Contents

Preface

Audience vii
Documentation Accessibility vii
Related Documents vii
Conventions viii

1 Learn About Device Connectivity

About IoT Devices 1-1
About the IoT Connectivity Protocols 1-1
Methods to Connect Devices to Oracle Internet of Things Cloud Service 1-3
About Selecting the Method to Connect Devices 1-3
Get Started 1-5
Before You Use the Cloud Service 1-5
Access the Management Console 1-6
Supported Browsers 1-6
Supported Platforms 1-6
How to Get Support 1-7
Locating Diagnostic Information for Oracle Support 1-7

2 Develop Device Software Using the Client Software Libraries

Device Virtualization 2-1
Client Library Best Practices 2-2
Use the Java SE Client Software Libraries 2-2
Set Up Your Development Environment to Use the Java SE Client Software Libraries 2-3
Prepare an Embedded Device to Use the Java SE Client Software Library 2-8
Create the Java SE Client Software Library Sample Applications 2-10
Use the Provisioning Tool to Create the Truststore 2-10
Run the Sample Java SE Directly Connected Device Applications 2-11
Run the Sample Java SE Gateway Application 2-12
Run the Sample Java SE Gateway Application Using Apache Felix 2-13
Run the Sample Java SE Enterprise Applications 2-19
Build the Java SE Client Software Libraries 2-24
Use the JavaScript Client Software Libraries 2-25
  Set Up Your Development Environment to Use the JavaScript Client Software Libraries 2-25
  Prepare Your Device to Use the JavaScript Client Software Library 2-26
  Run the Sample JavaScript Directly Connected Device Applications 2-27
  Run the Sample JavaScript Enterprise Applications 2-28
Use the Android Client Software Libraries 2-29
  Set Up Your Development Environment to Use the Android Client Software Libraries 2-29
  Prepare Your Device to Use the Android Client Software Libraries 2-30
  Create the Android Client Software Library Sample Applications 2-31
  Run the Sample Android Directly Connected Device Application 2-32
  Run the Sample Android Enterprise Application 2-33
Use the Python Client Software Libraries 2-34
  Set Up Your Development Environment to Use the Python Client Software Library 2-35
  Prepare Your Device to Use the Python Client Software Library 2-35
  Run the Sample Python Directly Connected Device Application 2-36
Use the C POSIX Client Software Libraries 2-37
  Prepare Your Device to Use the C POSIX Client Software Libraries 2-37
  Building the C POSIX Client Software Library Sample Applications 2-38
  Running the C POSIX Sample Applications 2-38
  Build the C POSIX Client Software Libraries 2-39
  Setting Up Your Development Environment to use Mac OS X 2-40
  Building the C POSIX Client Software Libraries on Mac OS X 2-41
  Building the C POSIX Client Software Library Sample Applications 2-42
  Running the C POSIX Sample Applications on Mac OS X 2-42
Use the Windows Client Software Libraries 2-43
  Set Up Your Development Environment to Use the Windows Client Software Libraries 2-44
  Prepare Your Device to Use the Windows Client Software Library 2-45
  Create the Windows Client Software Library Sample Applications 2-45
  Run the Windows Sample Applications 2-46
  Build the Windows Client Software Libraries 2-47
Use the mbed Client Software Libraries 2-48
  Set Up Your Environment to Use the mbed Client Software Library 2-48
  Prepare Your Device to Use the mbed Client Software Library 2-49
Use the iOS Client Software Libraries 2-49
  Set Up Your Development Environment to Use the iOS Client Software Libraries 2-50
Run the Sample Directly Connected Device Application 2-50
Run the Sample Gateway Application 2-52
Run the Sample Enterprise Applications 2-54
Build the iOS Client Software Libraries 2-56
Network Provisioning Support in Client Libraries 2-57

3 Develop Gateway Device Software

About the Oracle Internet of Things Cloud Service Gateway Software 3-1
Gateway Security Concepts 3-2
Gateway Security Best Practices 3-6
Typical Workflow for Configuring Your Device with the Oracle Internet of Things Cloud Service Gateway Software 3-10
Install and Configure a Gateway for Application Development 3-13
Create a Self-Signed Certificate for Gateway Applications 3-15
Provision a Gateway 3-16
Configure the Gateway SDK 3-17
Create Your First Gateway Device Software 3-19
Deploy New Device Software to Your Gateway Device 3-23
Upload Device Software to the Repository 3-24
Install Device Software on a Registered Device 3-26
Manage Your Installed Device Software 3-26
Oracle IoT Gateway Software Terminology 3-28

4 Use the Gateway SDK to Integrate the Cloud Service with Java IDEs

Prepare the Gradle Build Script 4-1
Use IntelliJ IDEA with the Gateway SDK 4-2
Use Eclipse IDE with the Gateway SDK 4-5
Use NetBeans IDE with the Gateway SDK 4-9
Use a Java IDE to Build the Device Software 4-9

5 Work with Sample Adapters and Applications

Upload the Sample Device Models 5-1
Use Media with the Client Libraries 5-5
Deploy the Sample Gateway Application 5-6
Install the Bluetooth Adapter and Sample Applications in the Gateway 5-8
   Pairing a Bluetooth Device to a Gateway Device 5-11
Work with the Sample UART Device Adapter 5-14
Work with the Sample RS-232 Device Adapter 5-18
6 Integrate Oracle IoT Cloud Service with Third Party Device Management Applications

Register and Provision a Device Using Third Party Device Management Application  6-1
  Add Device Models to Oracle IoT Cloud Service 6-2
  Specify Devices as Third Party Partner Devices in Oracle IoT Cloud Service 6-4
  Register Devices with Oracle IoT Cloud Service 6-4
  Activate and Deactivate Devices in Oracle IoT Cloud Service 6-4
  Delete Devices from Oracle IoT Cloud Service 6-4

7 Troubleshoot

Troubleshoot Gateway Issues 7-1
Preface

Welcome to the developer documentation for Oracle Internet of Things Cloud Service. Use this documentation to learn how to access Oracle Internet of Things Cloud Service, manage the devices connected to an Oracle Internet of Things Cloud Service 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
• Documentation Accessibility
• Related Documents
• Conventions

Audience

This documentation is designed for application developers who create software applications that use the Oracle Internet of Things Cloud Service 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.

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:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><code>monospace</code></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
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 Internet of Things Cloud Service 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
- About the IoT Connectivity Protocols
- Methods to Connect Devices to Oracle Internet of Things Cloud Service
- About Selecting the Method to Connect Devices
- Get Started

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Security</th>
<th>Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQTT</td>
<td>Simple and lightweight IoT protocol designed for constrained devices and low network bandwidth. See mqtt.org.</td>
<td>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.</td>
<td>Small medical devices with limited network connectivity, mobile apps in mobile devices, sensors in remote locations that communicate with a gateway.</td>
</tr>
<tr>
<td>CoAP</td>
<td>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. See <a href="http://coap.technology/">http://coap.technology/</a>.</td>
<td>CoAP applies datagram transport layer security (DTLS) that's equivalent to 3072-bit RSA keys.</td>
<td>Smart energy applications and building automation applications.</td>
</tr>
<tr>
<td>WebSocket</td>
<td>A full-duplex communication channel over a TCP connection.</td>
<td>WebSocket protocol defines a ws:// and wss:// prefixes indicate a WebSocket and a WebSocket secure connection, respectively.</td>
<td>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.</td>
</tr>
<tr>
<td>XMPP</td>
<td>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 <a href="https://xmpp.org">https://xmpp.org</a></td>
<td>XMPP uses a security mechanism based on Transport Layer Security (TLS) and Simple Authentication and Security Layer (SASL).</td>
<td>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.</td>
</tr>
<tr>
<td>AMQP</td>
<td>The message queue asynchronous protocol is for communication of transactional messages between servers. See <a href="https://www.amqp.org/">https://www.amqp.org/</a></td>
<td>AMQP provides TLS/SSL and SASL for security.</td>
<td>AMQP is best used in server-based analytical functions. It’s effectively used in the banking industry.</td>
</tr>
</tbody>
</table>
Oracle IoT Cloud Service supports HTTP and MQTT.

Methods to Connect Devices to Oracle Internet of Things Cloud Service

Connect your devices to Oracle Internet of Things Cloud Service by selecting a method that works for your type of device, network, and protocol.

Typically, there are four ways to connect your device to Oracle Internet of Things Cloud Service.

- Use Oracle IoT Client Libraries that are embedded in edge applications and enable devices to connect to Oracle Internet of Things Cloud Service. 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 Oracle Internet of Things Cloud Service Gateway Software to develop your own edge gateway. Oracle Internet of Things Cloud Service Gateway Software is based on the OSGi framework and can be installed on a gateway device. This software acts like a device middle-ware that indirectly connects a wide range of sensors and devices to Oracle Internet of Things Cloud Service. This software can be remotely installed and managed from an Oracle Internet of Things Cloud Service instance.

- Use the packaged plug-ins or adapters that acquire on-premises data, and securely communicate with Oracle Internet of Things Cloud Service. 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 Internet of Things Cloud Service 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 Internet of Things Cloud Service. When Oracle Internet of Things Cloud Service sends a response either as data or as set of commands, then the connector transforms the response to the device. Some examples of packaged adapters are LoRA, M2M/telematic providers, and device partners. Custom adapters use proprietary data protocols, CSV, or XML.

About Selecting the Method to Connect Devices

You have several options to connect devices to Oracle Internet of Things Cloud Service. 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 to Oracle Internet of Things Cloud Service based on your setup and protocols.
Device Integration Models

Options to Select the Method

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Option to Connect the Devices</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices that support proprietary protocols.</td>
<td>Sensors and machines connect indirectly to Oracle Internet of Things Cloud Service through a standardized industrial gateway using proprietary protocols.</td>
<td>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 Internet of Things Cloud Service using the HTTP protocol.</td>
</tr>
<tr>
<td>Machine with a powerful in-built sensor that supports HTTP protocol.</td>
<td>Directly connect with Oracle Internet of Things Cloud Service using HTTP.</td>
<td>A device such as an on-board communicator directly exchanges messages with Oracle Internet of Things Cloud Service by using the HTTP protocol.</td>
</tr>
<tr>
<td>Devices that are already connected to an existing third-party cloud service using proprietary protocols.</td>
<td>Third-party clouds can connect with Oracle Internet of Things Cloud Service using HTTP. This results in the devices getting indirectly connected to Oracle Internet of Things Cloud Service.</td>
<td>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 Internet of Things Cloud Service using the HTTP protocol.</td>
</tr>
</tbody>
</table>
### Scenario

<table>
<thead>
<tr>
<th>Option to Connect the Devices</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices and gateways that are already connected to existing network providers that provide wireless connectivity to the devices.</td>
<td>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 Internet of Things Cloud Service using HTTP or MQTT.</td>
</tr>
<tr>
<td>Existing database with device data on events, time stamps, and alarms.</td>
<td>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 Internet of Things Cloud Service and provides the machine data for analysis.</td>
</tr>
<tr>
<td>Existing on-premises industrial gateway software</td>
<td>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 Internet of Things Cloud Service and send device messages received from the machines.</td>
</tr>
</tbody>
</table>

### Get Started

Use this information to learn how to obtain an instance of Oracle Internet of Things Cloud Service and access it. You learn about the supported browsers, how to get support and provide diagnostic information to Oracle support if you face an issue.

### Before You Use the Cloud Service

Before you use Oracle Internet of Things Cloud Service for the first time:

- Purchase subscriptions to Database Cloud Service, Oracle Java Cloud Service, and Oracle Internet of Things Cloud Service. See Buying a Metered Subscription to an Oracle Cloud Service or Buying a Non-Metered Subscription to an Oracle Cloud Service.
- (Optional) Purchase a subscription to Oracle Big Data Cloud Service - Compute Edition or Oracle Event Hub Cloud Service. These services are required for analytics functionality. See Buying a Metered Subscription to an Oracle Cloud Service or Buying a Non-Metered Subscription to an Oracle Cloud Service.
• Create the Database Cloud Service, Oracle Java Cloud Service, and Oracle Internet of Things Cloud Service instances. See Creating the Database Cloud Service, Creating the Oracle Java Cloud Service Instance, and Creating the Oracle Internet of Things Cloud Service Instance.

• (Optional) Create the Oracle Big Data Cloud Service - Compute Edition or Oracle Event Hub Cloud Service instances to enable analytics functionality. See Creating the Oracle Big Data Cloud Service - Compute Edition Instance or Creating the Oracle Internet of Things Cloud Service Instance.

Access the Management Console

Access the Oracle Internet of Things Cloud Service management console to manage your Oracle Internet of Things Cloud Service instance. This procedure assumes that you have created your Oracle Internet of Things Cloud Service instance.

1. Sign in to Oracle Cloud:
   b. Click Sign In.
   c. Select the data center where your services are located in the Select Data Center list.
   d. Click My Services.
   e. Enter the identity domain provided in your welcome email and then click Go.
   f. Enter your user name and your password and then click Sign In.

2. Select IoT Enterprise in the Cloud Services area.

3. Click Open Service Console.

4. Click the Manage this service ( ) icon for the Oracle Internet of Things Cloud Service instance you want to access and then select Management Console.

Supported Browsers

This table lists the supported web browsers for Oracle Internet of Things Cloud Service.

<table>
<thead>
<tr>
<th>Web Browser</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Internet Explorer</td>
<td>11</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>43</td>
</tr>
<tr>
<td>Mozilla Firefox</td>
<td>37, 38</td>
</tr>
<tr>
<td>Apple Safari</td>
<td>8.0</td>
</tr>
</tbody>
</table>

Supported Platforms

The Oracle Internet of Things Cloud Service 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:

- Review Troubleshoot.

- 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.


Locating 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 Finding Diagnostic Information to Help with Troubleshooting.

<table>
<thead>
<tr>
<th>Diagnostic Information</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Version Number</td>
<td>Record the version number of the Oracle Cloud instance.</td>
<td>17.1.5.0</td>
</tr>
<tr>
<td>Client Platform Version</td>
<td>Record the client platform version.</td>
<td>Client library information for each platform.</td>
</tr>
<tr>
<td>Java or Android Compiler Version</td>
<td>Record the version number of the device Java or Android compiler.</td>
<td>Android 7.0</td>
</tr>
<tr>
<td>Error Message</td>
<td>Record the error message or attach a screen image.</td>
<td>Operation failed. Cause:</td>
</tr>
</tbody>
</table>
|                                |                                                                             | {"type":"https://www.w3.org/Protocols/rfc2616/rfc2616-sect10.html","title":"Project was not found:
ProjectId=iotAppId=0-AD,serviceId=0-AD"","status":"404"}          |
<p>| Oracle Internet of Things Cloud Service URL | Record the URL of the Oracle Internet of Things Cloud Service instance. | <a href="https://targetcloudinstance.com">https://targetcloudinstance.com</a>                                        |
| 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                                                                |</p>
<table>
<thead>
<tr>
<th>Diagnostic Information</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen Images</td>
<td>Attach relevant screen images of the error message or the page where the error occurred.</td>
<td>Include a screen image with annotations.</td>
</tr>
<tr>
<td>Navigation Path</td>
<td>Provide the navigation path to the page where the error occurred.</td>
<td>Device Model &gt; Device Model Details</td>
</tr>
</tbody>
</table>
Develop Device Software Using the Client Software Libraries

Oracle Internet of Things Cloud Service client software libraries are provided to simplify how the devices work with Oracle Internet of Things Cloud Service.

Topics

• Device Virtualization
• Client Library Best Practices
• Use the JavaScript Client Software Libraries
• Use the Java SE Client Software Libraries
• Use the Android Client Software Libraries
• Use the C POSIX Client Software Libraries
• Use the Windows Client Software Libraries
• Use the mbed Client Software Libraries
• Build the iOS Client Software Libraries
• Network Provisioning Support in Client Libraries

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
Client Library Best Practices

Make sure that your application using the Oracle Internet of Things Cloud Service 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
- Prepare an Embedded Device to Use the Java SE Client Software Library
- Create the Java SE Client Software Library Sample Applications
- Use the Provisioning Tool to Create the Truststore
- Run the Sample Java SE Directly Connected Device Applications
- Run the Sample Java SE Gateway Application
- Run the Sample Java SE Gateway Application Using Apache Felix
- Run the Sample Java SE Enterprise Applications
- Build the Java SE Client Software Libraries

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 Internet of Things Cloud Service, 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.
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](https://gradle.org).

3. Open a web browser and browse to the Oracle Internet of Things Cloud Service 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:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-csl-javase-bin-&lt;version&gt;.zip</td>
<td>Contains the Java SE client software library binary files.</td>
</tr>
<tr>
<td>iotcs-csl-javase-src-&lt;version&gt;.zip</td>
<td>Contains the Java SE client software library binary file source code.</td>
</tr>
<tr>
<td>iotcs-csl-javase-samples-&lt;version&gt;.zip</td>
<td>Contains the Java SE sample applications.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Filename</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-csl-javase-bin-&lt;version&gt;.zip</td>
<td>iotcs/csl/javase/bin</td>
</tr>
<tr>
<td>Filename</td>
<td>Directory</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>iotcs-csl-javase-src-&lt;version&gt;.zip</td>
<td>iotcs/csl/javase/src</td>
</tr>
<tr>
<td>iotcs-csl-javase-samples-&lt;version&gt;.zip</td>
<td>iotcs/csl/javase/samples</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>System Variable Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_HOME</td>
<td>/home/&lt;user&gt;/iotcs/csl/javase/</td>
</tr>
<tr>
<td>GRADLE_HOME</td>
<td>/opt/gradle-&lt;version&gt;</td>
</tr>
<tr>
<td>JAVA_HOME</td>
<td>/usr/java/jdk&lt;version&gt;</td>
</tr>
</tbody>
</table>

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:

```bash
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.

