HPE Universal Internet of Things Platform

Onboarding Devices User Guide
Release 1.2

Hewlett Packard Enterprise

Notices

Legal notice

Printed in the US

Trademarks

Contents

List of tables

List of figures

Preface

This section provides information on the intention of the document, intended audience of the document, document history, typological conventions, related documents and the acronyms and abbreviations used.

About this guide

This document describes the process of onboarding devices on HPE UIoT Platform. This onboarding process applies to all HPE UIoT AaaS platform tenants.

Audience

This document is intended for system integrators and administrators who are responsible for onboarding various entities on the UIoT Platform as part of the solution delivery for customer deployments.

HPE UIoT Platform user documentation

The HPE UIoT Platform documentation set includes the following documents.

Guide	Description
User Guide	Explains the tasks that a user can perform in UIoT DSM GUI.
Installation and Configuration Guide	Describes the steps to install and configure HPE UIoT Platform.
External Interfaces Guide	Describes the interface definition between an external client application and HPE UIoT Platform.
Feature Descriptions	Describes the features of HPE UIoT Platform.
Solution Description	Describes HPE UIoT Platform, its components, modules and features.
High Level Design	Explains the high level design of HPE UIoT Platform.
Performance and Sizing Guide	Provides guidelines to extract high performance from UIoT Platform.
Companion Product Descriptions	Describes the HPE products that are packaged with HPE UIoT Platform
Solution Brief	Briefly introduces HPE UIoT Platform and its components.
Deployment High Level Design	Explains the deployment architecture.
Sizing Guide	Provides guidelines for sizing the components of HPE UIoT Platform.
Onboarding Devices User Guide	Provides instructions to onboard a new device into HPE UIoT Platform.
Onboarding Protocols User Guide	Provides instructions to onboard a new protocol into HPE UIoT Platform.
Onboarding Application User Guide	Provides instructions to onboard a new tenant into HPE UIoT Platform.
Onboarding Application User Guide	Provides instructions to onboard a new application into HPE UIoT Platform.
Reporting platform traffic	Describes the platform traffic report of HPE UIoT Platform.
Security Guide	Describes the security features implemented in HPE UIoT Platform.

Document history

Table 1: Document history

Edition	Version	Date	Description
1.0	HPE UIoT Platform 1.2	June 2016	First version.

Chapter 1
Introduction

The UIoT Platform is capable of onboarding new use cases defined by an application and a device type from any industry, and manage a whole lifecycle from the time the device is on-boarded until it's removed. The platform enables connection and information exchange between heterogeneous IoT devices and applications. It also simplifies integrating diverse devices with different communication protocols.

The UIoT Platform facilitates control points on data, so you can remotely manage millions of IoT devices for smart applications on the same multi-tenant platform.

The DSM module manages the end-to-end lifecycle of the IoT services and associated devices, including provisioning, configuration, and monitoring; IoT devices. Meanwhile, the NIP component provides a connected devices framework for managing and communicating with disparate IoT devices, and communicating over different types of underlying networks.

The DAV component is responsible for ensuring security of the platform including the registration of IoT devices, unique identification of devices and supporting data communication only with trusted devices.

1.1 Prerequisites

Following are the prerequisites for onboarding UIoT enterprise:

- All commercial agreements between HPE and clients are in place
- The IP connectivity between the UIoT client platform and the enterprise is already in place.
- Users and tenants are already on-boarded to the UIoT Platform.
  For more details, see the *HPE Universal Internet of Things Platform Onboarding Tenants User Guide*.
- Device controllers, are already on-boarded to the UIoT Platform.
  For more details, see the *HPE Universal Internet of Things Platform Onboarding Protocols User Guide*.

1.2 Device onboarding process

To onboard devices to the UIoT Platform, perform the following procedures.

1. Create device profile
2. Upload this device profile to the UIoT Platform
3. Add devices.
4. Register the devices on UIoT DSM portal or UIoT Platform console.
5. Develop and import required codecs
6. Integrate external keys
7. Certify or verify the devices

Chapter 2
Device profiles

A device profile provides an overall structure and schema of devices added to the UIoT Platform. The device profile is an XML-based file derived from a predefined application-specific XSD, which is used to capture the specifics on devices that can be used by the UIoT Platform to identify these devices. The specifics include the following information:

- General information about a device such as manufacturer or model of the device.
- Technical information such as the protocol used and the various attributes of that protocol.

Typically, a device vendor sends the information in an XML format to the service provider. The XML file is uploaded to the UIoT Platform DSM portal, which creates a logical layout for devices of that particular kind. All devices that are added to the UIoT Platform under a device profile, inherits the properties and attributes of that device profile.

2.1 Creating a device profile

You can create a device profile by creating an XML file with the following elements.

Table 2 Device profile elements

Element	Description
Manufacturer	Name of the device manufacturer.
Model	Model of the device.
Version	Version of the device.
DeviceType	Type of the device. • Sensor • Gateway
DeviceSubType	Provide the details of the function for which it is used. For example, temperature, humidity, and so on.
TransportChannel	Enter a unique name for each type of device. Name should also indicate the kind of protocol used. This ID is used in the device profile to identify the DC for each device.
Device-Description	Description of the device to be displayed on the UI.
ClassOfDevice	Classification of devices. For example, categorizing the devices to home devices or industrial devices.
DeviceProfileType	Type of device profile. For example, HPEIOT.
MessageFormat	This element is used as a codec identifier, which indicates the codec library. If a message should be decoded, before decoding, this element is referred to check which codec library to use for decoding.
Parameter	Dynamic set of parameters.
OntologyReference	A constant value. This element references the device profile schema, which is available at `http://www.hp.com/schema/m2m/"/`.
AssetParams	This element is used for populating the fields dynamically for every device. If the `Category` attribute is set to `IoT`, all attributes are displayed as fields in the UIoT portal under general asset parameters. If the `Category` attribute is set to `LoRa`, all fields appear under the `LoRa` pane.
Capability	All capabilities like for a sensor, to sense humidity or temperature can be configured under this element.

