available clients. Else, the network provisioner provisions the specified client.

C Client Library Bootstrapper

- Bootstrapper Usage:

```
# Posix
$PATH_TO_BT/bootstrapper taStore=trusted_assets_store
taStorePassword=password
$PATH_TO_SAMPLE/directly_connected_device_sample
trusted_assets_store password
# Windows
$PATH_TO_BT/bootstrapper.exe taStore=trusted_assets_store
taStorePassword=password
$PATH_TO_SAMPLE/directly_connected_device_sample.exe
trusted_assets_store password
```

For MBED, the samples automatically perform bootstrap. If the provisioning information is not found in the trusted assets store, network provisioning is attempted over the UDP connection.

Android Client Library and iOS Client Library Samples

The samples can use network provisioning if the network provisioning option is selected on the Provisioning screen.

3

Integrate Oracle IoT Cloud Service with Third Party Device Management Applications

This section provides information about integrating Oracle IoT Cloud Service with third party device management applications.

Topics

Register and Provision a Device Using Third Party Device Management Application

Register third party devices with Oracle IoT Cloud Service by using REST APIs.

Third party vendors must register all devices with Oracle IoT using REST APIs. See the REST APIs for registering devices.

Prior to registering the devices, you must decide the device model.

Device Model: Device models must be defined and then registered with Oracle IoT Cloud Service using the REST API, **POST /iot/api/v2/deviceModels**.

Device Type: At the time a device activates, it declares its device type. Device type determines the way a device is connected to Oracle IoT Cloud Service. A device can be directly connected or indirectly connected to Oracle IoT Cloud Service.

A directly connected device is capable of communicating directly with Oracle IoT Cloud Service by running an application that uses an Oracle IoT Cloud Service Client Software Library or by calling the Oracle IoT Cloud Service REST APIs. This device type cannot register any indirectly connected devices. The only supported direct connection protocol is HTTPS over TCP/IP.

An indirectly connected device communicates with Oracle IoT Cloud Service through a Gateway Device. These devices may communicate with the Gateway Device over a non-HTTPS TCP/IP protocol or interface, such as Bluetooth, Zigbee, I2C, or GPIO. For this device to indirectly communicate with Oracle IoT Cloud Service, it is required to be connected to a Gateway Device that has already been activated with Oracle IoT Cloud Service.

Topics

Add Device Models to Oracle IoT Cloud Service

You need to add device models before registering devices in Oracle IoT Cloud Service. To add device models, use the REST API, **POST /iot/api/v2/deviceModels**.

Parameters	Description
Uniform Resource Name (URN)	A device model uses the URN as its unique ID.
Name	Specify a descriptive name for the device.
Description	Enter a device description.
Attributes	A device model's attributes represent the basic variables that the device supports.

The following code snippet gives an example of a JSON message structure for adding device models:

```
{
   "urn":"urn:acao:codemax:box22",
   "name":"Codemax 1.1",
   "description":"This device model matches the Codemax specs
v20160513 limited to temp, hygr, lux, noiseAvg and noiseMax. No
feedback support yet.",
   "system":false,
   "attributes":[
      {
         "name":"temp",
         "description":"",
         "type":"NUMBER",
         "range":"0.0,50.0",
         "alias":"Temperature",
         "writable":false
      },
      {
         "name":"hygr",
         "description":"",
         "type":"NUMBER",
         "range":"5.0,85.0",
         "alias":"Hygrometry",
         "writable":false
      },
      {
         "name":"lux",
         "description":"",
         "type":"NUMBER",
         "range":"0.0,10000.0",
         "alias":"Luminosity",
         "writable":false
      },
      {
         "name":"noiseAvg",
         "description":"",
         "type":"NUMBER",
```

```
            "range":"0.0,100.0",
            "alias":"Noise_Avg",
            "writable":false
         },
         {
            "name":"noiseMax",
            "description":"",
            "type":"NUMBER",
            "range":"0.0,100.0",
            "alias":"Noise_Max",
            "writable":false
         }
      ],
      "actions":[

      ],
      "formats":[

      ]
}
```

> **Note:**
>
> The above JSON structure does not contain alerts for out of range values sent by devices. The structure ignores messages with incorrect values and does not send them to the Oracle IoT Cloud Service.

This table lists the fields required to specify an attribute within a device model:

Attributes	Required	Description
Name	Yes	Specify the name of the attribute.
Description	No	Enter a description.
Type	Yes	Specify the type of data represented by the attribute.
Range	No	Specify the minimum and maximum range.
writeable	No	By default, device model attributes cannot be modified from Oracle IoT Cloud Service.

Specify Devices as Third Party Partner Devices in Oracle IoT Cloud Service

You can register all devices in Oracle IoT Cloud as third party partner devices by using this REST API: `POST /iot/api/v2/private/partners`.

Attribute	Description
thirdPartyPartnerName	Specify the name of the third party vendor.

Attribute	Description
thirdPartyPartnerDescr	Enter a description for the third party device.
thirdPartyPartnerUrl	Provide the URL of the third party dashboard.

Here is an example of how to register a device as a third party entity in Oracle IoT Cloud Service:

```
thirdPartyPartnerName: ACME,
thirdPartyPartnerDescr: ACME Live Objects PTE,
thirdPartyPartnerUrl: https://acme-webportal.com,
```

When you specify the third party partner details using the **POST /iot/api/v2/private/partners** REST API, the third party device management application includes the following metadata for each device in Oracle IoT Cloud:

- **X-IOT-External-Device-ID**: The ID of the device in the third party Cloud instance.
- **X-IOT-External-Device-URL**: The URL of the device in the third party Cloud instance.

Register Devices with Oracle IoT Cloud Service

To register a device with Oracle IoT Cloud Service, use this REST API:
`POST /iot/api/v2/devices {serialNumber: <serial_number>, modelNumber: <model_number>, manufacturer: <manufacturer>, hardwareId: <hardware_id>}.`

See REST API Create a new Device.

Activate and Deactivate Devices in Oracle IoT Cloud Service

To activate a device in Oracle IoT Cloud Service that has been temporarily disabled, use this REST API:`PATCH /iot/api/v2/devices/<device-id> {state: ACTIVATED}.`
To deactivate a device in Oracle IoT Cloud Service, use this REST API:
`PATCH /iot/api/v2/devices/<device-id> {state: DISABLED}.`

See REST API Update a Device by ID.

Delete Devices from Oracle IoT Cloud Service

To delete a device from Oracle IoT Cloud Service, use this REST API:
`DELETE /iot/api/v2/devices/<device-id>`

See REST API Delete a Device by ID.