```bash
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:

```properties
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.


12. Associate the humidity and temperature sensor device models with the JavaSECLApp application. See Associate a Device Model with an IoT Application.

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 Internet of Things Cloud Service instance.
   b. Click the Menu (≡) icon adjacent to the Oracle Internet of Things Cloud Service 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 (редактировать) icon.
   f. Click the Overview tab.
      If you are using Oracle Internet of Things Cloud Service 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:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Hardware</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux/x86 desktop</td>
<td>x86 Linux Small Footprint – Headless</td>
<td>ejdk-&lt;version&gt;-linux-i586.tar.gz</td>
</tr>
<tr>
<td>ARM v6/v7 Linux — Hard Floating Point</td>
<td>ARM v6/v7 Linux - VFP, HardFP ABI, Little Endian 1</td>
<td>ejdk-&lt;version&gt;-linux-armv6-vfp-hflt.tar.gz</td>
</tr>
</tbody>
</table>

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

   ```bash
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:

   ```bash
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.


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 Internet of Things Cloud Service version 16.4.1 or earlier, use the provisioning tool to create the trusted assets truststore. The truststore contains the Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service 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 Internet of Things Cloud Service 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
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_integration_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_integration_ID.jks` is created. The trusted assets truststore contains the Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service.

**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 Internet of Things Cloud Service. See Upload the Sample Device Models.
3. Register the device and download the provisioning file. See Registering and Activating Devices.
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-20180813.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.

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 Internet of Things Cloud Service 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 Internet of Things Cloud Service. 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 Internet of Things Cloud Service 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:**

```
```

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,"minThreshold"=0,"maxThreshold"=65
Tue Feb 9 16:17:58 EST 2016 : 0-HE : Set : "humidity"=86
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-HA : Alert : "humidity"=91 (tooHumidAlert)
Tue Feb 9 16:18:08 EST 2016 : 0-HE : Alert : "humidity"=91
Tue Feb 9 16:18:08 EST 2016 : 0-HA : Set : "temp"=66.0,"maxTemp"=66.0
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 Internet of Things Cloud Service 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 Internet of Things Cloud Service. 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.

```java
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: (-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 {
    }
}
```

Chapter 2
Use the Java SE Client Software Libraries
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:

<table>
<thead>
<tr>
<th>Sample Name</th>
<th>MANIFEST.MF</th>
</tr>
</thead>
</table>
   | 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 |
   | 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.SampleActivator  
                                                     | Import-Package: org.osgi.framework, oracle.iot.client.enterprise, oracle.iot.client |
**Sample Name**  | **MANIFEST.MF**  
--- | ---  
`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.SampleActivator  

---

**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
   ```

<table>
<thead>
<tr>
<th><code>sample_name</code></th>
<th><code>sample_args</code></th>
</tr>
</thead>
</table>
| `DirectlyConnectedDeviceSample` | `provisioning_file_name`  
|                               | `provisioning_file_password`                     |
| `GatewayDeviceSample`         | `provisioning_file_name`  
|                               | `provisioning_file_password`  
<p>|                               | <code>temperature_sensor_endpointID</code>                 |
|                               | <code>humidity_sensor_endpointID</code>                     |</p>
<table>
<thead>
<tr>
<th>sample_name</th>
<th>sample_args</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnterpriseClientSample</td>
<td>provisioning_file_name provision_file_password provision_file_name deviceID</td>
</tr>
<tr>
<td></td>
<td>(deviceID) provision_file_name provision_file_password deviceID (reset</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>java</td>
</tr>
<tr>
<td></td>
<td>com.oracle.iot.sample.EnterpriseClientSample</td>
</tr>
<tr>
<td></td>
<td>java</td>
</tr>
<tr>
<td></td>
<td>com.oracle.iot.sample.EnterpriseClientSample</td>
</tr>
<tr>
<td></td>
<td>provisioning_file_name provision_file_password deviceID maxThreshold minThreshold</td>
</tr>
<tr>
<td>ext.DirectlyConnectedDeviceSample</td>
<td>provisioning_file_name provision_file_password</td>
</tr>
<tr>
<td>ext.GatewayDeviceSample</td>
<td>provisioning_file_name provision_file_password deviceID</td>
</tr>
<tr>
<td></td>
<td>temperature_sensor_endpointID humidity_sensor_endpointID</td>
</tr>
</tbody>
</table>

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-20160212.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 virtual temperature sensor 0-2UJQ
Created virtual 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,"minThreshold"=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 Internet of Things Cloud Service.

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 Internet of Things Cloud Service. See Upload the Sample Device Models.

3. Register the device and download the provisioning file.

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-20180813.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 DirectlyConnectedDeviceSample 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-20180813.jar
   ```
com.oracle.iot.sample.EnterpriseClientSample app_integration_ID-provisioning-file.conf Password123 0-HA,0-HE,0-BQ

Note:
Replace \texttt{app\_integration\_ID-provisioning-file.conf} with the name of the enterprise integration provisioning file and \texttt{app\_integration2\_ID-provisioning-file.conf} with the integration provisioning file for the control feature of the \texttt{EnterpriseClientSample} application. Replace \texttt{Password123} with the password used to secure the provisioning files.

Output similar to the following appears:

Press enter to exit.

\begin{verbatim}
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
\end{verbatim}

8. Run this command to set the maximum humidity threshold of the \texttt{DirectlyConnectedDeviceSample} device:

\begin{verbatim}
\end{verbatim}

This message appears:

\begin{verbatim}
Tue Feb 9 18:24:56 EST 2016 : 0-BQ : Set : "maxThreshold"=67 ..
... [Humidity readings] ...
Done.
\end{verbatim}

Output similar to the following appears on the device console:

\begin{verbatim}
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
\end{verbatim}
9. Run this command to reset the EnterpriseClientSample device temperature sensor:

```java
java -cp $CL_HOME/samples/build/libs/iotcs-csl-samples.jar:$CL_HOME/lib/enterprise-library.jar:$CL_HOME/lib/json-20180813.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
java -cp $CL_HOME/samples/build/libs/iotcs-csl-samples.jar:$CL_HOME/lib/enterprise-library.jar:$CL_HOME/lib/json-20180813.jar
com.oracle.iot.sample.EnterpriseClientSample app_integration2_ID-provisioning-file.conf Password123 0-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
java -cp $CL_HOME/samples/build/libs/iotcs-csl-samples.jar:$CL_HOME/lib/enterprise-library.jar:$CL_HOME/lib/json-20180813.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 ..
°C,"minThreshold"=0,"maxThreshold"=65,"temp"=64.99,"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:

```
```

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
```

Output similar to the following appears:

Creating the gateway instance...

Created virtual temperature sensor 0-HY
Created virtual 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,"minThreshold"=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
```

Output similar to the following appears:

Creating the gateway instance...

Created virtual temperature sensor 0-HY
Created virtual 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,"minThreshold"=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.
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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gradle</td>
<td>Builds the <code>device-library.jar</code> and <code>enterprise-library.jar</code> library files and places them in the <code>build/libs</code> folder.</td>
</tr>
<tr>
<td>gradle deviceClientJar</td>
<td>Compiles the code specific to a device client and creates <code>build/libs/device-library.jar</code></td>
</tr>
<tr>
<td>gradle enterpriseClientJar</td>
<td>Compiles the code specific to an enterprise client and creates <code>build/libs/enterprise-library.jar</code></td>
</tr>
<tr>
<td>gradle -PWITH_VIRTUALIZATION=false</td>
<td>Compiles the code specific to a device client without virtualization support. This reduces the size of the <code>device-library.jar</code> file.</td>
</tr>
<tr>
<td>gradle -PWITH_ENUMERATION=false</td>
<td>Compiles the code specific to an enterprise client without resource and message enumeration support. This reduces the size of the <code>enterprise-library.jar</code> file.</td>
</tr>
<tr>
<td>gradle doc</td>
<td>Generates the javadoc files and places them in the <code>build/docs</code> folder.</td>
</tr>
</tbody>
</table>
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
• Prepare Your Device to Use the JavaScript Client Software Library
• Run the Sample JavaScript Directly Connected Device Applications
• Run the Sample JavaScript Enterprise Applications

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 Internet of Things Cloud Service, 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 Internet of Things Cloud Service. See Upload the Sample Device Models.
3. Open a web browser and browse to the Oracle Internet of Things Cloud Service 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:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-csl-js-bin-&lt;version&gt;.zip</td>
<td>iotcs/csl/js/bin</td>
</tr>
<tr>
<td>iotcs-csl-js-src-&lt;version&gt;.zip</td>
<td>iotcs/csl/js/src</td>
</tr>
<tr>
<td>iotcs-csl-js-samples-&lt;version&gt;.zip</td>
<td>iotcs/csl/js/samples</td>
</tr>
</tbody>
</table>

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 Internet of Things Cloud Service instance.

b. Click the Menu (≡) icon adjacent to the Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service 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. To optionally use the MQTT protocol instead of HTTP, run this command: npm install -g mqtt.
Run this command to set the `NODE_PATH` environment variable:

```bash
SET NODE_PATH = "%C:\username\AppData\Roaming\npm\node_modules%"
```

10. Run this command to move to the `node-forge` directory:

```bash
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:

```bash
node node_modules\requirejs\bin\r.js -o minify.js optimize=none out=js/forge.bundle.js
```

11. Run this command to move to the `js` directory:

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

12. Create a new folder named `External`.

13. 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.

14. 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.

15. Open a text editor and the open the `EnterpriseClient.html` file located at this path: `C:\Users\username\Documents\iot\csl\js\samples`.

16. 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.

17. Locate the `src` value and make sure the value is `"../modules/enterprise-library.web.js"`.

18. Save and close the `EnterpriseClient.html` file.

19. 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 Internet of Things Cloud Service.

1. Download and install Node.js with npm (v6.9.1).

2. Open a command prompt and run this command to move to the `samples` directory:

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

3. Run this command to activate a registered device:

```bash
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 Internet of Things Cloud Service:

   a. Open the Oracle Internet of Things Cloud Service Management Console. See Access the Management Console.
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 Internet of Things Cloud Service.

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 Internet of Things Cloud Service:
   a. Open the Oracle Internet of Things Cloud Service Management Console. See Access the Management Console.
   b. Click the Menu (≡) icon adjacent to the Oracle Internet of Things Cloud Service 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
• Prepare Your Device to Use the Android Client Software Libraries
• Create the Android Client Software Library Sample Applications
• Run the Sample Android Directly Connected Device Application
• Run the Sample Android Enterprise Application

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

Before you can develop applications that let your devices communicate with Oracle Internet of Things Cloud Service, 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 Internet of Things Cloud Service. See Upload the Sample Device Models.
3. Download and install Gradle. Versions 2.2.1 to 2.13 are 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 Internet of Things Cloud Service 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:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-csl-android-bin-&lt;version&gt;.zip</td>
<td>iotcs/csl/android/bin</td>
</tr>
<tr>
<td>iotcs-csl-android-src-&lt;version&gt;.zip</td>
<td>iotcs/csl/android/src</td>
</tr>
<tr>
<td>iotcs-csl-android-samples-&lt;version&gt;.zip</td>
<td>iotcs/csl/android/samples</td>
</tr>
</tbody>
</table>

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 Internet of Things Cloud Service instance.
   b. Click the Menu (≡) icon next to the Oracle Internet of Things Cloud Service 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 (-pencil) icon.
   f. Click the Overview tab.
      If you are using Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service 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\android.
   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\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\android.
   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\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\android.

c. Run this command: gradle buildApkDCDDCLNoAssets.

2. Confirm the Android application package (APK) was created:
   a. Browse to C:\Users\user-name\Documents\iotcs\csl\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\iotcs\csl\android.
   c. Run this command: gradle buildApkECLNoAssets.

4. Confirm the Android application package (APK) was created:
   a. Browse to C:\Users\user-name\Documents\iotcs\csl\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 Internet of Things Cloud Service. 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\csi\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**.
   c. Review the messages displayed on the emulator screen.

7. Confirm data is being sent from the application to Oracle Internet of Things Cloud Service:
   a. Open the Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service.

---

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 Internet of Things Cloud Service. 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:

   ```bash
   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 Internet of Things Cloud Service:

   a. Open the Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service.

---

**Use the Python Client Software Libraries**

You can develop IoT applications using the Oracle Internet of Things Cloud Service 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
- Prepare Your Device to Use the Python Client Software Library
- Run the Sample Python Directly Connected Device Application

**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 Internet of Things Cloud Service, 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 Internet of Things Cloud Service. See Upload the Sample Device Models.
3. Create a new 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 Internet of Things Cloud Service client software libraries download site.
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. Download and install 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): `python -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: `python -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. Move the Python Client Libraries into the PVE.

11. Run these commands to install the libraries:

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

12. 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 Internet of Things Cloud Service.

1. Open a command prompt and browse to your iotcs directory, copy the sample Python code file in :

2. Copy the sample Python code file DirectlyConnectedDeviceSample.py located in the iotcs\sample to that directory.

3. Copy the provisioning file obtained in Set Up Your Development Environment to Use the Python Client Software Library to the same directory as the sample code file.

4. Run the following command in the same directory:

   python DirectlyConnectedDeviceSample.py <provisioner file> <passphrase>

5. Deactivate the virtual environment by using this command:

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

6. Confirm the device is sending messages to Oracle Internet of Things Cloud Service:
   a. Open the Oracle Internet of Things Cloud Service Management Console. See Access the Management Console.
   b. Click the Menu (☰) icon adjacent to the Oracle Internet of Things Cloud Service 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.

### 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
- Building the C POSIX Client Software Library Sample Applications
- Running the C POSIX Sample Applications
- Build the C POSIX Client Software Libraries
- Setting Up Your Development Environment to use Mac OS X
- Building the C POSIX Client Software Libraries on Mac OS X
- Building the C POSIX Client Software Library Sample Applications
- Running the C POSIX Sample Applications on Mac OS X

### 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 Internet of Things Cloud Service 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 Internet of Things Cloud Service. See [Setting Up a Raspberry Pi Device to Run Oracle IoT Cloud Service Client Software](#).

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 **l2 Change Timezone**.

   d. Select your geographical area.

   e. Select a city nearest to your location.

   f. Select **Finish**.
**Building 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:
   ```shell
cd /home/pi/iotcs/csl/posix/samples/make
   ``
2. Run this command to create the directly connected device sample application:
   ```shell
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:
   ```shell
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.

**Running 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 Internet of Things Cloud Service. 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 Internet of Things Cloud Service.

1. Build the samples. See [Building 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 Internet of Things Cloud Service. 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:
   \texttt{export IOTCS\_OS\_NAME="Raspbian GNU/Linux"}.

5. Run this command to set the operating system version variable:
   \texttt{export IOTCS\_OS\_VERSION="8"}.

6. Run this command to move to the build/sample directory:
   \texttt{cd /home/pi/iotcs/csl/posix/build/sample/arm/ts\_md\_vs}.

7. Run this command to run the directly connected device sample application:
   \texttt{./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 Internet of Things Cloud Service:
   a. Open the Oracle Internet of Things Cloud Service Management Console. See Access the Management Console.
   b. Click the Menu ( ) icon adjacent to the Oracle Internet of Things Cloud Service 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:
   \texttt{cd /home/pi/iotcs/csl/posix/build/sample/arm/ts\_md\_vs\_gw}.

10. Run this command to run the gateway sample application:
    \texttt{./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 Internet of Things Cloud Service.

### 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 the Oracle Internet of Things Cloud Service 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:
   \texttt{unzip iotcs-csl-posix-src-<release\_number>.zip}.

4. Run this command to install Doxygen:
   \texttt{sudo apt-get install doxygen}.

5. Run this command to move to the make directory:
   \texttt{cd /home/pi/iotcs/csl/posix/make}.

6. Run this command to compile the source code and build the libdeviceclient.a file:
   \texttt{make clean all LIB\_CFG=ts}.
7. Run this command to move to the samples/make directory: `cd /home/pi/iotcs/csli/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.

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

---

**Setting 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 your 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 Internet of Things Cloud Service 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 Internet of Things Cloud Service. 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:

   ```bash
   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 1.0.2*, where * is a lower case letter.

Building 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 Setting Up Your Development Environment to use Mac OS X.

2. Open a web browser and browse to the Oracle Internet of Things Cloud Service 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>`:

<table>
<thead>
<tr>
<th>Library Option</th>
<th>Messaging Thread Safety</th>
<th>Message Dispatcher</th>
<th>Virtualization Support</th>
<th>Gateway</th>
</tr>
</thead>
<tbody>
<tr>
<td>nots</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>nots_gw</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>ts</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>ts_gw</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>ts_md</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>ts_md_gw</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>ts_md_vs</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>ts_md_vs_gw</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
Building 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 Setting 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 directory `../build/sample/x86/ts_md_vs`.
   
   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 directory `../build/sample/x86/ts_md_vs_gw`.
   
   The value of `<library option>` must be one of the following:
   - `nots_gw`
   - `ts_gw`
   - `ts_md_gw`
   - `ts_md_vs_gw`

Running 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 Internet of Things Cloud Service.

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

2. Build the libraries. See Building the C POSIX Client Software Libraries on Mac OS X.

3. Build the samples. See Building 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 Internet of Things Cloud Service:
   a. Open the Oracle Internet of Things Cloud Service Management Console. See Access the Management Console.
   b. Click the `Menu ()` icon adjacent to the Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service.

### 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
- Prepare Your Device to Use the Windows Client Software Library
- Create the Windows Client Software Library Sample Applications
- Run the Windows Sample Applications
- Build the Windows Client Software Libraries
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 Internet of Things Cloud Service, 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 Internet of Things Cloud Service 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 Internet of Things Cloud Service. See Upload the Sample Device Models.

7. Open a command prompt and run this command to move to the iotcs/cs1/windows/bin directory: `cd iotcs/cs1/windows/bin`.

8. Run this command to download the OpenSSL file: `wget https://www.openssl.org/source/openssl-1.0.2h.tar.gz`.
   This command example assumes the wget utility is installed.

9. Run this command to extract the openssl-1.0.2h.tar.gz file: `tar -xvf openssl-1.0.2h.tar.gz`.

10. Run this command to move to the openssl-1.0.2h folder: `cd <filepath>/openssl-1.0.2h`.

11. Run this command to determine the path to the Visual Studio environment variable: `env | findstr -i comntools`.
   The env command should return a response similar to: `VS120COMNTOOLS=C:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\Tools`.

12. Run this batch script to set the path and environment variables for command-line builds: `"%VS120COMNTOOLS%\vsvars32.bat"`.

13. Run this command to configure OpenSSL for Windows: `perl Configure VC-WIN32 no-asn`.

14. Run this command to create the Windows files: `ms/do_ms.bat`.

15. Run this command to build the OpenSSL libraries: `nmake /nologo -f ms/nt.mak install`.

16. Run this command to copy the generated libraries to the iotcs/cs1/windows/bin folder: `copy /y out32/*.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\".
   ```


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.