Following is a sample of the device profile:

```
<?xml version="1.0" standalone="yes"?>
<DeviceProfile xmlns="http://www.hp.com/schema/m2m/">
    <Metadata>
        <Manufacturer>Adeunis</Manufacturer>
        <Model>DEM Extended Demokit</Model>
        <Version>1</Version>
        <DeviceType>SENSOR</DeviceType>
        <DeviceSubType>Temperature</DeviceSubType>
        <TransportChannel>LoRa</TransportChannel>
        <Device-Description>Adeunis DEM Extended Demokit device</Device-
Description>
        <ClassOfDevice>DEFAULT</ClassOfDevice>
        <DeviceProfileType>HPIOT</DeviceProfileType>
       <MessageFormat>AdeunisNetCoverage</MessageFormat>
        <Parameter name="String" type="long-unsigned"></Parameter>
        <OntologyReference xmlns:ns1="http://www.hp.com/schema/m2m/"/>

         <AssetParams ParamName="AppEUI" DisplayName="AppEUI" Mandatory="true"
ReadOnly="true" DataType="String" MinLength="16" MaxLength="16" DefaultValue=""
Decoding="HexToBaseEncode" Category="LoRa"/>
         <AssetParams ParamName="DevEUI" DisplayName="DevEUI" Mandatory="true"
ReadOnly="true" DataType="String" MinLength="16" MaxLength="16" DefaultValue=""
Decoding="HexToBaseEncode" Category="LoRa"/>

         <AssetParams ParamName="KeyManagerID" DisplayName="KeyManagerID"
Mandatory="false" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue="KeyManagerID"  Decoding="none" Category="LoRa"/>
         <AssetParams ParamName="ServiceProfileID"
DisplayName="ServiceProfileID" Mandatory="false" ReadOnly="false"
DataType="String" MinLength="0" MaxLength="255" DefaultValue=""  Decoding="none"
Category="LoRa"/>
         <AssetParams ParamName="DeviceProfileID" DisplayName="DeviceProfileID"
Mandatory="false" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>
         <AssetParams ParamName="DeviceProfileID" DisplayName="DeviceProfileID"
Mandatory="false" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>

         <AssetParams ParamName="ClientID" DisplayName="ClientID"
Mandatory="true" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>
         <AssetParams ParamName="DeviceID" DisplayName="DeviceID"
Mandatory="true" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>

         <AssetParams ParamName="Custom1" DisplayName="Custom1"
Mandatory="false" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>
         <AssetParams ParamName="Custom2" DisplayName="Custom2"
Mandatory="false" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>
         <AssetParams ParamName="Custom3" DisplayName="Custom3"
Mandatory="false" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>
         <AssetParams ParamName="Custom4" DisplayName="Custom4"
Mandatory="false" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>
         <AssetParams ParamName="Custom5" DisplayName="Custom5"
Mandatory="false" ReadOnly="false" DataType="String" MinLength="0"
MaxLength="255" DefaultValue=""  Decoding="none" Category="LoRa"/>


         <AssetParams ParamName="AdministrativeState"
DisplayName="AdministrativeState" Mandatory="false" ReadOnly="false"
```

```
DataType="String" MinLength="0" MaxLength="255" DefaultValue="Locked"
Decoding="none" Category="LoRa"/>

        <AssetParams ParamName="DecodingEnabled" DisplayName="DecodingEnabled"
Decoding="none"  Mandatory="false" ReadOnly="false" DataType="boolean"
DefaultValue="" Category="IoT"/>
       <AssetParams ParamName="latitude" DisplayName="Network Located Latitude"
Decoding="none"  Mandatory="false" ReadOnly="false" DataType="String"
DefaultValue=""  Category="IoT"/>
         <AssetParams ParamName="longitude" DisplayName="Network Located
Longitude"  Decoding="none"  Mandatory="false" ReadOnly="false"
DataType="String"  DefaultValue="" Category="IoT"/>
         <AssetParams ParamName="latitude1" DisplayName="Fixed Latitude"
Decoding="none"  Mandatory="false" ReadOnly="false" DataType="String"
DefaultValue="0.0"  Category="IoT"/>
         <AssetParams ParamName="longitude1" DisplayName="Fixed Longitude"
Decoding="none" Mandatory="fale" ReadOnly="false" DataType="String"
DefaultValue="o.o" Category="IoT"/>
         <AssetParams ParamName="latlongpref" DisplayName="Latlong Preference1"
Decoding="none" Mandatory="false" ReadOnly="false" DefaultValue=""
DataType="Choice" ChoiceElements="networkLocated,selfLocated,fixed"
Category="IoT"/>
         <AssetParams ParamName="latitude2" DisplayName="Self Located Latitude"
Decoding="none"  Mandatory="false" ReadOnly="false" DataType="String"
DefaultValue="" Category="IoT"/>
         <AssetParams ParamName="longitude2" DisplayName="Self Located
Longitude"  Decoding="none"  Mandatory="false" ReadOnly="false"
DataType="String" DefaultValue=""  Category="IoT"/>
        <AssetParams ParamName="AppKey" DisplayName="Security"  Decoding="none"
Mandatory="false" ReadOnly="false" DataType="boolean" DefaultValue=""
Category="IoT"/>
         <AssetParams ParamName="AppEUI" DisplayName="AppEUI" Mandatory="true"
ReadOnly="true" DataType="String" MinLength="16" MaxLength="16" DefaultValue=""
Decoding="HexToBaseEncode" Category="IoT"/>
         <AssetParams ParamName="DevEUI" DisplayName="DevEUI" Mandatory="true"
ReadOnly="true" DataType="String" MinLength="16" MaxLength="16" DefaultValue=""
Decoding="HexToBaseEncode" Category="IoT"/>
   <DeviceIdentifier>
        <identifiers id="1" name="OTA">
               <identifier id="1" name="LNS-DevEUI">DevEUI</identifier>
            <identifier id="2" name="LNS-AppEUI">AppEUI</identifier>
        </identifiers>
        <identifiers id="2" name="OTA1">
               <identifier id="1" name="LNS-DevEUI">DevEUI1</identifier>
            <identifier id="2" name="LNS-AppEUI">AppEUI1</identifier>
        </identifiers>
     </DeviceIdentifier>

    </Metadata>
    <Capabilities>
         <Capability>
               <Category>Identifier</Category>
               <Name>DateTime</Name>
               <DataType>datetime</DataType>
               <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
               <Unit>datetime</Unit>
               <acl>R</acl>
         </Capability>
         <Capability>
               <Category>Identifier</Category>
               <Name>AccelerometerTriggered</Name>
               <DataType>Boolean</DataType>
               <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
               <Unit></Unit>
               <acl>R</acl>
         </Capability>
         <Capability>
```

```
            <Category>Identifier</Category>
            <Name>Button1Pushed</Name>
            <DataType>Boolean</DataType>
            <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
            <Unit></Unit>
            <acl>R</acl>
      </Capability>
      <Capability>
            <Category>Identifier</Category>
            <Name>BatteryVoltage</Name>
            <DataType>Number</DataType>
            <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
            <Unit>mV</Unit>
            <acl>R</acl>
      </Capability>
             <Capability>
            <Category>Identifier</Category>
            <Name>Temperature</Name>
            <DataType>Double</DataType>
            <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
            <Unit>Degree C</Unit>
            <acl>R</acl>
      </Capability>
      <Capability>
            <Category>Identifier</Category>
            <Name>MessageTime</Name>
            <DataType>date</DataType>
            <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
            <Unit>datetime</Unit>
            <acl>R</acl>
      </Capability>
      <Capability>
            <Category>Identifier</Category>
            <Name>Latitude</Name>
            <DataType>Number</DataType>
            <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
            <Unit></Unit>
            <acl>R</acl>
      </Capability>
      <Capability>
            <Category>Identifier</Category>
            <Name>Longitude</Name>
            <DataType>Number</DataType>
            <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
            <Unit></Unit>
            <acl>R</acl>
      </Capability>
      <Capability>
            <Category>Identifier</Category>
            <Name>FCntUp</Name>
            <DataType>Number</DataType>
            <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
            <Unit></Unit>
            <acl>R</acl>
      </Capability>
      <Capability>
            <Category>Identifier</Category>
            <Name>FCntDown</Name>
            <DataType>Number</DataType>
            <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
            <Unit></Unit>
            <acl>R</acl>
      </Capability>
      <Capability>
            <Category>Identifier</Category>
            <Name>RSSI</Name>
            <DataType>Number</DataType>
```

```
                <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
                <Unit>dBM</Unit>
                <acl>R</acl>
          </Capability>
          <Capability>
                <Category>Identifier</Category>
                <Name>SNR</Name>
                <DataType>Number</DataType>
                <Value xmlns:ns1="http://www.hp.com/schema/m2m/"/>
                <Unit></Unit>
                <acl>R</acl>
          </Capability>
    </Capabilities>
</DeviceProfile>
```

2.1.1 Codec-specific attributes in device profile

This section explain the elements and attributes that connect the codecs to device profile.

In the current version, the `<MessageFormat>` element under metadata is the only one that refers to codecs. This element is used as a codec identifier, which indicates the codec library. If a message from a device should be decoded, this element is referred to see which codec library should be used for decoding.

2.2 Upload device profile

A UIoT Platform user with the service provider privileges can upload the device manufacturer profiles. To upload a new device profile file, do the following:

1. Log in to the UIoT Platform DSM portal.
2. Select the **Device Manufacturer Profile** > **Add Device Manufacturer Profile** menu option from the left pane. The **Add Device Profile** window appears.