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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Where Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cl</td>
<td><code>&lt;CC_CFG_Option&gt;</code></td>
<td>Builds the client software library on Windows computers. This is the default compiler.</td>
</tr>
<tr>
<td>gcc</td>
<td><code>&lt;CC_CFG_Option&gt;</code></td>
<td>Builds the client software library on Linux computers and adds the GNU Compiler Collection (GCC).</td>
</tr>
<tr>
<td>Command</td>
<td>Where Used</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>nots</td>
<td>&lt;LIB_CFG_Option&gt;</td>
<td>Builds the directly connected device sample application with these options: messaging thread safety=false, messaging dispatcher=false, and virtualization support=false.</td>
</tr>
<tr>
<td>ts</td>
<td>&lt;LIB_CFG_Option&gt;</td>
<td>Builds the directly connected device sample application with these options: messaging thread safety=true, messaging dispatcher=false, and virtualization support=false.</td>
</tr>
<tr>
<td>ts_md_gw</td>
<td>&lt;LIB_CFG_Option&gt;</td>
<td>Builds the gateway device sample application with these options: messaging thread safety=true, messaging dispatcher=true, virtualization support=false, and indirect activation=true.</td>
</tr>
<tr>
<td>ts_md_vs</td>
<td>&lt;LIB_CFG_Option&gt;</td>
<td>Builds a directly connected device sample application with these options: messaging thread safety=true, messaging dispatcher=true, and virtualization support=true.</td>
</tr>
<tr>
<td>ts_md_vs_gw</td>
<td>&lt;LIB_CFG_Option&gt;</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 Internet of Things Cloud Service. 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 Internet of Things Cloud Service.

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 Internet of Things Cloud Service. 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 Internet of Things Cloud Service:
   a. Open the Oracle Internet of Things Cloud Service Management Console. See Access the Management Console.
   b. Click the **Menu** icon adjacent to the Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service 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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Where Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cl</td>
<td><code>&lt;CC_CFG_Option&gt;</code></td>
<td>Builds the client software library on Windows computers. This is the default compiler.</td>
</tr>
<tr>
<td>gcc</td>
<td><code>&lt;CC_CFG_Option&gt;</code></td>
<td>Builds the client software library on Linux computers and adds the GNU Compiler Collection (GCC).</td>
</tr>
<tr>
<td>all</td>
<td><code>&lt;LIB_CFG_Option&gt;</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
Use the mbed Client Software Libraries

To connect an ARM mbed device to Oracle Internet of Things Cloud Service, you create an application that includes the mbed Client Software Library. To compile the mbed application, you must download and compile the mbed rtos and mbed TLS libraries separately as a static library `libmbedrtos.a`. You link this static library and your compiled mbed application to generate a binary file which is uploaded and run by mbed.

Topics

- Set Up Your Environment to Use the mbed Client Software Library
- Prepare Your Device to Use the mbed Client Software Library

Set Up Your Environment to Use the mbed Client Software Library

These items are required to let your ARM mbed devices communicate with Oracle Internet of Things Cloud Service:

- A desktop or laptop computer.
- An mbed enabled platform (board) with the latest firmware installed.
- A micro USB cable.
- A microSD card.
- A microSD to SD memory card adapter.

---

### Command Table

<table>
<thead>
<tr>
<th>Command</th>
<th>Where Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nots</code></td>
<td><code>&lt;LIB_CFG_Option&gt;</code></td>
<td>Builds the directly connected device sample application with these options: messaging thread safety=false, messaging dispatcher=false, and virtualization support=false.</td>
</tr>
<tr>
<td><code>ts</code></td>
<td><code>&lt;LIB_CFG_Option&gt;</code></td>
<td>Builds the directly connected device sample application with these options: messaging thread safety=true, messaging dispatcher=false, and virtualization support=false.</td>
</tr>
<tr>
<td><code>ts_md_gw</code></td>
<td><code>&lt;LIB_CFG_Option&gt;</code></td>
<td>Builds the gateway device sample application with these options: messaging thread safety=true, messaging dispatcher=true, virtualization support=false, and indirect activation=true.</td>
</tr>
<tr>
<td><code>ts_md_vs</code></td>
<td><code>&lt;LIB_CFG_Option&gt;</code></td>
<td>Builds a directly connected device sample application with these options: messaging thread safety=true, messaging dispatcher=true, and virtualization support=true.</td>
</tr>
<tr>
<td><code>ts_md_vs_gw</code></td>
<td><code>&lt;LIB_CFG_Option&gt;</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

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

---

**Oracle**

2-48
• A Linux-like environment for Windows. For example, Cygwin.
• A GNU ARM Embedded Toolchain greater than 4.9-2015-q3-update installed on the desktop or laptop computer. The project files are available for download here: https://launchpad.net/gcc-arm-embedded/+download.
• GNU Make installed on the desktop or laptop computer.

Prepare Your Device to Use the mbed Client Software Library

A device must be configured correctly and meet the minimum hardware requirements to successfully install the Oracle Internet of Things Cloud Service mbed client software libraries.

1. Make sure the hardware prerequisites are met before you install the Oracle Internet of Things Cloud Service client software library on your target device. For a list of supported platforms, see Oracle IoT Cloud Service Client Software Certified System Configurations.

2. Create a folder named third_party on the computer on which the mbed client software libraries will be installed.

3. Download and extract these mbed files to the third_party folder:

4. Open a command prompt and use the cd command to browse to the location of the script.sh file.

5. Run this command to run the script and create a folder named os.:/script.sh.

6. Open a web browser and browse to the Oracle Internet of Things Cloud Service client software libraries download site.

7. Scroll to C Client Software Libraries and download the Source Code zip file from the MBED table.

8. Extract the mbed source code client software library file.

9. Move the contents of the os folder into the mbed folder. The path for the mbed folder is typically iotcs/csl/mbed/src/mbed.

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
• Run the Sample Directly Connected Device Application
• Run the Sample Gateway Application
• Run the Sample Enterprise Applications
• Build the iOS Client Software Libraries

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 Internet of Things Cloud Service, 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 Internet of Things Cloud Service 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:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-csl-ios-bin-&lt;version&gt;.zip</td>
<td>Contains the iOS client software library binary files.</td>
</tr>
<tr>
<td>iotcs-csl-ios-samples-&lt;version&gt;.zip</td>
<td>Contains the iOS sample applications.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Filename</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-csl-ios-bin-&lt;version&gt;.zip</td>
<td>iotcs/csl/ios/bin</td>
</tr>
<tr>
<td>iotcs-csl-ios-samples-&lt;version&gt;.zip</td>
<td>iotcs/csl/ios/samples</td>
</tr>
</tbody>
</table>

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 Internet of Things Cloud Service. The attributes, actions, and alerts of the sample temperature and humidity
sensors are specified in device models which you upload to Oracle Internet of Things Cloud Service.

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 Internet of Things Cloud Service.

1. Upload the humidity and temperature sensor device models to Oracle Internet of Things Cloud Service. 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 Internet of Things Cloud Service. 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 Internet of Things Cloud Service.

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 Internet of Things Cloud Service.
1. Upload the humidity and temperature sensor device models to Oracle Internet of Things Cloud Service. 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/cs1/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 Internet of Things Cloud Service.

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 Internet of Things Cloud Service. 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 Internet of Things Cloud Service. 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 Internet of Things Cloud Service instance.
   b. Click the Menu icon adjacent to the Oracle Internet of Things Cloud Service 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 Internet of Things Cloud Service 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:

![Enterprise Client Sample]

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 Internet of Things Cloud Service 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build device</td>
<td>Builds the iOS device library. The library is created in the <code>iotcs/csl/ios/lib/DeviceLib.framework</code> directory.</td>
</tr>
<tr>
<td>build enterprise</td>
<td>Builds the iOS enterprise library. The library is created in the <code>iotcs/csl/ios/lib/EnterpriseLib.framework</code> directory.</td>
</tr>
</tbody>
</table>

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.

Gateway Bootstrapper and Network Provisioner

• Bootstrapper Usage:

  Note:
The JAVA_HOME and GATEWAY_HOME environment variables should be set
appropriately before using the bootstrapper command.

  gateway.[sh|bat] [-i provisioned_file -p password -t
code_truststore_path/file_name
     -w code_truststore_password -a certificate_pem 1 -a certificate_pem
N]

Where:

- i provisioned_file is the path/file name to use for the network provisioned
  trusted assets store file.
-p password is a password phrase used to protect the integrity of the trusted assets store.

-t code_truststore_path/file_name is the path/file name of the code truststore for importing certificates into the trusted assets store. This path is relative to the GATEWAY_HOME directory.

-w code_truststore_password is the password for the code truststore file.

-a certificate_pem is the path/file name of the certificate, in PEM format, to import into the trusted assets store. More than one certificate may be supplied.

For example:

./bin/gateway.sh -i config/trustedAssetsStore.upf -p password -t config/code.truststore -w iotgateway -a iotappdev-cert.pem

- 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.
Develop Gateway Device Software

Device software developed using the Oracle IoT Cloud Service Gateway Software can either be a device adapter that allows a native device to work with the Oracle IoT Cloud Service Gateway Software, or a device application that can manage the devices that have been configured with the Oracle IoT Cloud Service Gateway Software. Sample device adapters and device applications are provided with the Oracle IoT Cloud Service Gateway Software bundle.

Topics:
• About the Oracle Internet of Things Cloud Service Gateway Software
• Gateway Security Concepts
• Gateway Security Best Practices
• Typical Workflow for Configuring Your Device with the Oracle Internet of Things Cloud Service Gateway Software
• Install and Configure a Gateway for Application Development
• Create a Self-Signed Certificate for Gateway Applications
• Provision a Gateway
• Configure the Gateway SDK
• Create Your First Gateway Device Software
• Deploy New Device Software to Your Gateway Device
• Install Device Software on a Registered Device
• Upload Device Software to the Repository
• Manage Your Installed Device Software
• Oracle IoT Gateway Software Terminology

About the Oracle Internet of Things Cloud Service Gateway Software

When installed on a gateway the Oracle Internet of Things Cloud Service Gateway Software lets you connect multiple devices to Oracle Internet of Things Cloud Service through a variety of communication protocols.

A gateway is a device that receives data from sensors or legacy devices and then forwards the information to Oracle Internet of Things Cloud Service. These sensors or legacy devices are also known as indirectly connected devices.

What is the Oracle Internet of Things Cloud Service Gateway Software?

A gateway provides several benefits:
• Small sensors or legacy devices can directly transmit their data to a nearby gateway instead of to the cloud, reducing their power consumption and increasing the sensors’ battery life.

• A gateway communicates with different types of devices using different protocols and then sends the data to Oracle Internet of Things Cloud Service using a standard protocol.

• A gateway acts as a filter for the huge amount of data sent by the devices, processing the data and sending only relevant information to the cloud. Therefore, the processing and storage services of Oracle Internet of Things Cloud Service is utilized optimally.

• The response time for the sensors is considerably reduced. The nearby gateway receives the sensor data, processes it, and sends relevant commands back to the sensors.

• Gateways are highly secure and they also help secure the sensors and devices that are connected to them.

The Oracle Internet of Things Cloud Service Gateway Software is Java-based software that lets you connect multiple devices to Oracle Internet of Things Cloud Service and supports a variety of communication protocols. It is Open Service Gateway Initiative (OSGi)-based middleware for gateway devices. It includes an extensible device adapter framework that has built-in support for the industry communication protocols such as HTTPS and MQTT, and provides remote software management capabilities.

The Oracle Internet of Things Cloud Service Gateway Software manages the information moving in both directions and provides additional security to the IoT network and the data in the network. The Oracle Internet of Things Cloud Service Gateway Software acts like an application server for the gateway device. It allows remote management and deployment of a rich set of applications with added security.

Gateway Security Concepts

The Oracle Internet of Things Cloud Service Gateway Software provides security by using certificates while configuring the gateway runtime, the gateway SDK, the gateway bundles, and the application bundles. To better understand how gateway security is configured and managed in the various processes, review the following illustrations.

If needed, review the Oracle IoT Gateway Software Terminology to better understand the gateway security concepts.

Security Configurations and Management

While configuring truststores, you create and install certificates for the user-defined applications, in the truststore. You use the keytool utility that involves four steps as illustrated in the diagram.
In the image, keytool is used in the steps to:

1. Generate the IoT Application Developer certificate and save it in the keystore.
2. Export the IoT Application Developer certificate from the keystore into a file.
3. Import the certificate from the file into the truststore.
4. List all the certificates in the truststore and then confirm that the IoT Application Developer certificate exists in the list.

The gateway bundles that are located in the gateway directory are already signed with the IoT Gateway certificate. The IoT Gateway certificate is stored in GATEWAY_HOME/config/code.truststore and ships with the Oracle Internet of Things Cloud Service Gateway Software.

When you build an user-defined application, a certificate is used for signing it, and when you run it, the application certificate is verified by the Oracle Internet of Things Cloud Service Gateway Software. The application certificate can be any certificate. If your company has its own certificate, you can use it.

For provisioning the gateway, you register your gateway device in Oracle Internet of Things Cloud Service, download the provisioning file, and then use the unified provisioner tool to update the provisioning file with the certificate information. The updated provisioning file is known as the trusted assets store. The steps of this process are illustrated in the diagram.
The steps illustrated in the image are:

1. Enter the device details to register the gateway device on Oracle Internet of Things Cloud Service, download the provisioning file that is generated, and save it in your gateway device's config folder.

2. Run the uprovisioner tool that fetches the IoT Application Developer and IoT Gateway certificates from the code.truststore keystore and updates the provisioning file with the certificates. You can verify that this updated provisioning file (the trusted assets store) contains the Oracle Internet of Things Cloud Service URL and its SSL certificate, the device registration ID and its shared secret, and the IoT Application Developer and IoT Gateway certificates.

You can use more than one application certificate, where some application bundles are signed by one application certificate and other bundles are signed by another certificate. Ensure that all the certificates are provisioned to the trusted assets store.

Updating the security properties file typically follows the gateway provisioning process. You update the GATEWAY_HOME/config/security.properties file with the password and path of the trusted assets store relative to the GATEWAY_HOME/config directory. Ensure that the password is the one that you entered when you downloaded the provisioning file from Oracle IoT Cloud Service. In the properties file, you modify the following fields:

```
trusted.assets.password=password of the provisioning file
trusted.assets.path=name_of_provisioning_file
```

While building the application, the Oracle Internet of Things Cloud Service Gateway SDK signs your application bundles with the IoT Developer certificate.
When your sample application is built by the Oracle Internet of Things Cloud Service Gateway SDK, it obtains the IoT Application Developer certificate from `code.keystore`, generates the application runtime, and signs it with the certificate.

On starting the gateway device, the Oracle Internet of Things Cloud Service Gateway Software validates the security configuration of the gateway bundles and then activates the gateway device in Oracle IoT Cloud Service.

When you enter the command to start the gateway, the Oracle Internet of Things Cloud Service Gateway Software obtains the certificate information from the trusted assets store, compares it with the certificates of the gateway bundles, and if they match, sends a message to Oracle Internet of Things Cloud Service to start and activate the gateway device.

While running your application on the gateway, the Oracle Internet of Things Cloud Service Gateway Software validates the security configuration of the application runtime and then allows it to run on the gateway device.

When you execute your application on the gateway, the Oracle Internet of Things Cloud Service Gateway Software obtains the certificate information from the trusted assets store and compares it with the certificates of the application runtime. If they match, the application starts on the gateway and securely communicates with Oracle Internet of Things Cloud Service.
Gateway Security Best Practices

A list of security best practices for working with the Oracle Internet of Things Cloud Service Gateway Software is provided and should be followed by Oracle Internet of Things Cloud Service Gateway integrators and people involved with the development and deployment of device software.

The best practices are listed for the areas of IoT gateway integration, device software development, and device software deployment.

Ensure Gateway platform integrity

1. Consider physical protection.
   Consider using tamper-resistant hardware to protect the device from unauthorized physical access.

2. Enforce secure boot.
   Ensure that a Secure Boot process is in place to measure and verify early software stacks executed by your device. Those include software from the device firmware up to the point of launching the main Operating System (OS boot loader). The OS boot loader (e.g. BIOS / UEFI) can then employ alternatives schemes (such as a measured boot) to ensure security and integrity of the Operating System and any application components. If a secure boot is not followed by any boot scheme enforcing security, it is recommended to have the Secure Boot encompassing the kernel of the Operating system itself as part of the Secure Boot.

3. Consider Measured Boot and Device Attestation.
   Enforce Measured Boot and Device Attestation mechanisms whenever possible. These are usually related to the concept of Trusted Platform Module (TPM), but may take the appearance of any hardware module with a dedicated software. Measured Boot is available when a sufficient portion of the system has been booted with Secure Boot. The Measured Boot then takes measurement of any code and data module involved during the boot process. This might include modules up to the application level. Contrary to the Secure Boot, the code is executed and the data is used regardless of their measured values. However, their values have been stored within a hardware protected area, such as a TPM, so that later on, it is possible to generate an Attestation of the device by signing those measurements. This Attestation can be used by a third party to verify the software that is running within the Device.

4. Consider runtime integrity protection.
   Enforce Runtime Integrity protection during the Gateway runtime lifecycle. Both Secure Boot and Measured Boot happen at boot time only, it is recommended to enforce also integrity checks at runtime regarding the Operating System sensitive modules such as the one related to Access Control or the Gateway Stack (code + data). For Linux Operating System in particular, Linux Security Module (LSM) and Simplified Mandatory Access Control Kernel (SMACK) are examples of useful modules for that purpose.

Gateway Device Operating System Hardening

Understand and enforce the security of the operating system that is installed on the device. Using the Linux operating system as an example, consider the following as part of the hardening process:
1. Enforce system level protection.
   a. Enforce all relevant system and security patch updates during the entire life of the gateway device.
   b. Consider the use of LSM (Linux Security Modules) extensions, such as SELinux (Secure-Enhanced Linux).
   c. Ensure that the minimal and only required services are running. Disable all unnecessary services to reduce security vulnerabilities. These services include ftp, telnet, rlogin, serverX, and many more.
   d. Ensure that the minimal and only required devices and interfaces are enabled. Consider using USB, console devices, or network interfaces.
   e. Enforce partitioning.
      i. Consider storing sensitive source code and data in different partitions. In particular, consider having the Operating System in a partition different from partitions used by source code and data related to users and applications.
      ii. Define a partition permission and quota policy.
      iii. Consider encryption of partitions.
2. Enforce audit capabilities.
   Consider enabling the log system(s) to keep an accurate view of the system.
3. Define and enforce a user policy.
   a. Enforce a strong password policy.
   b. Limit users and groups, as much as possible.
   c. Define and enforce a user permission policy.
   d. Consider usage of the sudo command instead of log in as root.
4. Define and enforce a network policy.
   a. Limit the number of open ports on the system to the necessary ones only. Use a port scanner and other network tools to verify the configuration of the gateway software.
   b. Consider using firewall protection.
   c. Restrict remote accesses.
   d. Consider in particular the SSH usage and manage SSH settings.

Gateway Protection
1. Initially, host the Oracle IoT Cloud Service Gateway source code and data in a read-only partition.
   Once configured, only the $GATEWAY_HOME/data folder needs to be in a read-write partition.
2. Ensure that the latest patches are applied to the device.
   If a security vulnerability is identified in the Gateway or the underlying Java runtime, Oracle will release an update. Ensure that a proper mechanism exists for updating the devices which are running the Gateway.
3. Do not execute the Gateway with root account privileges.
To ensure a secure environment, it is recommended that you execute the Gateway with the minimal required user privilege. Running with `sudo` or `root` privilege defeats this purpose. Consequently, in a production environment, it is recommended that you have a user with the minimal required user privilege executing the Gateway.

4. Protect sensitive configurations.

   Enforce an integrity and confidentiality protection for all configuration files. At a minimum, the following files must be considered:
   
   a. `security.properties`
   b. `gateway.permissions`
   c. `gateway.policy`
   d. `gateway.properties`

   If possible, consider setting them to Read Only mode after provisioning and have them as part of an encrypted partition.

5. Separate credential stores.

   Ensure that the truststores and keystores are different stores. That is, the IoT Cloud Service Gateway should count different stores for:
   
   - Server authentication (`ssl.truststore`)
   - Code verification (`code.truststore`)
   - Gateway keystore (`ssl.keystore`)

6. Enforce password policy on truststores and keystores.

   Ensure that default passwords are replaced. The applicable password policy must be enforced and be applied to both the truststores and keystores. Note that for truststores, having the password protection guarantees integrity.

7. Populate truststores with trusted certificates.

   The Gateway uses truststores related to:
   
   - Server authentication (`ssl.truststore`)
   - Code verification (`code.truststore`)

   Each of these truststores must only contain the required certificates. In particular, there must not be any development or test certificates on a production device.

8. Ensure Trusted Certificates Validity

   For Code verification in particular, ensure certificate validity and manage revocation. This means the related truststore has to be up-to-date and it must not contain revoked certificates.

Device Software Development

1. Define and enforce coding rules.

   Device software must follow specific security coding rules. They may be inspired from Secure Coding Guidelines for Java SE whenever relevant.

2. Audit the permissions that would be required to execute device software.

   A device software should request only the minimal set of permissions for its execution. As part of the application development process, consider auditing the
set of permissions that might be required to execute the code. Consider working with a local permission file to determine the minimal set required by the application.

3. Encrypt sensitive data.

If the application deals with sensitive data, consider using encryption to save this data.

It is your responsibility to understand the rules and regulations applicable to the targeted deployment region. Local laws may regulate what type of data may be considered sensitive in the area. Consider the following options to protect sensitive data:

- Use the `KeyValueStorage` interface to manage sensitive data persistently. Data are bound to a given device software.
- Alternatively, rely on the Gateway database that is configured to use encryption to persist data. (The Gateway provides Java DB as a mechanism to persist application data on a database. It also exposes a JDBC service to abstract the underlying database engine. Applications are expected to use `org.osgi.services.jdbc.DataSourceFactory` API to obtain the driver/datasource and subsequently use JDBC APIs to interact with it). Data are not bound to a given device software.

4. Ensure a Gateway in developer mode is only registered/activated with a developer instance of Oracle IoT Cloud Service.

**Device Software Deployment**

Use the following best practices when developing device software for the Oracle Internet of Things Cloud Service Gateway platform.

1. Review and consider updating the Gateway permissions policy.

   The default configuration is as follows:

   a. All permissions are granted to the Gateway in the `$GATEWAY_HOME/config/gateway.policy` file.
   
   b. Global permissions set in the `$GATEWAY_HOME/config/gateway.permissions` file define the rules that apply to the Gateway bundles (including device software).
   
   c. Local permissions permit a bundle (including device software) to request permissions.

   The permission evaluation process is as follows:

   a. The standard Java security checks are applied first (i.e. those from the JVM-wide static security policy).
   
   b. At runtime, when a device software bundle requests a protected operation, the Gateway checks the bundle/local permissions settings. If the settings do not include the requested operation, then the operation fails.
   
   c. The Gateway checks the "global" permissions configured via the `ConditionalPermissionAdmin` service (in the order the rules are defined) until one allows or denies the operation. If no rule matches, then the operation is denied. Note that the "deny unless allowed" approach is still followed in the end. In particular, consider reviewing and updating global permissions since by default, system policy is configured to allow bundles (including device software) to be granted `AllPermission` except the `exitVM` one.
In particular, consider reviewing and updating the global permission settings since, by default, the system policy is configured to allow bundles (including device software) to be granted `AllPermission` except the `exitVM` one.

2. Define and enforce a Device Software permission policy.

   By default, system policy is configured to allow software bundles to be granted `AllPermission` except the `exitVM` one.

   Enforce device software audit and permissions checks:
   
   a. Mandate a local file permission.
   b. Audit the permissions required by a device software and apply the defined policy.
   c. Enforce Code Signing Policy.

   Ensure that the signing key of the device software has not been revealed and the related self-signed certificate has not been revoked.

Typical Workflow for Configuring Your Device with the Oracle Internet of Things Cloud Service Gateway Software

You need to configure your gateway device with the Oracle Internet of Things Cloud Service Gateway Software before it can begin communicating with Oracle Internet of Things Cloud Service.

Use the tasks and their descriptions below to configure your gateway device with the Oracle Internet of Things Cloud Service Gateway Software. After you install the software on the gateway device, you can deploy and run the sample applications that are included in the Oracle Internet of Things Cloud Service Gateway Software bundles.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identify the hardware,</td>
<td>• Hardware requirements: PC, Raspberry Pi, or iMX6 Sabrelite</td>
<td>If required, configure Gradle to run behind a proxy</td>
</tr>
<tr>
<td>software, and information</td>
<td>• Software requirements: JDK/eJDK, the Oracle IoT Gateway Software, and</td>
<td></td>
</tr>
<tr>
<td>requirements</td>
<td>Gradle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Information requirements: Oracle Internet of Things Cloud Service URL and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>port number, your network proxy details</td>
<td></td>
</tr>
<tr>
<td>Install prerequisite software</td>
<td>• Install Java for Windows on PC or eJDK for Linux/ARM on iMX6 or Raspberry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Install and configure Gradle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Configure the proxy information for Gradle if required</td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Description</td>
<td>More Information</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Installation and Configuration</strong></td>
<td><strong>Register the gateway device on Oracle Internet of Things Cloud Service</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sign in to Oracle Internet of Things Cloud Service</td>
<td>Register a Single Device</td>
</tr>
<tr>
<td></td>
<td>• Register a device that represents the gateway</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Enter a password and download the provisioning file</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Download, install, and configure the Oracle Internet of Things Cloud Service Gateway Software on your gateway device</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Download the Oracle Internet of Things Cloud Service Gateway Software</td>
<td>Oracle Internet of Things Cloud Service Gateway Downloads</td>
</tr>
<tr>
<td></td>
<td>• Create an <code>iot/gateway</code> folder and extract the zip files in the folder</td>
<td>Install and Configure a Gateway for Application Development</td>
</tr>
<tr>
<td></td>
<td>• Set the <code>JAVA_HOME</code> and <code>GATEWAY_HOME</code> environment variables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set the <code>gateway.http.proxy</code> property to <code>true</code></td>
<td></td>
</tr>
<tr>
<td><strong>Gateway Security Configuration</strong></td>
<td><strong>Configuring truststores</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Generate a self-signed certificate</td>
<td>Create a Self-Signed Certificate for Gateway Applications</td>
</tr>
<tr>
<td></td>
<td>• Export the certificate from the keystore</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Import the certificate into the truststore</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Verifying the certificates</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• List the certificates in the truststore</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Verify that the self-signed certificate exists</td>
<td></td>
</tr>
<tr>
<td><strong>Gateway Provisioning</strong></td>
<td><strong>Performing gateway provisioning</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Locate the provisioning file and have its password ready</td>
<td>Provision a Gateway</td>
</tr>
<tr>
<td></td>
<td>• Run the unified provisioner tool to update the trusted assets store</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Update the security properties file</td>
<td></td>
</tr>
<tr>
<td><strong>Starting the Gateway</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Description</td>
<td>More Information</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Validating the gateway installation    | • Verify that you installed Java  
• Check the environment variables  
• Verify that you installed the Oracle Internet of Things Cloud Service Gateway Software  
• Ensure that the trusted assets store exists  
• Verify that the security properties file is updated with values for trusted.assets.password and trusted.assets.path | Commands:  
• `java -version`  
• `grep | env _HOME`  
• `ls $GATEWAY_HOME`  
• `ls ${GATEWAY_HOME}/config/ <trustedAssetsStore file name>`  
• `cat ${GATEWAY_HOME}/config/security.properties`                                      |
| Starting the gateway                   | • Enter the command to start the gateway  
• Verify that the gateway device on Oracle Internet of Things Cloud Service Cloud Service is running and appears activated | Commands:  
|                                         | cd $GATEWAY_HOME  
# Linux:  
bash bin/gateway.sh  
# Windows  
bin\gateway.bat | Step 5 in Activating a Device |
| Application Deployment                 | Creating an application on Oracle IoT Cloud Service  
• Create a device model in Oracle Internet of Things Cloud Service  
• Create an application on Oracle Internet of Things Cloud Service  
• Create an enterprise application integration  
• Download the provisioning file | Creating a new Device Model  
Configure the Gateway SDK |
### Task Description

#### Configuring the gateway SDK

- Open the `sdk.properties` file in `$GATEWAY_HOME/sdk/config/`
- Update the value of `sdk.tastore.path` with the path where you saved the provisioning file
- Update the value of `sdk.tastore.storepassword` with the password that you entered to encrypt the provisioning file
- Run the `source ~/.sdk.properties` command to load the changes

#### Building and deploying the sample application

- Build your application on the gateway
- Deploy your application on the gateway
- Examine the output and verify that messages are exchanged between the gateway and your application on Oracle Internet of Things Cloud Service

#### More Information

For more information, see "Configure the Gateway SDK" and "Create Your First Gateway Device Software".

---

### Install and Configure a Gateway for Application Development

Install the gateway to develop and test gateway applications. This procedure describes Linux commands. For a Microsoft Windows environment, use the equivalent command.

1. Prepare your development computer for the software installation:
   a. Make sure the hardware prerequisites are met before you install the Oracle IoT Gateway Software. You require either a PC, a Raspberry Pi or iMX 6 SabreLite to run the gateway software.
   b. Download and install the latest Java SE Development Kit (JDK) 8.
c. If you are installing Java SE 8 Embedded, use the `jrecreate` tool included with the JDK software to create a JDK Compact 2 profile. For example:

```
cd ejdk1.8.0_73/bin
./jrecreate.sh -d /home/janeuser/jre1.8.0_73_compact2_minimal_vm -p compact2 --vm minimal
```

d. Download and extract Gradle. Versions 2.2.1 or later are supported.

2. If your system is behind a corporate firewall:
   a. Create a `gradle.properties` file.
   b. Open a command prompt and use a text editor to add these entries to the `gradle.properties` file:

```
systemProp.http.proxyHost=<your-company—proxy-server>
systemProp.http.proxyPort=<your-company-port-number>
systemProp.https.proxyHost=<your-company—proxy-server>
systemProp.https.proxyPort=<your-company-port-number>
```
   c. Run the `source ~/.gradle.properties` command to load the changes.

3. Install and configure the gateway software on the gateway device:
   a. Ensure that the time in your system is current. Use `date --s` to set the date if needed.
   b. Create a directory named `iot` and a subdirectory named `gateway`.
   c. Open a web browser and browse to Oracle Internet of Things Cloud Service Gateway Downloads.
   d. Accept the license agreement and then download the `iotcs-gw-bin-<version>.zip` and the `iotcs-gw-samples-<version>.zip` files to the `/iot/gateway` directory. Extract these files in the `iot/gateway` folder. The files are extracted to a directory `iotcs/gw`.
   e. Notice that the `iotcs-gw-samples-<version>` files extract to two types of samples:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iotcs-gw-samples-&lt;version&gt;.zip</code></td>
<td>Extract only on your gateway application development platform. Contains the source files of the sample device adapters and applications that can be run on the Oracle Internet of Things Cloud Service Gateway. These samples are platform independent and can be built using the bundled Oracle Internet of Things Cloud Service Gateway SDK.</td>
</tr>
<tr>
<td><code>iotcs-gw-linux-samples-&lt;version&gt;.zip</code></td>
<td>Contains the optional Oracle Internet of Things Cloud Service Gateway runtime extensions that can run on the Linux platform. It contains native components for both ARMv6-HardFloat and x86 platforms. Other sample bundles can also be overlaid on an existing Gateway installation.</td>
</tr>
</tbody>
</table>
4. Edit the ~/.profile file on the gateway device:
   
a. Open a command prompt and use a text editor to add these entries to the ~/.profile file:
   
   ```
   JAVA_HOME=<path_to_JRE_or_JDK_installation_folder>
   GATEWAY_HOME=<IoTCS-gateway-install-dir>/iot/gateway/iotcs/gw
   ```
   
   If your network requires a proxy for HTTP(S) and TCP traffic, set the GATEWAY_JVM_ARGS value to the default HTTPS and TCP proxy ports to route the gateway software through your network. For example:
   
   ```
   GATEWAY_JVM_ARGS="-Dhttps.proxyHost=myproxy.domainname.com -Dhttps.proxyPort=443"
   ```
   
   See https://docs.oracle.com/javase/8/docs/api/java/net/doc-files/net-properties.html for network configuration information.
   
   b. Run the source ~/ .profile command to load the changes.

Create a Self-Signed Certificate for Gateway Applications

This page describes the steps to create and install user application certificates into the Gateway trusted assets keystore.

To begin with, get familiar with the Gateway Security Concepts. When you create a custom application and run it on the gateway, a user certificate must be used for signing and verifying the application. The created certificate is stored in $GATEWAY_HOME/SDK/config/code.keystore and $GATEWAY_HOME/config/code.truststore. It is used for signing the application bundles and to authenticate your device application with the OSGi container. You can generate a certificate but if your company has its own certificate, you may use it. The gateway bundles, located in the gateway directory, are already signed with the IoT Gateway certificate. The IoT Gateway certificate is stored in $GATEWAY_HOME/config/code.truststore and ships with the Oracle IoT Gateway Software.
Note:

There are two passwords specified: storepass and keypass. The storepass value is used to access code.keystore. The keypass value is used to access this specific key within the storepass. The keytool utility supports locking down individual keys with individual passwords.

1. While logged in your device software development platform, navigate to where you installed the Oracle IoT Cloud Service Gateway.
   
   cd $GATEWAY_HOME

2. Using the keytool command, create a self-signed certificate that will be stored in the $GATEWAY_HOME/sdk/config/code.keystore file.
   
   The option values for the dname option must match the dname values used in the $GATEWAY_HOME/config/gateway.permissions file. They are set to CN=IoT Application Developer, OU=Internet of Things, O=Oracle Corporation.
   