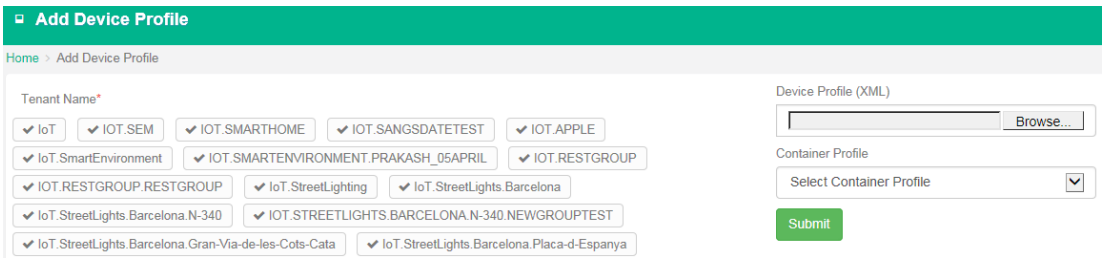


Figure 1 Add Device Profile

3. Select the tenant names to which you want to apply this device profile.
4. Click the **Browse** button and select the device profile file for uploading. You can upload only XML files.
5. Select the container profile from the drop-down list.
6. Click the **Submit** button. A confirmation message appears.

Chapter 3
Adding devices

The UIoT Platform administrator can add gateways and other devices using one of the two options:

- Add individual devices and their details
- Bulk upload devices

3.1 Add a gateway

To add a gateway, do the following:

1. Log in to the UIoT Platform DSM portal.
2. Select the **Device Mgmt.** > **Add Device** menu from the left pane.
   The **Add Device** window appears.

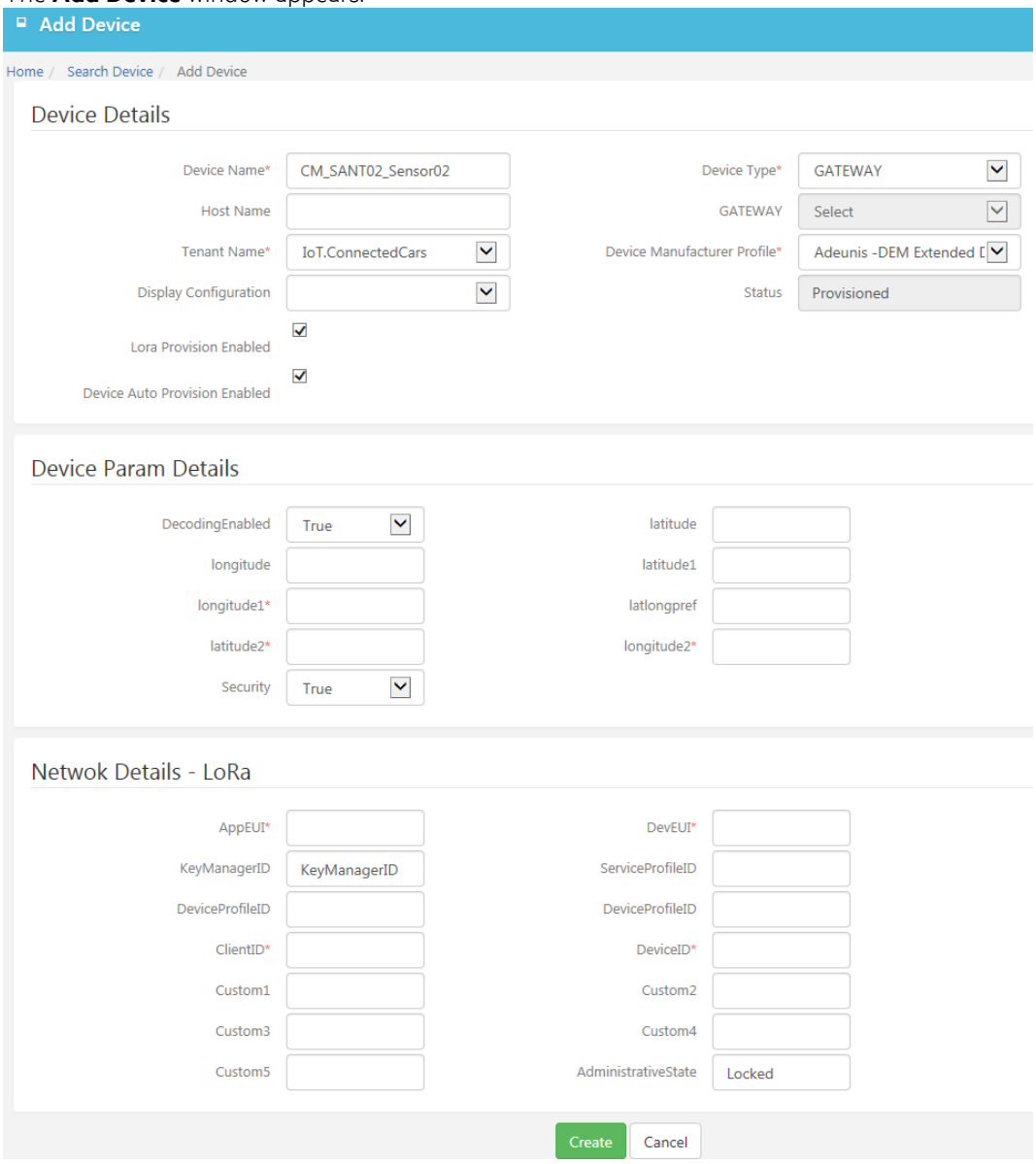


Figure 2 Add devices

3. Enter the required details.

Table 3 Add a gateway

Field	Description
Device Name	Enter the name of the gateway.
Device Type	Select the type of device from the drop-down menu.
Host Name	Enter the host name.
Tenant Name	Tenant to which this asset should be added. Select the tenant from the drop-down list
Device Manufacturer Profile	Enter the device profile details.
Display Configuration	Select a configuration from the drop-down list.
Status	By default, the status is selected as INITIAL.
Device Auto Provision Enabled Value	Select the checkbox to automatically provision the device.
Device Param Details	Based on the Device Manufacturer Profile that you selected, the parameters under this pane changes.

4. Click the **Create** button.
   A confirmation message appears.

3.2 Add individual devices

To add other devices, do the following:

1. Log in to the UIoT Platform DSM portal.
2. Select the **Device Mgmt.** > **Add Device** menu from the left pane.
   The **Add Device** window appears.

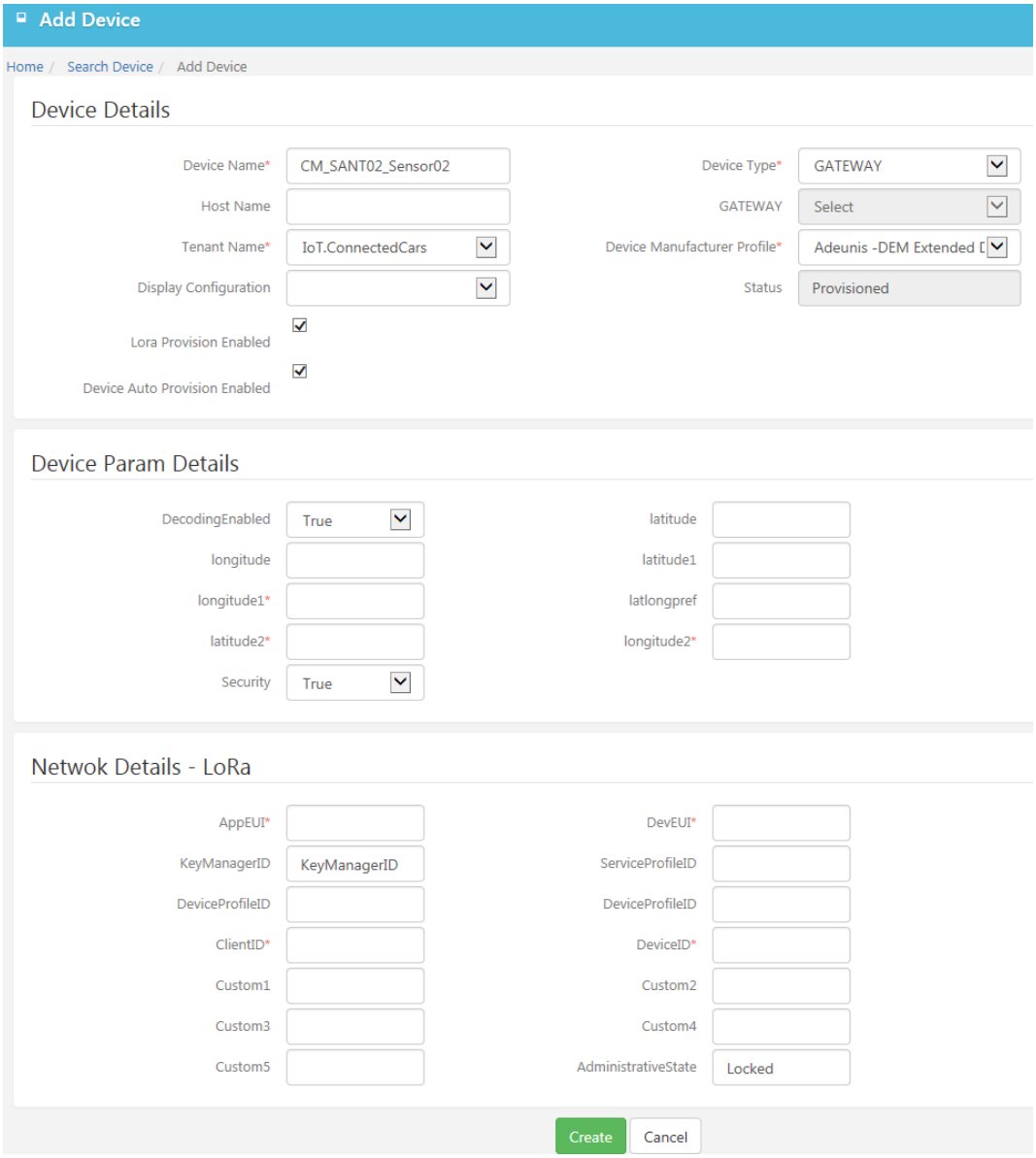


Figure 3 Add devices

3. Enter the required details.

Table 4 Add devices

Field	Description
Device Name	Enter the name of the device.
Device Type	Select the type of device from the drop-down menu.
Host Name	Enter the host name.
GATEWAY	Select a gateway if you are adding a device which is connected through the gateway.
Tenant Name	Tenant to which this asset should be added. Select the tenant from the drop-down list
Device Manufacturer Profile	Enter the device profile details.
Display Configuration	Select a configuration from the drop-down list.
Status	By default, the status is selected as INITIAL.
Device Auto Provision Enabled Value	Select the checkbox to automatically provision the device.
Device Param Details	Based on the Device Manufacturer Profile that you selected, the parameters under this pane changes.

4. Click the **Create** button.
   A confirmation message appears.

3.3 Upload devices in bulk

The UIoT Platform administrator can upload the devices in bulk using the **Import Data** option.

1. Select the **Device Mgmt.** > **Import Data** menu from the left pane.
   The **Import Data** window appears.

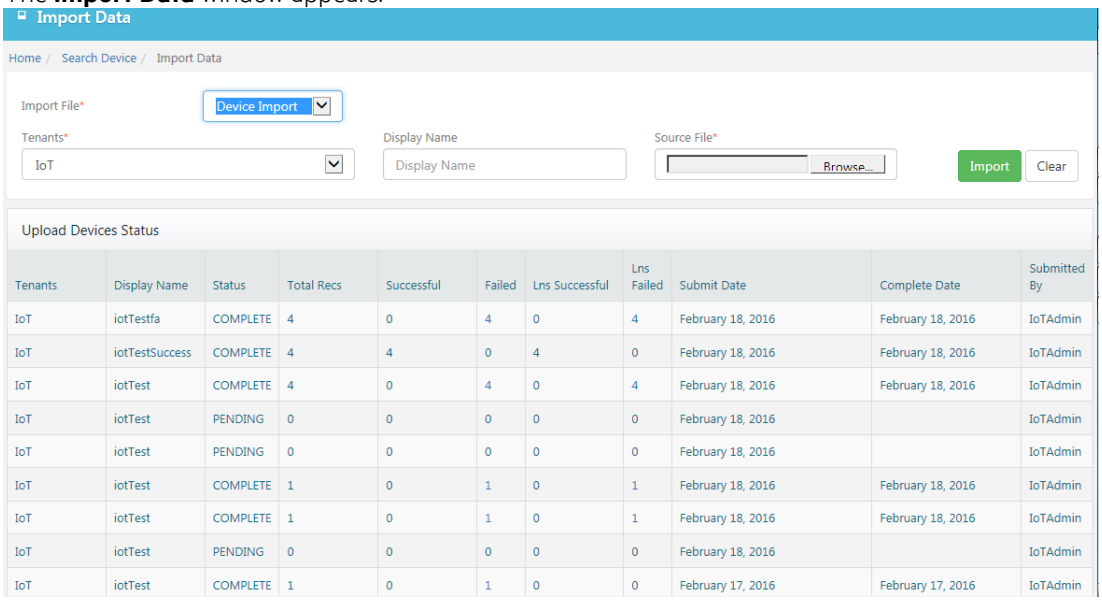


Figure 4 Import devices

2. Enter the following details.

Table 5 Import devices

Field	Description
Import File	Select **Device Import** from the drop-down list.
Tenants	Select the tenant to which you want to add the assets.
Display Name	Enter a name for this tenant of assets.
Source File	Click the **Browse** button to browse and select the file for uploading. The UIoT Platform DSM portal supports only the .txt or .csv file formats. The file should contain the Device Name, Manufacturer, Model and Protocol (mandatory parameters) and Security Type, Security Param, Master Key, Latitude and Longitude (non-mandatory parameters).

3. Click the **Import** button to upload the file.
   A confirmation message appears.

3.3.1 Bulk upload file format

The CSV/text files contains the following parameters.

Table 6 Bulk upload file parameters

Name	Required	Description	Format	Dependency	Min – Max length	Accepted Values
Operation	yes	Operation to be performed on Asset	String	ENUM dependent	Limited to create, update, delete	Create, Update, Delete
Device Name	yes	Device physical name	String	No	Min length-1 Max length-30	Numbers with special character combination Example: Asset_01_new_94
Device Type	Yes	Type of device	String	ENUM dependent	Limited to GATEWAY and SENSOR	GATEWAY SENSOR
Device Profile	yes	Combination of make and model of the device separated by hyphen (-)	String	Dependent on the device_profile_xml table	Min-1 MAX-	Example: Adeunis-Demokit
NetworkProvisioningEnabled	yes	Flag to provision asset on network	Boolean	Boolean only	True false	True false
Host Name	No	Hostname associated with the device	String	No,	Min-1 Max-30	IP or host URL Example: 10.10.10.11,
DisplayConfigurationName	No	Display configuration assigned to the device	String	No	Min-1 Max-30	IP or host URL Example: Displayprfile1,
Asset Params	yes	Parameters of a device such as AppEUI,DevEUI,latlongpref, latitude, longitude	String	Latlongpref limited to “SelfLocated, Network and fixed”	Min 1 Max 30	Example: AppEUI#0088AB5423223233,DevEUI#0088AB5423223231
Asset Params (AppEUI, DevEUI)	Yes	Identifies the device. hexadecimal values are accepted	Hexa String		Min & Max Length 16	AppEUI#0088AB5423223233, DevEUI#0088AB5423223231
Asset Params (Latlongpref = selfLocated)	No	Self-located Usually values are populated based on the payload decoding Note: The lat long data are not required during a device upload.		Lat/Long param names should be latitude2, longitude 2		latlongpref#selfLocated, latitude2#48.4325, longitude2#2.233
Asset Params (Latlongpref = Network)	No	Network located. Lat long values provided by		Lat/Long param names should be		latlongpref#network, latitude#22.4325, longitude#23.23