   The following is an example. For more information about the keytool command, see keytool documentation
   
   keytool -genkey -dname "CN=IoT Application Developer, OU=Internet of Things, O=Oracle Corporation" -alias "IoT Application Developer" -keyalg RSA -validity 365 -keystore sdk/config/code.keystore -storepass iotgateway -keypass iotgateway

3. From the $GATEWAY_HOME/config/code.keystore, export the public key certificate that was generated in the previous step.
   
   keytool -exportcert -alias 'IoT Application Developer' -file iotappdev-cert -keystore sdk/config/code.keystore -storepass iotgateway

4. Import the exported certificate to the $GATEWAY_HOME/config/code.truststore file. This code truststore will now have a public key.
   
   keytool -importcert -noprompt -alias 'IoT Application Developer' -file iotappdev-cert -keystore config/code.truststore -storepass iotgateway

Provision a Gateway

Provisioning the gateway consists of obtaining a trusted assets store. The trusted assets store is an encrypted file that contains information such as the server certificate the gateway will use when creating an SSL connection to the Oracle IoT Cloud Service instance, the Oracle IoT Cloud Service host name and port, and other certificates. It is typically downloaded from the Oracle Internet of Things Cloud Service and is known as the provisioning file. You use the Gateway Unified Provisioner tool that appends this provisioning file with additional fields such as code signing certificates which then forms the trusted assets store. This procedure describes Linux commands. For a Microsoft Windows environment, use the equivalent command.
1. Use the Gateway Unified Provisioner tool to append the contents of a trusted assets store downloaded from the Oracle IoT Cloud service UI.
   a. Open a command prompt and ensure that the `JAVA_HOME` and `GATEWAY_HOME` environment variables are set.

```
export JAVA_HOME=<path_to_JRE_or_JDK_installation_folder>
export GATEWAY_HOME=<IoTCS-gateway-install-dir>/iot/gateway/iotcs/gw
```

If your network requires a proxy for HTTP(S) and TCP traffic, set the `GATEWAY_JVM_ARGS` value to the default HTTPS and TCP proxy ports to route the gateway software through your network. For example:

```
GATEWAY_JVM_ARGS="-Dhttps.proxyHost=myproxy.domainname.com -Dhttps.proxyPort=443"
```

See https://docs.oracle.com/javase/8/docs/api/java/net/doc-files/net-properties.html for network configuration information.

b. Execute this interactive command from the `GATEWAY_HOME` directory.

```
./bin/uprovisioner.sh
```

c. Enter the location of the downloaded trusted assets store file.

d. Enter the password used to encrypt the trusted assets store file, as specified in the Oracle Internet of Things Cloud Service UI when generating the assets store.

e. To add a code signing certificate, respond yes and specify the location of the certificate.

f. Specify the location of the resulting truststore, which can be used to add gateway code signing certificates. Note that this field must be set to `config/code.truststore` for the gateway to execute properly.

g. Enter the password of the gateway truststore.

h. If you wish to display the resulting contents of the gateway truststore, response yes.

2. Open and update the `$GATEWAY_HOME/config/security.properties` file with the values of following fields:

```
trusted.assets.password=<password of your provisioning file>
trusted.assets.path=<path and name of your provisioning file relative to $GATEWAY_HOME/config directory>
```

Configure the Gateway SDK

Use the Oracle Internet of Things Cloud Service Gateway Software Development Kit (SDK) Gradle plugin to develop device software for your gateway devices. The device software can either be a device adapter or a device application that provides logic at
the gateway device level to manage connectivity, bandwidth, and the type of data to report.

If you deploy your device software by using Gradle deploy or the Oracle IoT Cloud Service UI, then you must register the gateway device as an Oracle Internet of Things Cloud Service enterprise application.

**Note:**

As a developer, if you deploy your software to the gateway device by simply copying the software bundles to $GATEWAY_HOME/samples/dist, then you do not need to perform this procedure.

1. Register the Oracle Internet of Things Cloud Service Gateway SDK:
   a. Log in to your Oracle Internet of Things Cloud Service instance.
   b. Click the **Menu** icon adjacent to the Oracle Internet of Things Cloud Service title on the Management Console.
   c. Click **Applications** and then **Browse Applications**.
   d. Click **Create Application**.
   e. Complete the **Name** and **Description** text fields, and then click **Create**.
   f. Click the name of the of your application in the left pane.
   g. Click **Integration**.
   h. Click **Create Integration** and select **Enterprise Application**.
   i. Complete the **Name**, **Description**, and **URL** fields. The value for **URL** field can be a value such as https://helloworld.com/.
   j. Click **Create**.

2. Download the SDK provisioning file:
   a. Select the integration and click the **Edit** icon.
   b. Enter a password in the **File Protection Password** field to encrypt the provisioning file that contains the configuration and credentials to activate your integration.
   c. Enter the password again in the **Confirm Password** field.
   d. Click **Download Provisioning File**.
   e. Click **Save File**.
   f. Browse to the /iot/gateway/iotcs-gw-sdk-<version>/iotcs/gw/sdk/config directory and click **Save**.

3. Update the SDK provisioning file properties:
   a. Open a command line and run the `cd $GATEWAY_HOME/sdk/config` command.
   b. Run the `vi ~/.sdk.properties` command.
   c. Update the `sdk.tastore.path` value to the path where you saved the provisioning file.
d. Update the `sdk.tastore.storepass` value to the password used to encrypt the provisioning file.

e. Run the `source ~/.sdk.properties` command to load the changes.

Create Your First Gateway Device Software

A Gradle template file is included in the Oracle IoT Cloud Service Gateway SDK bundle. It enables you to create the template Java source files you can use to create your first gateway device software.

1. Set up your Oracle IoT Cloud Service Gateway application development environment. See Install and Configure a Gateway for Application Development.

2. Create your first MyDevice project.

   a. Navigate to the `$GATEWAY_HOME/samples/tutorial` folder.

   b. Run the following `gradle` command. This command will create the "MyDevice" project directory and the `$GATEWAY_HOME/samples/tutorial/MyDevice/build.gradle` file and the source file templates for custom device type tuple. The generated interfaces are: `MyDevice`, `MyDeviceEndpoint` and `MyDeviceEvent`.

```
gradle -b $GATEWAY_HOME/sdk/build.gradle.template createDeviceType
```

An output, similar to the following, is generated:

```
$ gradle -b $GATEWAY_HOME/sdk/build.gradle.template createDeviceType
:createDeviceType
Project directory MyDevice created.
Overwriting project build script D:\ws\iot\gateway\iotcs\gw\samples\tutorial\MyDevice\build.gradle.
File: MyDevice\build.gradle written.
File: MyDevice\src\main\java\com\mycompany\iot\device\MyDevice.java written.
File: MyDevice\src\main\java\com\mycompany\iot\device\MyDeviceEndpoint.java written.
File: MyDevice\src\main\java\com\mycompany\iot\device\MyDeviceEvent.java written.
BUILD SUCCESSFUL
Total time: 6.282 secs
```

A new `MyDevice` project directory is created in the `$GATEWAY_HOME/samples/tutorial/MyDevice` folder. The new `MyDevice` folder contains the `build.gradle` file and the source file templates for the custom device type. The generated interfaces are `MyDevice, MyDeviceEndpoint, and MyDeviceEvent`.

3. Build the `MyDevice` project.

   a. Make sure that you’ve provisioned the trusted assets of your SDK, as described above.

   b. Navigate to your `MyDevice` folder that you just created.
c. Edit the build.gradle file and uncomment the OSGi instruction statement (line 36) so that it appears as shown below.

```gradle
instruction 'Export-Package', 'com.mycompany.iot.device
```

d. Build the project by running the `gradle assemble` command, as illustrated below.

```
$ gradle assemble
:processResources UP-TO-DATE
:classes
:jar
:assemble
BUILD SUCCESSFUL
Total time: 10.474 secs
```

e. Navigate back to the parent folder `$GATEWAY_HOME/samples/tutorial`.

4. Create your application.

a. In the `$GATEWAY_HOME/samples/tutorial` folder, run the following `gradle` command.

```
$ gradle -b $GATEWAY_HOME/sdk/build.gradle.template
createApplication
```

This command creates the `MyDeviceApplication` project folder, the `MyDeviceApplication/build.gradle` file, and the source file template, `MyDeviceApplication.java`, for a gateway device software.

The following output is generated:

```
$ gradle -b $GATEWAY_HOME/sdk/build.gradle.template
createApplication
:createApplication
Project directory MyDeviceApplication created.
Overwriting project build script D:\ws\iot-re10.git\build\gateway\iotcs-gw-1\samples\tutorial\MyDeviceApplication\build.gradle.
File: MyDeviceApplication\build.gradle written.
File: MyDeviceApplication\src\main\java\com\mycompany\iot\MyDeviceApplication.java written.
BUILD SUCCESSFUL
Total time: 6.664 secs
```

5. Build the application.

a. Navigate to the `MyDeviceApplication` folder.

b. Edit the `build.gradle` file by changing the compile configuration on line 43 in the dependencies block so that it looks like the following:

```gradle
compile files('..:/MyDevice/build/libs/MyDevice.jar')
```
c. Build the project by running the `gradle assemble` command, which will generate the following output:

```
$ gradle assemble
:compileJava
:processResources UP-TO-DATE
:classes
:jar
:assemble
BUILD SUCCESSFUL
Total time: 10.908 secs
```

6. Configure the Gateway security properties and permissions.

a. Modify the `$GATEWAY_HOME/config/security.properties` file and set the `gateway.sdk` property to `true`.

   By default, this property is set to `false`. Setting it to `true` enables direct deployment via the SDK.

b. Copy and paste the following block of code to the end of the `$GATEWAY_HOME/config/gateway.permissions` file:

   ```
   ##################### All directly deployed bundles
   ########################
   DENY {
   [org.osgi.service.condpermadmin.BundleLocationCondition "iot://devapps/**"]
   ( java.lang.RuntimePermission "exitVM" )
   ( java.lang.RuntimePermission "setSecurityManager" )
   ( org.osgi.framework.AdminPermission "*" "execute" )
   ( org.osgi.framework.AdminPermission "*" "lifecycle" )
   } "Permissions disallowed for directly deployed bundles"
   
   ALLOW {
   [org.osgi.service.condpermadmin.BundleLocationCondition "iot://devapps/**"]
   ( java.security.AllPermission "*" "*" )
   } "Permissions allowed for directly deployed bundles"
   ```

   **Note:**

   You must restart the Gateway device after making changes to the Gateway security properties and permissions for the changes to take effect. This can be done using the **Restart Main Process** option under the **Diagnostics and Remote Configuration** section on the Device Details page.

7. Deploy and run the application.

   The `deploy` task will deploy the project's bundle to the configured gateway device. It will install the bundle on the gateway and start it. The `MyDeviceApplication` bundle depends on the `MyDevice` bundle, so you need to deploy the `MyDevice` bundle first.
a. Navigate to the $GATEWAY_HOME/samples/tutorial/MyDevice project folder.

b. Deploy the MyDevice bundle using the gradle deploy command, which generates the following output:

```
gradle deploy
:deploy
Bundle 'com.mycompany.iot.device.mydevice_1.0.0' was successfully deployed.
BUILD SUCCESSFUL
Total time: 10.471 secs
```

c. Navigate to the $GATEWAY_HOME/samples/tutorial/MyDeviceApplication project folder.

d. Deploy the MyDeviceApplication bundle using the gradle deploy command, which generates the following output:

```
$ gradle deploy
:deploy
Bundle 'com.mycompany.iot.mydeviceapplication_1.0.0' was successfully deployed.
BUILD SUCCESSFUL
Total time: 9.093 secs
```

8. Modify and redeploy the device software.

a. Make some changes to the source files in the MyDevice or MyDeviceApplication project folders.

b. Run the gradle assemble in the corresponding project folder to rebuild the bundle.

c. Run the gradle deploy to redeploy the project bundle to the gateway device.

   If there is already a bundle running, that bundle will be stopped, updated, and restarted.

9. Uninstall the gateway device application using the gradle undeploy command.

Perform either of the following steps:

- To uninstall the MyDeviceApplication bundle, navigate to the $GATEWAY_HOME/samples/tutorial/MyDeviceApplication project folder and run gradle undeploy. The following output is generated:

  ```
  $ gradle undeploy
  :undeploy
  Bundle 'com.mycompany.iot.mydeviceapplication_1.0.0' was successfully undeployed.
  BUILD SUCCESSFUL
  Total time: 8.827 secs
  ```
To uninstall the MyDevice bundle, navigate to the $GATEWAY_HOME/samples/tutorial/MyDevice project folder and run gradle undeploy. The following output is generated:

```
$ gradle undeploy
:undeploy
Bundle 'com.mycompany.iot.mydevice_1.0.0' was successfully undeployed.
BUILD SUCCESSFUL
Total time: 8.723 secs
```

10. Release your gateway device software to production.

Once you are satisfied with the device software and are ready to release it to production, publish the device software using the information in Upload Device Software to the Repository and Install Device Software on a Registered Device.

## Deploy New Device Software to Your Gateway Device

After successfully building your new device software, you can deploy it to your gateway device.

There are three ways to deploy your new software to the gateway device. Before you deploy your software to the gateway using any of the methods, ensure that you have added the information about your new software bundle to the $GATEWAY_HOME/config/gateway.permissions file and the $GATEWAY_HOME/config/gateway.properties file. If your device software has dependencies with other bundles, then the information about those bundles should also be present in these files.

- As a developer you can do a direct deployment by copying your software files to the $GATEWAY_HOME folder by using the procedure listed in this page.
- In a development environment, you can directly deploy to a gateway device by using the Gradle plugin. See steps 6 and 7 of Create Your First Gateway Device Software.
- In a production environment, you use the Oracle Internet of Things Cloud Service management console for deployment. See Upload Device Software to the Repository and Install Device Software on a Registered Device.

1. Run the `cd $GATEWAY_HOME` command to browse to the gateway home directory.

2. Run this command to create a new folder in `<IoTCS-Gateway-install_dir>/iot/gateway/iotcs/gw/`:

   ```bash
   mkdir $GATEWAY_HOME/iotapp_bundles
   ```

3. Copy the software files from the device folder `<projectdir>/build/libs/* .jar` to the `$GATEWAY_HOME/iotapp_bundles` folder. For example, run this command to publish the device software files to an installation of the gateway software on the same platform as your development environment:

   ```bash
   cp <projectdir>/build/libs/* .jar $GATEWAY_HOME/iotapp_bundles
   ```
4. Add the information about the new gateway device software bundles to the $GATEWAY_HOME/config/gateway.properties file. For example, run this command to add the IoTGatewayApp.jar software bundle:

```bash
gateway.auto.start.3= \n ${gateway.bundles.path}/../iotapp_bundles/IoTGatwayApp.jar
```

**Note:**

If there are other device software bundles already listed in this property, add the information about the new device software bundles to the end of the list. Ensure to add the device software bundles in the order they must be loaded if there are dependencies among the software bundles.

5. Add the information about the new device software to the $GATEWAY_HOME/config/gateway.permissions file. For example:

```ini
# ALLOW {
[org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../iotapp_bundles/*"]
[org.osgi.service.condpermadmin.BundleSignerCondition "CN=Device Software Developer, OU=Internet of Things, O=Example Corporation"]
(java.security.AllPermission "" "")
}"Permissions for iotapp_bundles"
```

6. Run this command on the gateway device to remove the $GATEWAY_HOME/data/felix-cache folder:

```bash
cd $GATEWAY_HOME
rm -rf data/felix-cache
```

7. Run these commands to run the gateway:

```bash
cd $GATEWAY_HOME
bin/gateway.sh
```

### Upload Device Software to the Repository

You can upload a device software bundle to the Oracle Internet of Things Cloud Service Repository using the **Software** tab of the Oracle Internet of Things Cloud Service Management Console. To ensure that the **Software** tab is visible, in your Oracle Internet of Things Cloud Service instance, on the **Settings** page, under the **Software Management** section, select **Enable Software Deployment**.

Once you upload a device software, whether it be a device adapter or a device application, you can deploy that software bundle to the registered devices that you have authorization to manage. Before you deploy your software to the device, ensure that you have added the information about the software bundle to the gateway device in the $GATEWAY_HOME/config/gateway.properties file and
the $GATEWAY_HOME/config/gateway.permissions file. If your software bundle has dependencies with other software bundles, then the information about those bundles should also be added to these files.

Login to your Oracle Internet of Things Cloud Service instance and complete these steps to upload your device software.

1. From the Management Console, click **Devices**.
2. Click **Software** on the left side.
3. Upload the device software.
   - If you currently do not have any device software uploaded, you will see a page similar to the following image. Click **Upload a Device Software**.

![Image of Oracle Internet of Things Cloud Service interface](image)

4. In the **Upload Software** dialog, click **Browse**.
   The Open dialog displays your system’s File navigation window.
5. Navigate and select the OSGi bundle you want to upload. Click **Open**.

   **Note:**
   The OSGi bundle must have a start level of 2 or greater in the gateway.

6. Back in the **Upload Software** dialog, click **Upload**.
   The software is uploaded to the Oracle Internet of Things Cloud Service artifact repository and its information is displayed in a table.

After you've successfully uploaded your device software, you can install it on a single device or multiple devices. Use the information at **Install Device Software on a Registered Device**.
Install Device Software on a Registered Device

After successfully uploading your device software to the Oracle IoT Cloud Service repository, you can install it on your gateway device. The gateway device must be already activated and online at the time of device software installation.

This deployment method is usually done in a production environment. Before you deploy your software to the device, ensure that you have added the information about the software bundle to the gateway device in the $GATEWAY_HOME/config/gateway.properties file and the $GATEWAY_HOME/config/gateway.permissions file. If your software bundle has dependencies with other software bundles, then the information about those bundles should also be added to these files. To install the device software on your gateway device:

1. Sign in to your Oracle IoT Cloud Service instance and complete these steps to install device software.

   1. Click Devices in the Management Console.
   2. Click Software on the left navigation bar.
      
      A table of device software that has been uploaded are shown.
   3. Select the row for the device software that you want to install and click Install.
      
      The Install Software dialog displays a list of devices on which you can install the device software. Only the devices that have an Activated state and an Online connectivity status are displayed.
   4. On the Install Software dialog, select the device to install the device software on by clicking on its row.
      
      Use the Property and Value search fields to narrow down the list of devices to choose from.
   5. Click Install on the bottom right corner of the screen.

You can now start the device software using the information at Manage Your Installed Device Software.

Manage Your Installed Device Software

Once you've installed a device software on your gateway device, you can start, stop, or delete it using the Oracle IoT Cloud Service Management Console.

To manage your installed device software:

1. From the Oracle IoT Cloud Service Management Console, click Devices.
2. Click Management in the left navigation area.
   
   The Overview - All Devices page is displayed and lists
3. Search for the gateway device you want to work with.
   
   Only gateway devices can have device software installed on them, so use the Property and Value search fields to narrow your selection list, if necessary.
a. In the Property field, select Type from the drop-down list.

b. Click in the Value text field and select Gateway.

The list of gateway devices is displayed.