		LoRa Gateway Antenna Note: The lat long data are not required during the device upload		latitude, longitude		
Asset Params (Latlongpref = Fixed)	No	Fixed User input values for lat long. Note: If the user wants to set fixed Lat long for a device.		Lat/Long param names should be Latitude1, longitude1		latlongpref#fixed, latitude1#88.4325, longitude1#28.233
Asset Params (AppKey = True)	No	Shows whether security key is available for the device or not	Boolean			True / false
Asset Params (DecodingEnabled = True)	No	Shows whether decoding is enabled for the device or not	Boolean			True/false
Network Params	yes	Parameters of the device such as AppEUI,DevEUI, KeyManagerID, ServiceProfileID DeviceProfileID ClientID	String	Depended on the device profile attribute	Depended on the device profile attribute	Example: AppEUI#0088AB5423223233,DevEUI#0088AB5423223231
NetworkParams (AppEUI, DevEUI)	yes	Identifier for the device	String		Defined in the device profile	Example: AppEUI#0088AB5423223233,DevEUI#0088AB5423223231
NetworkParams (KeyManagerID)	yes	Identifies the Key Manager Server which can deliver the device AppSKey	String	If the value is not found in the csv file, the default value LNSInternalKMS is taken.	Max 255 char	KeyManagerID# LNSInternalKMS
NetworkParams (ServiceProfileID)	yes	Identifies a set of parameters that defines the LNS behavior to adapt the device radio parameters	String	If the value is not found in the CSV file, the default	MAX 255 char	Example: ServiceProfileID#NB4

				value NB3 is taken.		
NetworkParams (DeviceProfileID)	No	Identifies a set of parameters that defines the device capabilities (Example: FCntUp/FCntDown 16 or 32 bits format)	String	If not available, a default profile is applied by the LNS.	Max 255 char	Example: DeviceProfileID#12vg4
NetworkParams (ClientID)	Yes	An identifier associated to the client (the service provider) by the IT system	String	LNS does not verify the validity of this value	255 characters max	Example: ClientID#EWM123
NetworkParams (DeviceID)	Yes	An identifier associated to the device by the IT system	String	LNS does not verify if another device is already declared with this value.	255 characters max	Example: DeviceID#67654
NetworkParams (Custom1)	No	Additional field that can be set by the CRM	String	If absent, the value is set to <null> by the LNS.	255 characters max	Example: Custom1#1234Custom1
NetworkParams (Custom2)	No	Additional field that can be set by the CRM	String	If absent, the value is set to <null> by the LNS.	255 characters max	Example: Custom2#1234Custom2
NetworkParams (Custom3)	No	Additional field that can be set by the CRM	String	If absent, the value is set to <null> by the LNS.	255 characters max	Example: Custom3#1234Custom3
NetworkParams (Custom4)	No	Additional field that can be set by the CRM	String	If absent, the value is set to <null> by the LNS.	255 characters max	Example: Custom4#1234Custom4
NetworkParams (Custom5)	No	Additional field that can be set by the CRM	String	If absent, the value is set to <null> by the LNS.	255 characters max	Example: Custom5#1234Custom5
NetworkParams (AdministrativeState)	No	Locked: The device cannot access the network; the messages from this device are ignored	String	ENUM Locked Unlocked If absent, the value	Locked Unlocked	Example: AdministrativeState#Unlocked

		Unlocked: The device can access the network		is set to Unlocked by the LNS.		

Network parameters and asset parameters should be enclosed inside brackets ([and]) and the parameters should be placed in the following order:

1. Network Parameter
2. Asset Parameter

The key and value are separated by a hash (*#*). Sign.

Chapter 4
Manage device data

You should register applications and devices on the UIoT Platform for exchanging or storing device data. UIoT Platform provides a REST based interface for registering devices. After registering the devices, you can access the data from devices, operations, and capabilities from applications by calling their parameters and operations.

4.1 Register a device on UIoT Platform

The following example shows API commands for registering a device, which are sent from a remote gateway or an application.

```
POST: http://206.164.250.140/davc/m2m/HPE_IoT?ty=2

HEADERS:
X-M2M-Origin : ApplicationResourceId
X-M2M-RI : RI_10142015_1345
Authorization: NDIxNDU1NTIwOTk1NjA4NDI4NjpDR1RCU0NFQ1pH

Body

{"m2m:ae":
   {
    "resourceType": 2,
    "resourceID": null,
    "parentID": "842",
    "creationTime": null,
    "lastModifiedTime": null,
    "labels": [],
    "name": "Demo_TV",
    "accessControlPolicyIDs": [],
    "expirationTime": null,
    "announceTo": [],
    "announcedAttribute": [],
    "appName": "Demo_TV",
    "appID": "Demo_TV",
    "aeid": "Demo_TV",
    "pointOfAccess": [
        "http://206.164.250.140/davc//echoTest"
    ],
    "ontologyRef": "http://ontologyUrl",
    "nodeLink": "Demo_TV",
    "childResource": [],
    "containerOrGroupOrAccessControlPolicy": []
   }
}


Response Headers
1.  Status Code: 201 Created
2.  Connection: Keep-Alive
3.  Content-Length: 518
4.  Content-Type: application/vnd.onem2m-res+json
5.  Date: Wed, 14 Oct 2015 06:18:12 GMT
6.  Server: Apache-Coyote/1.1
7.  X-M2M-RI: RI_10142015_1
8.  X-M2M-RSC: 2001
9.  X-M2M-Origin: HPE_IoT
10. Content-Location: HPE_IoT/16da4362d79
```

```
Response Body
{
    "m2m:ae":
    {
        "rty": 2,
        "ri": "843",
        "pi": "842",
        "ct": null,
        "lt": null,
        "lbl": [],
        "rn": "Demo_TV",
        "acpi": [],
        "et": null,
        "at": [],
        "aa": [],
        "apn": "Demo_TV",
        "api": "Demo_TV",
        "aei": "Demo_TV",
        "poa": [
            "http://206.164.250.140/davc//echoTest"
        ],
        "or": "http://ontologyUrl",
        "nl": "843",
        "ch": [],
        "containerOrGroupOrAccessControlPolicy": []
    }
}
```

4.2 Access data from a device

The following example shows the API `read` commands sent from an application to read latest data from a device.

```
GET: http://206.164.250.140/davc/m2m/HPE_IoT/deviceName/default/latest


X-M2M-Origin : ApplicationResourceId
X-M2M-RI : RI_10142015_1346
Authorization: NDIxNDU1NTIwOTk1NjA4NDI4NjpDR1RCU0NFQ1pH

Response
1.  Status Code: 201 Created
2.  Connection: Keep-Alive
3.  Content-Length: 605
4.  Content-Type: text/html
5.  Date: Wed, 14 Oct 2015 07:00:17 GMT
6.  Server: Apache-Coyote/1.1
7.  X-M2M-RI: RI_10142015_1346
8.  X-M2M-RSC: 2001
9.  X-M2M-Origin: HPE_IoT


   Response Body:



[
    {
        "m2m:cin": {
            "rty": 4,
```

```
            "ri": "3_6917201895583037906",
            "pi": "4331878012081649688",
            "ct": "2016-06-13 19:33:24.838",
            "lt": "2016-06-13 19:33:24.838",
            "lbl": [],
            "rn": "Readings",
            "et": "2016-06-13 19:33:24.83",
            "at": [],
            "aa": [],
            "st": 0,
            "cr": null,
            "cnf": null,
            "cs": 0,
            "or": null,


"con":
"{\"AccelerometerTriggered\":\"true\",\"Button1Pushed\":\"true\",\"Temperature\"
:\"27\",\"BatteryVoltage\":\"411\",\"MessageTime\":\"2014-11-
03T10:58:00.148+05:30\",\"Latitude\":\"69\",\"Longitude\":\"103\",\"FCntUp\":\"4
10\",\"RSSI\":\"410\",\"SNR\":\"410\",\"DateTime\":\"2014-11-
03T10:58:00.148+05:30\"}"
        }
    },
    {
        "m2m:cin": {
            "rty": 4,
            "ri": "3_7440385847292547028",
            "pi": "4331878012081649688",
            "ct": "2016-06-14 14:16:01.997",
            "lt": "2016-06-14 14:16:01.997",
            "lbl": [],
            "rn": "36857133a3c",
            "et": "2016-06-14 14:16:01.991",
            "at": [],
            "aa": [],
            "st": 0,
            "cr": null,
            "cnf": null,
            "cs": 0,
            "or": null,



"con":
"{\"AccelerometerTriggered\":\"true\",\"Button1Pushed\":\"true\",\"Temperature\"
:\"27\",\"BatteryVoltage\":\"411\",\"MessageTime\":\"2014-11-
03T10:58:00.148+05:30\",\"Latitude\":\"69\",\"Longitude\":\"103\",\"FCntUp\":\"4
10\",\"RSSI\":\"410\",\"SNR\":\"410\",\"DateTime\":\"2014-11-
03T10:58:00.148+05:30\"}"
        }
    }
]
```


4.3 Post data to UIoT Platform

The following example shows API commands for posting data from a device to the UIoT Platform.

```
POST http://206.164.250.140/davc/m2m/HPE_IoT/deviceName/default?ty=4&rt=3

HEADERS:


X-M2M-Origin : ApplicationResourceId
```

```
X-M2M-RI : RI_10142015_1347
Authorization: NDIxNDU1NTIwOTk1NjA4NDI4NjpDR1RCU0NFQ1pH

Request Body:


{
    "m2m:cin": {

"con":
"{\"AccelerometerTriggered\":\"true\",\"Button1Pushed\":\"true\",\"Temperature\"
:\"27\",\"BatteryVoltage\":\"411\",\"MessageTime\":\"2014-11-
03T10:58:00.148+05:30\",\"Latitude\":\"69\",\"Longitude\":\"103\",\"FCntUp\":\"4
10\",\"RSSI\":\"410\",\"SNR\":\"410\",\"DateTime\":\"2014-11-
03T10:58:00.148+05:30\"}"
}


RESPONSE:
Headers:
1.  Status Code: 201 Created
2.  Connection: Keep-Alive
3.  Content-Length: 951
4.  Content-Type: text/html
5.  Date: Wed, 14 Oct 2015 12:02:57 GMT
6.  Server: Apache-Coyote/1.1
7.  X-M2M-RI: RI_10142015_1347
8.  X-M2M-RSC: 2001
9.  Content-Location: HPE_IoT/16da4362d79/default/36857133a3c
10. X-M2M-Origin: HPE_IoT


Response body:

{
    "m2m:cin": {
        "rty": 4,
        "ri": "HPE_IoT/16da4362d79/default/36857133a3c",
        "pi": "HPE_IoT/16da4362d79/default",
        "ct": "2016-06-14 14:16:01.997",
        "lt": "2016-06-14 14:16:01.997",
        "lbl": [],
        "rn": "36857133a3c",
        "et": "2016-06-14 14:16:01.991",
        "at": [],
        "aa": [],
        "st": 0,
        "cr": null,
        "cnf": null,
        "cs": 0,
        "or": null,


"con":
"{\"AccelerometerTriggered\":\"true\",\"Button1Pushed\":\"true\",\"Temperature\"
:\"27\",\"BatteryVoltage\":\"411\",\"MessageTime\":\"2014-11-
03T10:58:00.148+05:30\",\"Latitude\":\"69\",\"Longitude\":\"103\",\"FCntUp\":\"4
10\",\"RSSI\":\"410\",\"SNR\":\"410\",\"DateTime\":\"2014-11-
03T10:58:00.148+05:30\"}"
    }
}
```

Chapter 5
Managing codecs

Codec is attached as a plug-in component in the UIoT Platform. You can add, enable, or disable codecs dynamically without rebooting or redeploying the DAV module.

NOTE: If a codec is overwritten from the DSM portal UI, reboot or redeploy the DAV module.

5.1 Codec interfaces for third-party

Third-party software responsible for codec development, implements the following contract. This contract is provided as a JAR file to the third-party developers. This JAR has a dependency for both third-party developer and UIoT Platform.

The supporting classes of this abstract class are as follows:

```
public abstract class AbstractCodec implements Codec {
   boolean enabled;
   public boolean isEnabled() {
         return enabled;
   }
   public void setEnabled(boolean enabled) {
         this.enabled = enabled;
   }
}
```

The third-party codec should extend the abstract class. The AbstractCodec implements the codec interface. Third-party codec development must implement the methods of codec interface.

```
public interface Codec {
   public CodecIOBase encode(CodecIOBase codecIOBase) throws CodecException;
   public CodecIOBase decode(CodecIOBase codecIOBase) throws CodecException;
   public String getUniqueCodecIdentifier() throws CodecException;
   public String getCodecName() throws CodecException;
   public String getCodecDescription() throws CodecException;
   public String getCodecVersion() throws CodecException;
}



public class CodecException extends Exception {
   public CodecException(String message) {
        super(message);
    }
}

public class CodecIOBase{
    ExecutionStatus executionStatus;    //Used for returning success(true) or
failure( for encode/decode
    Map<String,String> codecStringInputOutput;
    Map<String,byte[]> codecBinaryInputOutput;
    Map<String,Object> codecObjectInputOutput;
}

public enum ExecutionStatus {
        success(1),
        failure(2),
        suspended(3);
}
```

5.2 Codec Log4J dependency for third-party

Following is the Log4J dependency expressed in terms of Maven.

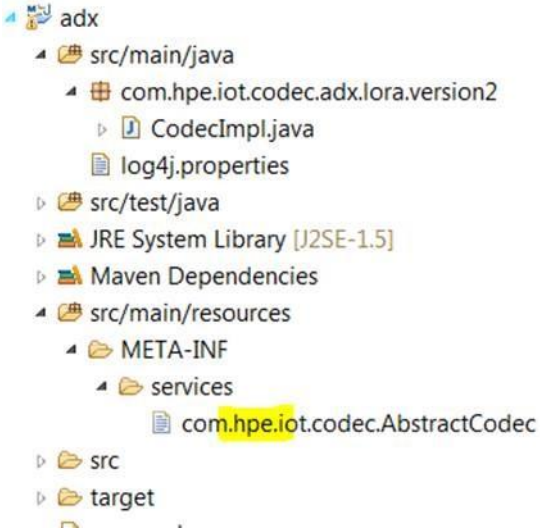
```
<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
</dependency>
```

Each third-party codec includes the unique logging path as described here.

```
log4j.rootLogger=DEBUG, file
log4j.appender.file.File=<CodecImplementor>/<deviceVendor>/<CodecName>/<CodecCla
ssVersion>/<CodecName>.log
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n
```

5.3 Develop third-party codec

UIoT Platform codec server framework uses a ServiceLoader to load the third-party codec. The third party codec must conform to the following directory structure.



The `com.hpe.iot.codec.AbstractCodec` file must contain the qualified name of implementation class that implements codec interface.

For example, `com.hpe.iot.codec.adx.lora.version1_2_2.CodecImpl`, third-party codecs must set the `UniqueCodecIdentifier` with a unique ID provided by the HPE delivery team.

For example:

```
public String getUniqueCodecIdentifier() throws CodecException {
         return "uniqueId";
}
```

This ID is a unique identifier of the codec in the UIoT Platform. Following is a list of unique Codec Identifiers.

Unique Codec Identifier
Bleeper
Adeunis DEM
ARF Pulse Meter
AdeunisNetCoverage
InO_ILD

5.4 UIoT Platform codec server framework

Codec Server Framework uses ServiceLoader to dynamically load the third-party codecs. The key to this design having single CodecContext available to the DAV module. CodecContext is a singleton spring class managed by the DAV root spring application context.

The framework is designed to use only one instance of the Codec. A Codec is stateless. The CodecIOBase is the primary IO for the codec. As CodecIOBase preserves state, a new instance is required for exchange of data.

CodecContext is a spring managed Single class that loads the Codec whichever codec is available under `/hpe/codec` directory during boot strap. Whenever a third-party code jar is available, the "reloadCodec" method of CodecContext should be invoked so that it reloads the codecs.