4. Select the row for the gateway device you want to work with and click pencil (Edit) icon to view and edit the device’s information.

The detailed page for the Gateway Device is displayed.

5. At the bottom of the page, expand the Software tab.

If there’s any, the list of device software applications that have been installed on the gateway device is displayed.

6. Select the row for the gateway device software you want to work with and in the Action column, select the action you want to take.

Note:

Ensure that you are aware of all the dependencies that your device software has before starting or stopping it. For example, device types must be installed first before installing and starting device adapters or device software.

- **Start** - runs the installed device software on the gateway device.
- **Stop** - stops the running device software.
- **Delete** - removes the installed device software from the gateway device.

Note:

If the gateway device is currently offline, the Action column will display N/A (the gateway is offline), as shown in the image below.
Depending on the action you chose to take on the device software, the Status column is continuously updated until the chosen action is completed. The following Status values are possible:

- **Installing** - the device software is currently in the process of being installed on the gateway device.
- **Installed** - the device software is installed and ready to be started on the gateway device.
- **Ready** - the device software is ready to be started or uninstalled.
- **Starting** - in the process of starting the device software on the gateway device.
- **Active** - the device software has been successfully started and is active on the gateway device.
- **Stopping** - in the process of stopping the device software on the gateway device. Once the device software is stopped, the status changes to **Ready**.
- **Uninstalling** - in the process of removing the device software from the gateway device.
- **Uninstalled** - device software has been removed from the gateway device.

### Oracle IoT Gateway Software Terminology

Learn about the terminology used when installing and configuring the Oracle Internet of Things Cloud Service Gateway Software.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GATEWAY_HOME</td>
<td>An environment variable that you create to save the installation location of the Oracle Internet of Things Cloud Service Gateway Software</td>
</tr>
<tr>
<td>%GATEWAY_HOME%</td>
<td></td>
</tr>
<tr>
<td>gateway bundles</td>
<td>The Oracle Internet of Things Cloud Service Gateway Software files in the $GATEWAY_HOME directory</td>
</tr>
<tr>
<td>application bundles</td>
<td>The sample applications located in the $GATEWAY_HOME/samples/dist directory</td>
</tr>
<tr>
<td>SDK</td>
<td>The Oracle Internet of Things Cloud Service Gateway Software development kit that you can use to develop gateway applications</td>
</tr>
<tr>
<td>keytool</td>
<td>A Java utility that manages keys and certificates, and lets you to save certificates as files in keystores</td>
</tr>
<tr>
<td>provisioning</td>
<td>The process of updating the provisioning file that you downloaded from Oracle Internet of Things Cloud Service with the certificates needed for the gateway software to run securely</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>provisioning file and trusted assets store</td>
<td>A downloadable file that Oracle Internet of Things Cloud Service creates when you register a device. It is password-protected and stores information including the Oracle Internet of Things Cloud Service URL, the registration ID, the registration shared secret, and the server's SSL certificate. On the gateway, this provisioning file is updated by the unified provisioner tool with security certificates to allow signed code bundles to run on the gateway. The updated provisioning file is also known as the trusted assets store.</td>
</tr>
<tr>
<td>gateway unified provisioner</td>
<td>A stand-alone Java application that modifies the provisioning file to help secure gateway functionality. You run it through the gateway provisioner script ($GATEWAY_HOME/bin/uprovisioner.sh)</td>
</tr>
<tr>
<td>IoT Gateway certificate</td>
<td>A certificate that is stored in the $GATEWAY_HOME/config/code.truststore file and ships with the Oracle Internet of Things Cloud Service Gateway Software</td>
</tr>
<tr>
<td>IoT Application Developer certificate</td>
<td>A generated self-signed certificate that is stored in the $GATEWAY_HOME/sdk/config/code.truststore file and ships with the Oracle Internet of Things Cloud Service Gateway Software</td>
</tr>
<tr>
<td>code-signing certificate truststore</td>
<td>A Java keystore file that contains code-signing certificates. The gateway uses these certificates during runtime to validate code bundles such as applications and internal libraries. Typically, the path of the file is $GATEWAY_HOME/config/code.truststore, but you can choose another name when you configure the gateway.</td>
</tr>
<tr>
<td>SDK code-signing certificate truststore</td>
<td>A Java keystore file whose path is $GATEWAY_HOME/sdk/config/code.truststore. It contains the certificates that are used to sign applications that are built by the SDK and run on the gateway.</td>
</tr>
</tbody>
</table>
Use the Gateway SDK to Integrate the Cloud Service with Java IDEs

The Oracle IoT Cloud Service Gateway SDK can be used to integrate Oracle Internet of Things Cloud Service with Java IDEs, such as Eclipse, IntelliJ IDEA, and NetBeans IDE. You can create and build a device adapter or a device application using a Java IDE, and then deploy the device software to your gateway devices that have been registered with Oracle IoT Cloud Service.

Topics:
- Preparing the Gradle Build Script
- Use IntelliJ IDEA with the Gateway SDK
- Use Eclipse IDE with the Gateway SDK
- Use NetBeans IDE with the Gateway SDK
- Use a Java IDE to Build the Device Software

Prepare the Gradle Build Script

You need to create the build.gradle configuration script file that defines the project and the tasks that you can use to start developing a device adapter or a device application for your Oracle IoT Cloud Service gateway device.

1. Set up your development environment.
   See Install and Configure a Gateway for Application Development.

2. Create a build.gradle file in the same folder where you will be building your device software.

3. Copy the following lines of code and paste them in the build.gradle file.

```groovy
apply plugin: 'oracle-iot'

repositories {  
    // For other dependencies  
    mavenCentral()  
}

buildscript {  
    repositories {  
        // For the IOT Gradle plugin  
        flatDir { dirs "$System.env.GATEWAY_HOME" + "/sdk/lib/" }  
        // For other dependencies  
        mavenCentral()  
    }  
    dependencies {  
        classpath files("$System.env.GATEWAY_HOME" + "/sdk/lib/")
    }
}
```
GradlePlugin.jar")
classpath group: 'oracle.iot', name: 'GradlePlugin', version: '1.0'
   // For registering a device
classpath 'org.apache.httpcomponents:httpclient:4.4'
classpath 'org.apache.httpcomponents:httpcore:4.4'
}
}
// This task is defined by the Oracle IoT Cloud Service SDK Gradle
// plugin for generating an IoT
// application source file template.
createApplication {
   templateOverride = true // or false
classPackage = 'com.mycompany.iot'
className = 'IotApp'
}
// This task is defined by the Oracle IoT Cloud Service SDK Gradle
// plugin to register a
// gateway with the Oracle IoT Cloud Service server
// Specify your device's registration information
registerEndpoint {
   epType = 'DEVICE'
   manufacturer = 'MyCompany'
   sharedSecret = 'Jo34909ssdgs'
   serialNumber = '22234-d34BC'
   modelNumber = '2REI'
   description = 'This is The Device'
}
// If you are registering more than on device from the same build file
// use this form of task configuration
/*
task registerDevice(type: oracle.iot.tasks.RegisterEndpointTask) {
   epType = 'DEVICE'
   manufacturer = 'MyCompany'
   sharedSecret = 'Jo34909ssdgs'
   serialNumber = '224-d334BC'
   modelNumber = '2REI'
   description = 'This is The Device'
}
*/

Use IntelliJ IDEA with the Gateway SDK

You can use IntelliJ IDEA with the Oracle IoT Cloud Service Gateway SDK to create
and build device adapters or device software applications to use with gateway devices
that have Oracle IoT Cloud Service Gateway installed.

1. Download IntelliJ IDEA for your platform from https://www.jetbrains.com/idea/download/.
2. Launch IntelliJ IDEA.
3. Click Import Project, navigate to the location of the build.gradle file you just created, and click OK.
4. In the Import Project from Gradle dialog, click OK to use the local Gradle distribution.

5. From the Main menu, select View, Tools Windows, and then Gradle.
6. From the **Gradle projects** pane, expand your project’s folder, **Tasks**, and then **oracleiot** folder to display the available templates to choose from.

   - Double-click **createApplication** to create a device software application that will run on the gateway device.

     A source file template for an Oracle IoT Cloud Service device software application is generated and the following output is displayed.

     ```
     11:40:24 AM: Executing external task 'createApplication'.
     BUILD SUCCESSFUL
     Total time: 0.965 secs
     11:40:24 AM: External task execution finished 'createApplication'.
     ```

   - To create a device adapter software to run on your gateway device, double-click **createDeviceAdapter**, as shown below.
A source file template for an Oracle IoT Cloud Service device adapter is generated and the following output is displayed.

7. Open the source application file that was generated to learn more about the structure used to develop a device adapter or a device software.
   a. In the Project window, expand the project folder, then the src, main, java, and org.iot.sample folders
   b. Double-click SampleIoTDeviceAdapter, as shown below, to learn about creating a new device adapter.
   c. Make changes to the file to develop your own device adapter.
8. Build your project using Use a Java IDE to Build the Device Software.

Use Eclipse IDE with the Gateway SDK

The Oracle IoT Cloud Service Gateway SDK is available for use with IDEs, such as Eclipse, to help you create and build device software to use with your gateway devices that have Oracle IoT Cloud Service Gateway installed.

1. Download the latest Eclipse IDE for Java Developers for your platform from https://eclipse.org/downloads/.
2. Install Eclipse IDE on your device software development platform.
3. Start the Eclipse IDE.

4. Install the Gradle Integration plugin.
   a. From the Eclipse IDE's main menu, select Help and then Eclipse Marketplace
   b. In the Search tab of the Eclipse Marketplace window, type Gradle Integration in the Find field and click Go.
   c. Click Install for the Gradle Integration.
   d. In the Confirm Selected Features pane, click Confirm to proceed with the installation.
   e. Accept the terms of the license and click Finish.
   f. Restart Eclipse when prompted.

5. From the Eclipse IDE main menu, select File and then Import.

6. In the Import dialog, expand the Gradle node and select Gradle Project, as shown below.

7. Click Next.

8. In the Import Gradle Project window, click Browse.

9. Navigate to the project folder that contains the build.gradle file and click OK.

10. Back in the Import Gradle Project, with the Root folder populated, click Build Model.

11. Select the discovered project and click Finish.
12. From the main menu, select **Window, Show View**, and then **Other**.
   The Show View dialog appears.

13. From the Show View dialog, expand the **Gradle** node and select **Gradle Tasks**. 
   Click **OK**, as shown below.

   ![Show View dialog](image)

   The **Gradle Tasks** tab is added to the IDE’s window.

14. Set the **GATEWAY_HOME** environment variable after importing the project, so that the **oracle-iot** plug-in can be found.

   a. Right-click the **Samples** folder under Gradle Tasks, and select **Open Gradle Run Configuration** from the context menu that appears.

   ![Open Gradle Run Configuration](image)

   b. Click **Variables** under the Gradle Tasks tab.
c. Click **Edit Variables**.
   The Preferences dialog appears.

d. Click **New** to add a new variable.

e. Set a value for **GATEWAY_HOME**. This should be the directory into which you have installed the SDK.
Use NetBeans IDE with the Gateway SDK

The Oracle IoT Cloud Service Gateway SDK is available for use with IDEs, such as NetBeans IDE, to help you create and build device software to use with your gateway devices that have Oracle IoT Cloud Service Gateway software installed.

To use NetBeans IDE with Oracle IoT Cloud Service Gateway SDK:
1. Download the latest NetBeans IDE version at https://netbeans.org/.
2. Install NetBeans IDE on your device software development platform.
3. Start NetBeans IDE.
4. Install the Gradle plugin.
   a. From the NetBeans main menu, select Tools and then Plugins.
   b. From the Plugins dialog, select Available Plugins tab.
   c. In the Search field, type gradle.
      The list of available plugins that contain Gradle is listed.
   d. Select Gradle Support and click Install.
   e. In the NetBeans IDE Installer dialog, click Next.
   f. Accept the terms of the license agreement and click Install.
      The installer proceeds with downloading the requested plugin.
   g. Click Finish when the installation has successfully completed.
   h. Restart the IDE, if prompted.
5. Import your gateway application project’s build.gradle file.
   a. From the main menu, select File and then Open Project.
   b. Select the folder that contains the build.gradle file that you created earlier using Preparing the Gradle Build Script.
   c. Click Open Project.
      The project folder is loaded in the Projects window.
6. From the Projects window, right-click the project node and choose Tasks.

Use a Java IDE to Build the Device Software

Once you’re ready to build your device software, create a self-signed certificate that you will need to use to build and run the device software using a Java IDE. Use the build task provided with the Gradle plugin template to build the software.

The IntelliJ IDEA is used as an example of a Java IDE. This section assumes that you have set up your IntelliJ IDEA development environment and have created an IntelliJ IDEA project using the Gradle plugin provided with the Oracle IoT Cloud Service Gateway software bundle. If necessary, see Prepare the Gradle Build Script for more information.

1. In your device software’s project folder, create a config folder that you will use to hold the self-signed certificate you will be generating.
2. Create a self-signed certificate for your new device software project using Create a Self-Signed Certificate for Gateway Applications.

3. In the IntelliJ IDEA task window, double-click build to build the project, as shown below.

```
Graddle projects

IoT-test-app
- Tasks
  - build
    - documentation
    - oracle iot
    - other
    - verification
    - Dependencies
    - Run Configurations
```

4. Make modifications to your software as needed and repeat the build step.

5. When the device software is ready for deployment, use Deploy New Device Software to Your Gateway Device to deploy your device software.
Work with Sample Adapters and Applications

Use the sample device adapters and gateway applications included with Oracle Internet of Things Cloud Service to help you develop applications for your environment.

Topics

• Upload the Sample Device Models
• Use Media with the Client Libraries
• Deploy the Sample Gateway Application
• Install the Bluetooth Adapter and Sample Applications in the Gateway
• Work with the Sample UART Device Adapter
• Work with the Sample RS-232 Device Adapter
• Work with the Sample RS-485 Device Adapter
• Work with the Gateway Service Sample

Upload the Sample Device Models

To complete the Oracle Internet of Things Cloud Service samples, the humidity and temperature sensor device models must be uploaded to your Oracle Internet of Things Cloud Service 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:

```json
{
    "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 Internet of Things Cloud Service instance:

```bash
curl -k -u "<username>:<password>" https://<your_Oracle_IoT_CloudService_instance>:<instance_port_number>/iot/api/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": "0,80",
      "writable": true,
      "defaultValue":80
    }
  ]
}
```
"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: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 Internet of Things Cloud Service:

```bash
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:

```bash
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"
```

## 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.

Deploy the Sample Gateway Application

Install the Oracle Internet of Things Cloud Service gateway sample applications to run the sample device adapters.

1. Install and configure the gateway for development. See Install and Configure a Gateway for Application Development.
2. Open a command prompt and then run these commands to build the samples:

```
$ cd $GATEWAY_HOME/samples
$ gradle assembleAll
dist
```

3. If the gateway software is running on the target gateway device, press `Ctrl+C` in the terminal window where the gateway process is running.

4. Copy the JAR files from the development computer to the `$GATEWAY_HOME/samples/dist` folder on the gateway device.

5. Modify the `gateway.properties` file:
   a. Browse to `$GATEWAY_HOME/config/gateway.properties` and then open the `gateway.properties` file in a text editor.
   b. Add this text to the `gateway.properties` file:

```
gateway.auto.start.3= 
   ${gateway.bundles.path}/../samples/dist/HumiditySensor.jar 
   ${gateway.bundles.path}/../samples/dist/HumiditySensorAdapter.jar 
   ${gateway.bundles.path}/../samples/dist/HumiditySensorApplication.jar 
   ${gateway.bundles.path}/../samples/dist/TemperatureSensor.jar 
   ${gateway.bundles.path}/../samples/dist/TemperatureSensorAdapter.jar 
   ${gateway.bundles.path}/../samples/dist/SensorApplication.jar
```
   c. Save and close the `gateway.properties` file.

6. Modify the `gateway.permissions` file:
   a. Browse to `$GATEWAY_HOME/config/gateway.permissions`, and then open the `gateway.permissions` file in a text editor.
   b. Add this text to the `gateway.permissions` file:

```
##################### Samples #####################
#
ALLOW {
   [org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/dist/*"]
   [org.osgi.service.condpermadmin.BundleSignerCondition "CN=IoT Application Developer, OU=Internet of Things, O=Oracle Corporation"]
   ( java.security.AllPermission "" "")
}"Permissions for samples"
```
   c. Save and close the `gateway.permissions` file.

7. (Optional) Add the trusted assets password and trusted assets path values to the `security.properties` file:
   a. Browse to `$GATEWAY_HOME/config/security.properties` and then open the `security.properties` file in a text editor.
b. Add this text to the `security.properties` file.

```
trusted.assets.password=<the password used to encrypt the provisioning file>
trusted.assets.path=<the path where you saved the provisioning file>
```

c. Save and close the `security.properties` file.

8. On the gateway device, run this command to remove the `felix-cache` folder:

```
cd $GATEWAY_HOME
rm -rf data/felix-cache
```

9. Run one of these commands to run the gateway:
   - For Linux use the command: `bash bin/gateway.sh`
   - For Windows, use the command `\bin\gateway.bat`

The output should be similar to the following example:

```
Humidity measured by humidity sensor with endpoint ID: 0-BY = 32.
HumiditySensor (Hardware ID: Manufacturer:0-BE-HumiditySensorDevice-
SerialNumber:0-BE-1) value (33%)
Humidity measured by humidity sensor with endpoint ID: 0-BU = 33.
Temperature measured by temperature sensor with endpoint ID: 0-CA = 56.
Temperature measured by temperature sensor with endpoint ID: 0-CE = 59.
HumiditySensor (Hardware ID: Manufacturer:0-BE-HumiditySensorDevice-
SerialNumber:0-BE-1) value (31%)
HumiditySensor (Hardware ID: Manufacturer:0-BE-HumiditySensorDevice-
SerialNumber:0-BE-2) value (31%)
```

10. Press Ctrl+C to stop the gateway process.

---

### Install the Bluetooth Adapter and Sample Applications in the Gateway

Use the sample Bluetooth adapter and application to learn how to collect data from Bluetooth enabled devices.

---

**Note:**

The sample Bluetooth adapters and applications can only be run with Linux/x86 or Linux/ARM instruction sets. The JAR (Java ARchive) file required to run the Bluetooth samples is located in the `iotcs-gw-linux-samples-<version>` folder.
1. Install and configure the gateway for development. See Install and Configure a Gateway for Application Development.

2. Open a web browser and browse to Oracle Internet of Things Cloud Service Gateway Downloads.

3. Accept the license agreement and then download the `iotcs-gw-samples-<version>.zip` file to your `/iot/gateway` directory.

4. Extract these files in the `iotcs-gw-samples-<version>` folder:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iotcs-gw-samples-&lt;version&gt;.zip</code></td>
<td>Extract only on your gateway application development platform. Contains the source files of the sample device adapters and applications that can be run on the Oracle Internet of Things Cloud Service Gateway. These samples are platform independent and can be built using the bundled Oracle Internet of Things Cloud Service Gateway SDK.</td>
</tr>
<tr>
<td><code>iotcs-gw-linux-samples-&lt;version&gt;.zip</code></td>
<td>Contains the optional Oracle Internet of Things Cloud Service Gateway runtime extensions that can run on the Linux platform. It contains native components for both ARMv6-HardFloat and x86 platforms. Other sample bundles can also be overlaid on an existing Gateway installation.</td>
</tr>
</tbody>
</table>

5. Open a command prompt and then run these commands to build the samples:

   ```
   cd $GATEWAY_HOME/samples
   gradle assembleAll dist
   ```

6. If the gateway software is running on the target gateway device, press Ctrl+C in the terminal window where the gateway process is running.

7. Copy the JAR files from the development computer to the `$GATEWAY_HOME/samples/dist` folder on the gateway device.

8. Modify the `gateway.properties` file:
   a. For x86 gateway devices, browse to `$GATEWAY_HOME/config/gateway.properties`, open the `gateway.properties` file in a text editor, and then add this text to the `gateway.properties` file:

   ```
   gateway.auto.start.3= \
   ${gateway.bundles.path}/../samples/dist/HeartRateMonitor.jar \
   ```
b. For ARM gateway devices, browse to $GATEWAY_HOME/config/
gateway.properties, open the gateway.properties file in a text editor, 
and then add this text to the gateway.properties file:

gateway.auto.start.3= \n${gateway.bundles.path}/../samples/dist/HeartRateMonitor.jar \n${gateway.bundles.path}/../samples/dist/PulseOximeter.jar \n${gateway.bundles.path}/../samples/dist/BodyWeightScale.jar \n${gateway.bundles.path}/../samples/dist/BloodPressureMonitor.jar \n${gateway.bundles.path}/../samples/lib/linux/arm/Bluettooth.jar 
${gateway.bundles.path}/../samples/dist/BluetoothHealthAdapter.jar \n${gateway.bundles.path}/../samples/lib/linux/PersonalHealthDevice.jar \n${gateway.bundles.path}/../device/dist/PersonalHealthEndpoint.jar \n${gateway.bundles.path}/../device/dist/BluetoothHealthAdpaterImpl.jar \n${gateway.bundles.path}/../device/dist/BluetoothHealthApplication.jar 


c. Save and close the gateway.properties file.

9. Modify the gateway.permissions file:

a. Browse to $GATEWAY_HOME/config/gateway.permissions, and then 
open the gateway.permissions file in a text editor.

b. Add this text to the gateway.permissions file:

```
#################### Device Protocols ##############
#
ALLOW {
[org.osgi.service.condpermadmin.BundleLocationCondition "$\n(gatew.bundles.uri)/../samples/lib/*"]
(java.security.AllPermission "*" "")
} "Permissions for device protocols"
```
### Samples

```java
ALLOW {
    [org.osgi.service.condpermadmin.BundleLocationCondition "$/gatew.bundles.uri/../samples/dist/*"]
    [org.osgi.service.condpermadmin.BundleSignerCondition "CN=IoT Gateway Developer, OU=Internet of Things, O=Your Corporation"]
    (java.security.AllPermission "" "")
} "Permissions for samples"
```

c. Save and close the `gateway.permissions` file.

10. Run this command on the gateway device to remove the `$/GATEWAY_HOME/data/felix-cache` folder:

    ```bash
    cd $GATEWAY_HOME
    rm -rf data/felix-cache
    ```

11. Pair your Bluetooth enabled device with the gateway. See Pairing a Bluetooth Device to a Gateway Device.

12. Run one of these commands to run the gateway:

    For Linux use the command: `bash bin/gateway.sh`

    For Windows, use the command `\bin\gateway.bat`

    The output should be similar to this example:

    ```
    ("jdk.wireless.bluetooth.health.HealthPermission" "00:02:72:DA:01:86:4100" "sink")
    ("jdk.wireless.bluetooth.health.HealthPermission" "00:02:72:DA:01:86:4103" "sink")
    ("jdk.wireless.bluetooth.health.HealthPermission" "00:02:72:DA:01:86:4111" "sink")
    Sample BluetoothHealthApplication application created BluetoothHealthApplication started.
    ```

13. Take measurements with a Bluetooth-enabled personal health device (PHD) that is paired with your gateway. Examples of a PHD device include pulse oximeter, heart rate monitor, blood pressure monitor, or weigh scale.

    Information collected by the PHD is displayed in the Oracle Internet of Things Cloud Service management console. For example, this is the type of information displayed for a pulse oximeter:

    ```
    onChannelConnected called. channelHandle: -207498810
    O2/Pulse measured by 887301756: 98.0 : 68
    ```

14. Press Ctrl+C to stop the gateway process.

### Pairing a Bluetooth Device to a Gateway Device

Your Bluetooth device needs to be paired with your gateway device so that the gateway device can send data from your Bluetooth device to Oracle IoT Cloud Service. Your gateway device will automatically register the Bluetooth device by
sending the Bluetooth device’s endpoint information to the Oracle IoT Cloud Service server.

1. Attach a Bluetooth USB dongle to the gateway device.

2. Login to your gateway device, open a command prompt, and then run this command to confirm that BlueZ software stack version 4.98 or later is installed on your gateway device:

   `hcitool | grep ver`

   The output would be similar to the following:

   `hcitool - HCI Tool ver 4.101`

3. Run this command to list the Bluetooth devices currently paired with the gateway:

   `bt-device -l`

   **Note:**

   If you receive the following message after running that last command, use the next step to modify the udev rules to allow non-root users to access Bluetooth services:

   `bt-device: bluez service is not found Did you forget to run bluetoothd?`

4. (Optional) Modify the udev rules to allow non-root users access to Bluetooth.

   a. As the root user or using the `sudo` command, create a `/etc/udev/rules.d/bluetooth.rules` file.

      `sudo touch /etc/udev/rules.d/bluetooth.rules`

   b. Determine the path to the Bluetooth adapter in your system’s `/sys` folder using the `find` command.

      `find /sys -name bluetooth`

      You should see something like the following

      `/sys/devices/platform/bcm2708_usb/usb1/1-1/1-1.2/1-1.2:1.0/bluetooth`

   c. Add the following line to the `bluetooth.rules` file, using the path found in the previous step for the `DEVPATH` property entry. You also need to append `/hci[0-9]` to the `DEVPATH` value.

      `ACTION="add", SUBSYSTEM="bluetooth", KERNEL="hci[0-9]"`

      `DEVPATH="/sys/devices/platform/bcm2708_usb/usb1/1-1/1-1.2:1.0/bluetooth/hci[0-9]", RUN +=="/bin/sh -c 'cd /sys%p; chown :bluetooth export unexport; chmod g+w export unexport'"`

   d. Add the user to the `bluetooth` group, which should be already present in the system.

      i. (Optional) Add bluetooth group, if needed.

         `sudo groupadd bluetooth`
ii. Add user to to the bluetooth group using the following:

```
sudo usermod -aG bluetooth <username>
```

e. Reboot your gateway device.

5. (Optional) If the Bluetooth device is already showing as paired with the device gateway but you are having problems connecting to the device, remove the pairing information.

```
sudo bt-device -r BT_ADDR
```

where `BT_ADDR` is a colon-separated value. For example, for a Nonin Pulse Oximeter device, it would be similar to `00:1C:05:00:XX:XX`.

6. Determine the device’s Bluetooth address by using the `hcitool scan` command, as illustrated below.

```
hcitool scan
Scanning ...
40:A6:Z8:T7:795:8C      bt-device0
40:X6:D9:G8:58:A8       bt-device1
```

7. To pair the Bluetooth device, turn it off and turn it back on to place it back in discovery mode. For example, for the Nonin Pulse Oximeter, take out its batteries to turn it off and reinsert its batteries in the device to turn it back on and pair it.

8. Set the Bluetooth device aside and type the following command quickly (the pairing period expires in a short period).

```
sudo bt-device -c <BT_ADDR>
```

where `<BT_ADDR>` is a colon separated value. For example, for a Nonin Pulse Oximeter device with a pairing information of `00:1C:05:00:B7:E4`, you would type the following:

```
sudo bt-device -c 00:1C:05:00:B7:E4
```

You would then see something similar to the following:

```
Connecting to: 00:1C:05:00:b7:e4 Agent registered
Device: Nonin_Medical_Inc._577981 (00:1C:05:00:B7:E4)
```

When prompted for the PIN code, locate it on the device or in the device’s user manual, and type it at the prompt. In the following example, `567890` is the sample PIN code:

```
Enter PIN code: 567890
Agent released
```

9. Verify that the Bluetooth device is paired with the gateway device:

```
sudo bt-device -l
```

The output displayed would be similar to the following:

```
Added devices:Nonin_Medical_Inc._577981 (00:1C:05:00:B7:E4)
```
Work with the Sample UART Device Adapter

A sample device I/O Universal Asynchronous Receiver/Transmitter (UART) thermometer adapter is included with Oracle Internet of Things Cloud Service.

The sample device I/O UART adapter is located in the oracle-iotcs-gw-samples-<version>.zip file. Extract the zip file to access the adapter in the <IoTCS-Gateway-install-dir>/iot/gateway/iotcs/gw/samples/deviceio folder.

Note:

The following instructions have been verified with the Raspberry Pi platform only.

1. Install and configure the gateway for development. See Install and Configure a Gateway for Application Development.
2. Open a web browser and browse to Oracle Internet of Things Cloud Service Gateway Downloads.
3. Accept the license agreement and then download the iotcs-gw-samples-<version>.zip file to your /iot/gateway directory.
4. Extract these files in the iotcs-gw-samples-<version> folder:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-gw-samples-&lt;version&gt;.zip</td>
<td>Extract only on your gateway application development platform. Contains the source files of the sample device adapters and applications that can be run on the Oracle Internet of Things Cloud Service Gateway. These samples are platform independent and can be built using the bundled Oracle Internet of Things Cloud Service Gateway SDK.</td>
</tr>
<tr>
<td>iotcs-gw-linux-samples-&lt;version&gt;.zip</td>
<td>Contains the optional Oracle Internet of Things Cloud Service Gateway runtime extensions that can run on the Linux platform. It contains native components for both ARMv6-HardFloat and x86 platforms. Other sample bundles can also be overlaid on an existing Gateway installation.</td>
</tr>
</tbody>
</table>

Note:

You can create your own Oracle Internet of Things Cloud Service Gateway runtime extension bundle for non-Linux target platforms. You can create your own device types, or you can copy the device types provided in the iotcs-gw-linux-samples-<version>.zip bundle and modify them using the right protocol stack for your target platform.
5. Open a command prompt and then run these commands to build the samples:

   cd $GATEWAY_HOME/samples
   gradle assembleAll dist

6. If the gateway software is running on the target gateway device, press Ctrl+C in the terminal window where the gateway process is running.

7. Copy the JAR files from the development computer to the $GATEWAY_HOME/samples/dist folder on the gateway device.

8. Install and configure a USB-to-serial Raspberry Pi I/O Pin compatible adapter on your Raspberry Pi gateway device:

   b. Open a terminal window on your Raspberry Pi console.
   c. Open a text editor and replace this text in the /boot/cmdline.txt file:

      ```
      dwc_otg.lpm_enable=0 console=ttymA0,115200 kgdboc=ttymA0,115200 console=ttym0 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
      ```

      with:

      ```
      dwc_otg.lpm_enable=0 console=ttym0 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
      ```

   d. Comment out this text in the /etc/inittab file:

      ```
      T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
      ```

   e. Reboot your Raspberry Pi gateway device.
   f. Run this command to verify that the user account you are using on the Raspberry Pi gateway is a member of the dialout group. The dialout group has access to the UART port.

      ```
      pi@raspberrypi $ id
      uid=1000(pi) gid=1000(pi) groups=1000(pi),4(adm),20(dialout),
      24(cdrom),27(sudo),29(audio),44(video),46(plugdev),60(games),
      100(users),105(netdev),999(input),1002(spi),1003(gpio)
      ```
g. (Optional) If the user account you’re using is not a member of the dialout group, run this command:

```
sudo usermod -a -G dialout <username>
```

9. On another device such as a MAC OS, Linux, or Windows platform, install and configure a terminal emulator program to interface with the USB-to-serial adapter. For example, TeraTerm for the Windows platform and picocom for the Linux.

10. Modify the `gateway.properties` file:

### Note:

The following steps need to be performed once. Subsequent executions of the gateway do not require you to redo these steps.

a. For x86 gateway devices, browse to `$GATEWAY_HOME/config/gateway.properties`, open the `gateway.properties` file in a text editor, and then add this text to the `gateway.properties` file:

```
gateway.auto.start.3= \\n   ${gateway.bundles.path}/../samples/dist/Thermometer.jar \\n   ${gateway.bundles.path}/../samples/lib/linux/x86/DeviceIO.jar \\n   ${gateway.bundles.path}/../samples/dist/UartThermometerAdapter.jar \\
```

b. For ARM gateway devices, browse to `$GATEWAY_HOME/config/gateway.properties`, open the `gateway.properties` file in a text editor, and then add this text to the `gateway.properties` file:

```
gateway.auto.start.3= \\n   ${gateway.bundles.path}/../samples/dist/Thermometer.jar \\n   ${gateway.bundles.path}/../samples/lib/linux/arm/DeviceIO.jar \\n   ${gateway.bundles.path}/../samples/dist/UartThermometerAdapter.jar \\
```

c. Save and close the `gateway.properties` file.

11. Modify the `gateway.permissions` file:

a. Browse to `$GATEWAY_HOME/config/gateway.permissions`, and then open the `gateway.permissions` file in a text editor.

b. Add this text to the `gateway.permissions` file:

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```

```
ALLOW { 
   org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*" 
   ( java.security.AllPermission "" "") 
} "Permissions for device protocols"
```
12. (Optional) Add the trusted assets password and trusted assets path values to the `security.properties` file:
   
a. Browse to `$GATEWAY_HOME/config/security.properties` and then open the `security.properties` file in a text editor.
   
b. Add this text to the `security.properties` file.

   ```
   trusted.assets.password=<the password used to encrypt the provisioning file>
   trusted.assets.path=<the path where you saved the provisioning file>
   ```

   c. Save and close the `security.properties` file.

13. Start the terminal emulator software and set up your serial port using these values:
   
   - Port: COM3
   - Baud Rate: 115200
   - Data: 8 bit
   - Parity: none
   - Stop: 1 bit
   - Flow control: none
   - Transmit delay: 0 (zero)

14. Run this command to configure the UART adapter serial port name. This must be the same value assigned to your hardware, for example, `ttyAMA0`.

   ```bash
   export GATEWAY_JVM_ARGS=-Dcom.oracle.iot.sample.uart=ttyAMA0
   ```

15. Run this command on the gateway device to remove the `$GATEWAY_HOME/data/felix-cache` folder:

   ```bash
   cd $GATEWAY_HOME
   rm -rf data/felix-cache
   ```

16. On the gateway device, run one of these commands:

   For Linux use the command: `bash bin/gateway.sh`

   For Windows, use the command: `\bin\gateway.bat`

   The input prompt appears in the terminal emulator terminal window.

17. Open a command prompt on your terminal emulator and enter `75.5` and press Enter.
An entry similar to this appears on the gateway device console:

Temperature measured by 2144562925 : 75.5

On a non-Windows platform with the gateway running, run this cURL command in a terminal window:

```
echo -e '\01' |curl -v -k --user username:password --data @- -X PUT "https://<your_IoT_CS_instance_URL>:<your_IoTCS_port>/iot/api/v1/endpoints/{gateway-endpoint-id}/resources/manage/power"
```

On a Windows platform, run the `$GATEWAY_HOME\bin\gateway.bat` command.

18. Press Ctrl+C to stop the gateway process.

### Work with the Sample RS-232 Device Adapter

A sample device I/O RS-232 thermometer adapter is included in the `oracle-iotcs-gw-samples-<version>.zip` file. Extract the zip file to access the adapter in the `<IoTCS-Gateway-install-dir>/iot/gateway/iotcs/gw/samples/deviceio/Rs232ThermometerAdapter` folder.

This hardware is required to complete this procedure:

- USB-to-serial adapter (The RS-232 device adapter sample has been tested with the ATEN UC232A USB-to-Serial Converter).

**Note:**

The Raspberry Pi board does not have a DTR pin. In order to run the RS-232 adapter with the Raspberry Pi, you need a USB-to-serial adapter. Please make sure the USB-to-serial adapter is assigned to serial port `ttyUSB0`. Otherwise, you have to update your source code with the assigned serial port.

- Papouch TM - RS-232 Thermometer.