When a codec is loaded, its default status is disabled. When the codec is enabled by using the "setEnabled" method, it becomes ready for use. Important methods of the CodecContext are as follows:

```
public class CodecContext {
    public CodecIOBase encode(String uniqueCodecIdentifier,CodecIOBase
codecIOBase) throws    CodecException, IOException
    public CodecIOBase decode(String uniqueCodecIdentifier,CodecIOBase
codecIOBase) throws CodecException, IOException
    private  void loadCodecs() throws IOException, CodecException;
    public String getCodecName(String uniqueCodecIdentifier) throws
CodecException;
    public String getCodecDescription(String uniqueCodecIdentifier) throws
CodecException;
    public String getCodecVersion(String uniqueCodecIdentifier) throws
CodecException;
    public String reloadCodec(String uniqueCodecIdentifier) throws
CodecException;
    public boolean isEnabled(String uniqueCodecIdentifier) throws
CodecException;
    public boolean setEnabled(String uniqueCodecIdentifier,Boolean throws
CodecException;

}
```

1. Upload the third-party codec to the following directory:
   `/hpe/codec/<CodecImplementor>/<deviceVendor>/<CodecName>/<CodecClassVersion>/<CodecName>.jar`
   For example
   `/hpe/codec/nke/nke-tic/1_0/nke-tic.jar`
2. After uploading the codec to this directory, the CodecContext loads it automatically.

However, the codec remains disabled till it is enabled.
Any usage call for the codec starts a check to see if the codec exists and is enabled. If any condition is false, the codec returns a message stating the codec is not available for use.

5.5 Data model and DAO

This table is created when installing the platform. The table shows the codec and its current status.

```
Create Table codec(
   Unique_codec_identifier character varying(10) not null,
   Codec_name character varying(100),
   Codec_description character varying(1000),
   Codec_version character varying (10),
   Enabled Boolean,
   String lifecycleStatus character varying (10)
)
```

5.6 UIoT Platform codec API for UI

A new controller is added to the DAV for CodecUI. All access from the UI to the codec is through the REST API exposed through this controller.

The REST controller gets the application wide CodecContext injected. The REST APIs are wrappers over the methods exposed by the CodecContext.

The following is the REST API:

```
//Returns Codec Name
@RequestMapping(value = "/3rdparty/codec/getCodecName", method =
RequestMethod.GET, produces = { MediaType.APPLICATION_JSON_VALUE })
public String getCodecName(@RequestParam("unqcodecid") String unqcodecid,
HttpServletRequest req) throws CodecException, IOException {
}

//Returns Codec Version
@RequestMapping(value= "/3rdparty/codec/getCodecVersion", method =
RequestMethod.GET, produces = { MediaType.APPLICATION_JSON_VALUE })
public String getCodecVersion(
               @RequestParam("unqcodecid") String unqcodecid,
               HttpServletRequest req) throws CodecException {
}

//Returns Codec Description
@RequestMapping(value = "/3rdparty/codec/getCodecDesc", method =
RequestMethod.GET, produces = { MediaType.APPLICATION_JSON_VALUE })
   public String getCodecDesc(@RequestParam("unqcodecid") String unqcodecid,
               HttpServletRequest req) throws CodecException {
}

//Return status of the given Codec
@RequestMapping(value = "/3rdparty/codec/getCodecStatus", method =
RequestMethod.GET, produces = { MediaType.APPLICATION_JSON_VALUE })
   public Boolean getCodecStatus(
               @RequestParam("unqcodecid") String unqcodecid,
               HttpServletRequest req) throws CodecException {
}

//Enables or disables Codec
```

```
@RequestMapping(value = "/3rdparty/codec/setCodecStatus", method =
RequestMethod.PUT, produces = { MediaType.APPLICATION_JSON_VALUE })
   public void setCodecStatus(@RequestParam("unqcodecid") String unqcodecid,
               @RequestParam("status") boolean status, HttpServletRequest req)
               throws CodecException {
}


//Load all Codec
@RequestMapping(value = "/3rdparty/codec/loadCodecs", method =
RequestMethod.GET, produces = { MediaType.APPLICATION_JSON_VALUE })
   public String loadCodecs(HttpServletRequest req)
               throws CodecException, IOException {
}


//Returns All Codecs
@RequestMapping(value = "/3rdparty/codec/getCodecIds", method =
RequestMethod.GET, produces = { MediaType.APPLICATION_JSON_VALUE })
   public String[] getCodecIds(HttpServletRequest req) {
}
```

5.7 Import codecs using DSM portal UI

You can add, enable, or disable a codec dynamically without rebooting or redeploying the DAV module. If the codec is overwritten through the DSM portal UI, reboot or redeploy the DAV application.

You can import the required codecs. Use the following procedure to import codecs.

1. Select the **Device Mgmt.** > **Import Codec** menu from the left pane.
2. The **Import Codec** window appears.

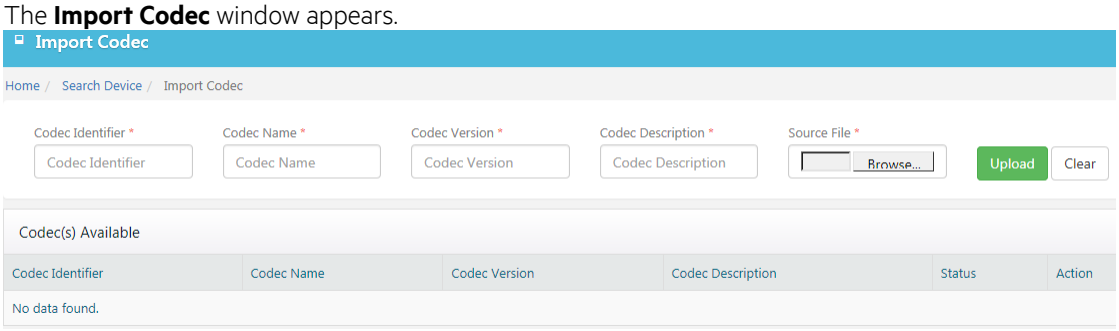


Figure 5 Import Codec

By default, the codec status is disabled. Use the Action button to toggle between enable and disable actions.

3. Enter the required details.

Table 7 Import codecs

Field	Description
Codec Identifier	Enter a unique ID for the codec. This ID is used to identify the particular implementation of encoding and decoding.
Codec Name	Enter a name for the codec.
Codec Version	Enter the version of the codec.
Codec Description	Enter a description for the codec.
Source File	Click the **Browse** button to browse and select the file for uploading. The UIoT Platform DSM portal supports JAR file format.

4. Click the **Upload** button.
   After the codec JAR file is uploaded, the file is added to the `/hpe/codec/` directory and the codec details are stored in the `Codec` table in the UIoT platform database.

5.8 Codec server framework and contract deployment

The UIoT platform codec server framework is packaged as `dav.war`. All source code is available under `dav`.

The UIoT platform codec interface is packaged as `IOTCodecContract.jar`. This is a dependency `dav` and the third-party codecs.

5.9 Encoding and decoding

Decoding

The codec library should accept the clear text string, process the same according to the device-specific decoding. Return the key value pair of decoded data.

For example:

`Input`:

```
a95ffj7493qutfee
```

`Output`:

```
Temperature: 20
Latitude: 40.12
Longitude: 2.54
```

Encoding

The codec library should accept the key value pair of the data, process the same according to the device specific encoding, and return the encoded string data.

For example:

`Input`:

```
Temperature: 20
Latitude: 40.12
Longitude: 2.54
```

`Output`:

```
a95ffj7493qutfee
```

Chapter 6
Integrate external key store

Currently, the UIoT platform has the internal key store, where the keys (AppKey/AppSessionKey) are stored corresponding to the devices with version ID. The UIoT platform supports integration of external key store.

6.1 Current—KeyStore table

Keys (LoRaWAN specific) are uploaded via the UIoT DSM portal for a device ID and stored in the keystore table with the keywrap (using the master key).