1. Install and configure the gateway for development. See Install and Configure a Gateway for Application Development.
2. Open a web browser and browse to Oracle Internet of Things Cloud Service Gateway Downloads.
3. Accept the license agreement and then download the `iotcs-gw-samples-<version>.zip` file to your `/iot/gateway` directory.
4. Extract these files in the `iotcs-gw-samples-<version>` folder.
### Work with the Sample RS-232 Device Adapter

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-gw-samples-&lt;version&gt;.zip</td>
<td>Extract only on your gateway application development platform. Contains the source files of the sample device adapters and applications that can be run on the Oracle Internet of Things Cloud Service Gateway. These samples are platform independent and can be built using the bundled Oracle Internet of Things Cloud Service Gateway SDK.</td>
</tr>
<tr>
<td>iotcs-gw-linux-samples-&lt;version&gt;.zip</td>
<td>Contains the optional Oracle Internet of Things Cloud Service Gateway runtime extensions that can run on the Linux platform. It contains native components for both ARMv6-HardFloat and x86 platforms. Other sample bundles can also be overlaid on an existing Gateway installation.</td>
</tr>
</tbody>
</table>

**Note:**

You can create your own Oracle Internet of Things Cloud Service Gateway runtime extension bundle for non-Linux target platforms. You can create your own device types, or you can copy the device types provided in the `iotcs-gw-linux-samples-<version>.zip` bundle and modify them using the right protocol stack for your target platform.

5. Open a command prompt and then run these commands to build the samples:

   ```
   cd $GATEWAY_HOME/samples
   gradle assembleAll dist
   ```

6. If the gateway software is running on the target gateway device, press Ctrl+C in the terminal window where the gateway process is running.

7. Copy the JAR files from the development computer to the `$GATEWAY_HOME/samples/dist` folder on the gateway device.

8. Modify the `gateway.properties` file:

   a. For x86 gateway devices, browse to `$GATEWAY_HOME/config/gateway.properties`, open the `gateway.properties` file in a text editor, and then add this text to the `gateway.properties` file:

   ```
   gateway.auto.start.3= \
   ${gateway.bundles.path}/../samples/dist/Thermometer.jar \
   ${gateway.bundles.path}/../samples/lib/linux/x86/DeviceIO.jar \
   ${gateway.bundles.path}/../samples/dist/<application name>.jar
   ```

   b. For ARM gateway devices, browse to `$GATEWAY_HOME/config/gateway.properties`, open the `gateway.properties` file in a text editor, and then add this text to the `gateway.properties` file:

   ```
   gateway.auto.start.3= \
   ${gateway.bundles.path}/../samples/dist/Thermometer.jar \
   ```
c. Save and close the gateway.properties file.

9. Modify the gateway.permissions file:
   a. Browse to $GATEWAY_HOME/config/gateway.permissions, and then open the gateway.permissions file in a text editor.
   b. Add this text to the gateway.permissions file:

```
ALLOW {
    [org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*"]
    ( java.security.AllPermission "" "")
} "Permissions for device protocols"

ALLOW {
    [org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/dist/*"]
    [org.osgi.service.condpermadmin.BundleSignerCondition "CN=IoT Gateway Developer, OU=Internet of Things, O=Your Corporation"]
    ( java.security.AllPermission "" "")
} "Permissions for samples"
```

c. Save and close the gateway.permissions file.

10. Run this command to configure the serial port name for the RS-232 adapter. This value must be the same as the value assigned to your hardware. For example, ttyUSB0.

    ```bash
    export GATEWAY_JVM_ARGS=-Dcom.example.iot.sample.rs232=ttyUSB0
    ```

11. Run this command on the gateway device to remove the $GATEWAY_HOME/data/felix-cache folder:

    ```bash
    cd $GATEWAY_HOME
    rm -rf data/felix-cache
    ```

12. On the gateway device, run this command to start the gateway:

    ```bash
    cd $GATEWAY_HOME
    sudo -E sh bin/gateway.sh
    ```

    When the gateway starts, output similar to the following should appear:

    ```
    Temperature measured by 0-BY 25.9
    ```

    These values are returned if an error occurs: 32767.0 (0x7FFF) or 65535.0 (0xFFFF)

    Possible reasons for these error values are as follows:
• The sensor is not configured properly.
• The sensor is not connected or the wiring is bad.
• There are missing bytes in the sensor data frame.

Work with the Sample RS-485 Device Adapter

A sample device I/O RS-485 thermometer adapter is included in the oracle-iotcs-gw-samples-<version>.zip file. Extract the zip file to access the adapter in the <IoTCS-Gateway-install-dir>/iot/gateway/iotcs/gw/samples/deviceio/Rs485ThermometerAdapter folder.

This hardware is required to complete this procedure:
• RS485 to RS232 Converter - ATC-107N
• Papouch TM - RS-485 Thermometer with Spinel and Modbus protocol
• RockPower NT 5 Power Supply, Euro 230V - 9V DC/1300 mA
• 2-1mm Female Terminal Adapter

1. Install and configure the gateway for development. See Install and Configure a Gateway for Application Development.

2. Open a web browser and browse to Oracle Internet of Things Cloud Service Gateway Downloads.

3. Accept the license agreement and then download the iotcs-gw-samples-<version>.zip file to your /iot/gateway directory.

4. Extract these files in the iotcs-gw-samples-<version> folder:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iotcs-gw-samples-&lt;version&gt;.zip</td>
<td>Extract only on your gateway application development platform. Contains the source files of the sample device adapters and applications that can be run on the Oracle Internet of Things Cloud Service Gateway. These samples are platform independent and can be built using the bundled Oracle Internet of Things Cloud Service Gateway SDK.</td>
</tr>
<tr>
<td>iotcs-gw-linux-samples-&lt;version&gt;.zip</td>
<td>Contains the optional Oracle Internet of Things Cloud Service Gateway runtime extensions that can run on the Linux platform. It contains native components for both ARMv6-HardFloat and x86 platforms. Other sample bundles can also be overlaid on an existing Gateway installation.</td>
</tr>
</tbody>
</table>
Note:

You can create your own Oracle Internet of Things Cloud Service Gateway runtime extension bundle for non-Linux target platforms. You can create your own device types, or you can copy the device types provided in the `iotcs-gw-linux-samples-<version>.zip` bundle and modify them using the right protocol stack for your target platform.

5. Open a command prompt and then run these commands to build the samples:

```
cd $GATEWAY_HOME/samples
gradle assembleAll dist
```

6. If the gateway software is running on the target gateway device, press Ctrl+C in the terminal window where the gateway process is running.

7. Copy the JAR files from the development computer to the `$GATEWAY_HOME/samples/dist` folder on the gateway device.

8. Using the ModBus configurator tool from Papouch, change the default sensor settings from `protocol=Spinel, address=0x31, baudrate=9600, bits=8, parity=N, stopbit=1` to `protocol=Modbus, address=0x31, baudrate=9600, bits=8, parity=N, stopbit=1`.

9. Modify the `gateway.properties` file:
   a. For x86 gateway devices, browse to `$GATEWAY_HOME/config/gateway.properties`, open the `gateway.properties` file in a text editor, and then add this text to the `gateway.properties` file:

   ```
gateway.auto.start.3= \
   ${gateway.bundles.path}/../samples/dist/Thermometer.jar \
   ${gateway.bundles.path}/../samples/lib/linux/x86/DeviceIO.jar \
   ${gateway.bundles.path}/../samples/dist/<application name>.jar
   ```

   b. For ARM gateway devices, browse to `$GATEWAY_HOME/config/gateway.properties`, open the `gateway.properties` file in a text editor, and then add this text to the `gateway.properties` file:

   ```
gateway.auto.start.3= \
   ${gateway.bundles.path}/../samples/dist/Thermometer.jar \
   ${gateway.bundles.path}/../samples/lib/linux/arm/DeviceIO.jar \
   ${gateway.bundles.path}/../samples/dist/<application name>.jar
   ```

c. Save and close the `gateway.properties` file.

10. Modify the `gateway.permissions` file:
   a. Browse to `$GATEWAY_HOME/config/gateway.permissions` and then open the `gateway.permissions` file in a text editor.
   b. Add this text to the `gateway.permissions` file:

   ```
   ################### Device Protocols #####################
   ALLOW {
   [org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/lib/*"]
   }
   ```
c. Save and close the `gateway.permissions` file.

11. Run this command to configure the serial port name for the RS-485 adapter. This must be the same value as what’s assigned to your hardware, for example, `ttyUSB0`.

   ```bash
   export GATEWAY_JVM_ARGS=-Dcom.example.iot.sample.rs485=ttyUSB0
   ```

12. Run this command on the gateway device to remove the `$GATEWAY_HOME/data/felix-cache` folder:

   ```bash
   cd $GATEWAY_HOME
   rm -rf data/felix-cache
   ```

13. On the gateway device, run this command to start the gateway:

   ```bash
   cd $GATEWAY_HOME
   sudo -E sh bin/gateway.sh
   ```

   When the gateway starts, output similar to the following should appear:

   ```
   Temperature measured by 0-BY 25.9
   ```

   These values are returned if an error occurs: 32767.0 (0x7FFF) or 65535.0 (0xFFFF)

   Possible reasons for these error values are as follows:
   - The sensor is not configured properly.
   - The sensor is not connected or the wiring is bad.
   - There are missing bytes in the sensor data frame.

---

**Work with the Gateway Service Sample**

Use the gateway service sample to learn how to create and use a service.

1. Install and configure the gateway for development. See Install and Configure a Gateway for Application Development.

2. Open a web browser and browse to Oracle Internet of Things Cloud Service Gateway Downloads.
3. Accept the license agreement and then download the `iotcs-gw-samples-<version>.zip` file to your `/iot/gateway` directory.

4. Extract these files in the `iotcs-gw-samples-<version>` folder:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iotcs-gw-samples-&lt;version&gt;.zip</code></td>
<td>Extract only on your gateway application development platform. Contains the source files of the sample device adapters and applications that can be run on the Oracle Internet of Things Cloud Service Gateway. These samples are platform independent and can be built using the bundled Oracle Internet of Things Cloud Service Gateway SDK.</td>
</tr>
<tr>
<td><code>iotcs-gw-linux-samples-&lt;version&gt;.zip</code></td>
<td>Contains the optional Oracle Internet of Things Cloud Service Gateway runtime extensions that can run on the Linux platform. It contains native components for both ARMv6-HardFloat and x86 platforms. Other sample bundles can also be overlaid on an existing Gateway installation.</td>
</tr>
</tbody>
</table>

**Note:**
You can create your own Oracle Internet of Things Cloud Service Gateway runtime extension bundle for non-Linux target platforms. You can create your own device types, or you can copy the device types provided in the `iotcs-gw-linux-samples-<version>.zip` bundle and modify them using the right protocol stack for your target platform.

5. Open a command prompt and then run these commands to build the samples:

   ```
   cd $GATEWAY_HOME/samples
   gradle assembleAll dist
   ```

6. If the gateway software is running on the target gateway device, press `Ctrl+C` in the terminal window where the gateway process is running.

7. Copy the JAR files from the development computer to the `$GATEWAY_HOME/samples/dist` folder on the gateway device.

8. Modify the `gateway.properties` file:

   a. Browse to `$GATEWAY_HOME/config/gateway.properties` and then open the `gateway.properties` file in a text editor.

   b. Add this text to the `gateway.properties` file:

   ```
   gateway.auto.start.3= \
   ${gateway.bundles.path}/../samples/dist/SampleContract.jar \ 
   ${gateway.bundles.path}/../samples/dist/SampleProvider.jar \ 
   ${gateway.bundles.path}/../samples/dist/SampleConsumer.jar
   ```

   c. Save and close the `gateway.properties` file.
9. Modify the gateway.permissions file:
   a. Browse to $GATEWAY_HOME/config/gateway.permissions, and then open the gateway.permissions file in a text editor.
   b. Add this text to the gateway.permissions file:

```
################### Samples ####################
#
ALLOW {
    [org.osgi.service.condpermadmin.BundleLocationCondition "${gateway.bundles.uri}/../samples/dist/*"]
    [org.osgi.service.condpermadmin.BundleSignerCondition "CN=Device Software Developer, OU=Internet of Things, O=Example Corporation"]
    ( java.security.AllPermission " " )
} "Permissions for samples"
```
   c. Save and close the gateway.permissions file.

10. (Optional) Add the trusted assets password and trusted assets path values to the security.properties file:
   a. Browse to $GATEWAY_HOME/config/security.properties and then open the security.properties file in a text editor.
   b. Add this text to the security.properties file:

```
trusted.assets.password=<the password used to encrypt the provisioning file>
trusted.assets.path=<the path where you saved the provisioning file>
```
   c. Save and close the security.properties file.

11. Run this command on the gateway device to remove the $GATEWAY_HOME/data/felix-cache folder:

```
    cd $GATEWAY_HOME
    rm -rf data/felix-cache
```

12. Run one of these commands to run the gateway:

For Linux use the command:
```
    bash bin/gateway.sh
```
For Windows, use the command
```
    \bin\gateway.bat
```
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 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
• Specify Devices as Third Party Partner Devices in Oracle IoT Cloud Service
• Register Devices with Oracle IoT Cloud Service
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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Resource Name (URN)</td>
<td>A device model uses the URN as its unique ID.</td>
</tr>
<tr>
<td>Name</td>
<td>Specify a descriptive name for the device.</td>
</tr>
<tr>
<td>Description</td>
<td>Enter a device description.</td>
</tr>
<tr>
<td>Attributes</td>
<td>A device model's attributes represent the basic variables that the device supports.</td>
</tr>
</tbody>
</table>

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

```json
{
    "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":null,
            "range":null,
            "alias":null,
            "writable":false
        }
    ]
}
```
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:

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thirdPartyPartnerName</td>
<td>Specify the name of the third party vendor.</td>
</tr>
<tr>
<td>thirdPartyPartnerDescr</td>
<td>Enter a description for the third party device.</td>
</tr>
<tr>
<td>thirdPartyPartnerUrl</td>
<td>Provide the URL of the third party dashboard.</td>
</tr>
</tbody>
</table>

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.
Troubleshoot

Review the following topics to resolve issues with Oracle Internet of Things Cloud Service.

Topics:

• Troubleshoot Gateway Issues

Troubleshoot Gateway Issues

Use the information in this topic to resolve issues that you may have encountered while working with the gateway.

To determine the issue, look at the following location or files:

• Gateway runtime console output: This displays the output while the gateway is running
• $GATEWAY_HOME/data/logs/bootstrap.log: This displays the log output while bootstrapping the gateway
• $GATEWAY_HOME/data/logs/gateway-N.N.log: This displays the log output while the gateway is running
• $GATEWAY_HOME/data/logs/derby-N.log: This displays the log output of the Derby database that the gateway uses

The issues and their solutions:

• Issue: Gateway does not respond.
  
  Description: The gateway installed on your device has stopped working.
  
  Solution: Connect to your gateway device using an SSH tool and review the bootstrap log or gateway log files to determine the cause of the problem. Resolve the cause of the issue, if possible, and restart the gateway.

• Issue: Exception message: java.security.cert.CertPathValidatorException:  
  Path does not chain with any of the trust anchors

  Description: An exception occurred when validating the certification path.

  Solution:

  – Review the Gateway Security Concepts and verify if you have missed out any step.

  – Run this command to verify code.truststore has system certificates:

    keytool

    -list -storetype JKS -keystore config\code.truststore. This message should be returned:

    Keystore type: JKS
    Keystore provider: SUN
Your keystore contains 4 entries

verisignrootca, Apr 14, 2015, trustedCertEntry,
verisignrootcacode, Apr 14, 2015, trustedCertEntry,
verisignrootcag5, Apr 14, 2015, trustedCertEntry,
56:BE:3D:9B:67:44:A5:E5

iot application developer, Jul 25, 2016, trustedCertEntry,
Certificate fingerprint (SHA1): 90:38:2F:47:1D:EF:

If only iot application developer, <date>, trustedCertEntry is returned, the system certificates are corrupt or they have been deleted. Extract the config\code.truststore file from the gateway and complete Create a Self-Signed Certificate for Gateway Applications.

• Issue: Gateway does not start.
Description: Gateway cannot communicate with Oracle IoT Cloud Service.
Solution:
– Check if you can reach the IoT Cloud Service server by sending a ping.
– Check that the provisioning file contains the correct Activation ID and Activation Secret. If the values are wrong, then download the file again or create a new device registration.

• Issue: Failed to enroll with given Gateway ID / secret data.
Description: The Activation ID and/or Activation Secret are incorrect in your provisioning file.
Solution:
– Check Oracle IoT Cloud Service to verify the Activation ID and Activation Secret are correct and create a new device registration if it's required.
– Check that the provisioning file contains the correct Activation ID and Activation Secret. If the values are wrong, then download the file again or create a new device registration.
– Check that the provisioning file contains the correct server host and server port. If the values are wrong, then download the file again or create a new device registration.
– Check that the provisioning file contains the certificates for signing your application in Oracle IoT Cloud Service.

• Issue: Permission errors.
Description: Permission errors usually occur for the gateway bundles or the application bundles due to security issues.
Solution:
– All bundles that run in the gateway device must be signed with the appropriate certificates.
– The certificates used to sign the bundle must be stored in the trusted assets store. See Gateway Security Concepts.
– The certificate CN must be used for the BundleSignerCondition for the bundles in $GATEWAY_HOME/config/gateway.permissions when assigning a BundleSignerCondition.

• Issue: Wiring errors.

Description: Wiring errors usually occur when the OSGi framework tries to start a bundle and/or “connect” two bundles together.

Solution:
– Check $GATEWAY_HOME/config/gateway.permissions to make sure your bundle has the proper permissions.
– Make sure the bundle is signed with the certificates in the provisioning file.
– If one bundle is dependent on another, make sure that the bundle being depended on is exporting the correct packages and that it has the appropriate permissions, and check that the dependent bundle is importing the required packages and that it has the appropriate permissions.

• Issue: HTTP Response Code as 200 when you run gradle deploy.

Description: The gateway is unreachable for deployment.

Solution:
– Verify that in the Oracle IoT Cloud Service UI, the gateway device is running and activated.
– Verify that if the gateway is running behind a firewall, then the proxy information is provided in the gradle.properties file.