To integrate an external keystore, get the App key or AppSession key based on the parameters such as AppNonce, DevNonce, and NetId. These parameters are specific to LoRa network.

Based on the join request/accept, get the AppKey/AppSession key from the external key store and store it in the internal key store.

Table 8 KeyStore table

logical_name	device_id	keyset	key_value	key_ver	key_algorithm	masterkey_ver
AppKey	1	2	B087CF619BBF6BE5090F906809D46ADBB087CF619BBF6BE5090F906809D46ADB8C512E7A784C5D656426438A691A3052	1	1	5
AppKey	1	2	B087CF619BBF6BE5090F906809D46ADBB087CF619BBF6BE5090F906809D46ADB8C512E7A784C5D656426438A691A3052	2	1	5
AppSessionKey	1	2	5791A05832E0AF48B251BFB62845DD08A80B1D5E904D7ECF04DB62D22ED9834A8C512E7A784C5D656426438A691A3052	1	1	5
AppSessionKey	1	2	5791A05832E0AF48B251BFB62845DD08A80B1D5E904D7ECF04DB62D22ED9834A8C512E7A784C5D656426438A691A3052	2	1	5

6.2 Map new KeyStore with parameters

The parameters for the AppKey/AppSession key are maintained in a property file.

For example, `Iot.keystore.parameter.list=AppNonce,DevNonce,NetId,Xyz`

The mapping of the parameters should be similar to the following:

Table 9 KeyStore parameter mapping

Device Id	ParamKey	ParamValue	KeyVersion

1	AppNonce	1Aeiu50=	1
1	DevNonce	1Hfk#dow=	1
1	NetId	1DFJduen=	1
1	Xyz	1Sldkfj	1
1	logical_name	AppKey	1
1	AppNonce	2Aeiu50=	2
1	DevNonce	2Hfk#dow=	2
1	NetId	2DFJduen=	2
1	Xyz	1Dkljfh	2
1	Logical_name	AppKey	2
1	AppNonce	1Aeiu50=	1
1	DevNonce	1Hfk#dow=	1
1	NetId	1DFJduen=	1
1	Xyz	1Sldkfj	1
1	logical_name	AppSessionKey	1
1	AppNonce	2Aeiu50=	2
1	DevNonce	2Hfk#dow=	2
1	NetId	2DFJduen=	2
1	Xyz	Dkljfh	2
1	Logical_name	AppSessionKey	2

6.3 Keystore integration workflow

The keystore integration workflow is similar to the following:

- Join Request/Accept
- Uplink Message
- Downlink Message

6.3.1 Join request/accept

1. DAV (NIP) receives the join request/accept messages.
2. Retrieve the device ID based on the AppEUI and DevEUI.
3. Get the keyStore parameter keys from the `Iot.keystore.parameter.list` property file.
   (This file is cached)
   - AppNonce,
   - DevNonce,
   - NetId,
   - Xyz
4. Get the values for the keys from the message.
5. Get the AppSession key from external KeyStore based on these keyStore parameters.
   - UIoT platform has an adaptor-based approach to retrieve the key from the external source.
   - Adaptor implements the interface defined, so that to integrate a new key store, develop an adaptor and the configuration changes come from the UIoT platform.

   The adaptor is responsible to connect the external key store and fetch the key based on the parameters provided via the interface. The connection to the external key store is specific to the adaptor (for KMS, it is REST based API calls).
   The package for the external key store adaptor can be picked from a property file.
6. Store the AppKey/AppSession key (withKeyWrap) for the device ID and get the key version for the same from internal keystore.
7. Add entries to `KeyStore_Param_Mapping` table with the device ID, parameters, and version.
8. Store the updated message.

9. Select the routing rule and forward the message to the application.

6.3.2 Uplink Message

1. DAV (NIP) receives the uplink message.
2. Retrieve the device ID based on the AppEUI and DevEUI.

```
// device_id=from asset_params_view appeui= , deveui=
```

3. Get the latest key for the given device ID from the internal keystore.

```
//Only if asset_parameters, for the device_id, if appKey=true, then do
decrypt ..
```

4. Get unmask key from the module passing AppNonce,
   - DevNonce,
   - NetId,
   - Lora-dev-adr

```
//Unmask the key based on master key
```

5. Decrypt the payload in the message based on the unmasked key.
6. Append the clear text (decrypted) message to the message
7. Add the LORA-FRMPayload clear text to the message
8. Decode the clear text, based on the decode methodology for the device (based on device profile)
9. Select from asset_parameters for the `device_id` if `key="DecodingEnabled"` and `value=true`,
   If the value is `true`, it decodes and appends it to the JSON as captured object.
10. Select the `device_pfofile_id` from asset where `resource_id=device_id`.
11. Select `messageformat device_profile_parameters_view` where `resource_id=device_profile_id`
    `messageformat` is the `codeuniqueid`.
12. Store the updated message.

```
//Store in containerInstance table
```

13. Select the routing rule and forward the message to the application.
14. Call the `async http get` from existing code.

6.3.3 Downlink Message

1. DAV (NIP) receives the downlink message.
2. Retrieve the device ID based on the AppEUI and DevEUI.

```
// Get the parent_id from the contanerInstance which is device_id
```

3. Check if decrypt is enabled.
4. Get the key from `asset_nounce_parameters` where `device_id=`, and get the `key_value`.
5. Get the latest key for the given device ID from the internal keystore.
6. Unmask the key based on master key.
7. Encrypt the clear text payload in the message based on the unmasked key.
8. Append the encrypted payload to the message.
9. Store the updated message. `//Update the row`
10. Push the stored message to the device.

Chapter 7
Certify the devices

After onboarding the devices to the UIoT platform, you should verify the devices to see whether they are compliant to the UIoT platform and the other management applications running on it.

The checklists and guidelines for device verification and certification are available. For more details, contact the HPE support and integration team.

Appendix A
Key store interface

The external key store integration adaptor should implement the following interface.

```
package com.hpe.iot.keyStore;

import java.io.Serializable;
import java.util.Map;


public interface KeyStoreIntf extends Serializable {

   /**
    * Returns key corresponding to the key and values provided in the parameter
map.
    * Return null in case no key is available for the provided parameter map.
    *
    * @param keyMap
    * @return key as a string.
    */
   String getKey(Map<String,String> keyMap);


   /**
    * Returns true if key is available for the provided parameter map.
    * Returns false if key is not available for the provided parameter map.
    *
    * @param keyMap
    * @return true/false
    */
   boolean isKeyAvailable(Map<String,String> keyMap);

}
```

Appendix B
Sample of CodecImplementation

The following is a sample CodecImplementation: package `com.hpe.iot.codec.adx.lora.version2`

```
import java.util.HashMap;
import java.util.Map;

import org.apache.log4j.Logger;
import org.json.JSONException;
import org.json.JSONObject;

import com.hpe.iot.codec.AbstractCodec;
import com.hpe.iot.codec.CodecException;
import com.hpe.iot.codec.CodecIOBase;

public class CodecImpl extends AbstractCodec {

   private byte[] mbytearray;
   private Map<String, String> codecStringInputOutput;
   private JSONObject jsonObject;
   private String jsonString;
   final static Logger logger = Logger.getLogger(CodecImpl.class);

   public CodecIOBase encode(CodecIOBase codecIOBase) throws CodecException {

    codecIOBase.getCodecStringInputOutput();
    CodecIOBase codecIOBase1 = new CodecIOBase();
    codecIOBase1.setCodecStringInputOutput(codecStringInputOutput);

    return codecIOBase;
   }

   public String getUniqueCodecIdentifier() throws CodecException {
         String uid = "2";//This should be InO_ILD
         return uid;
   }

   public String getCodecName() throws CodecException {
         return "Adeunis codec ";
   }

   public String getCodecDescription() throws CodecException {

         return "Adeunis uplink 12 bytes downlink 8 bytes";
   }

   public String getCodecVersion() throws CodecException {

         return "0.1";
   }

   public CodecIOBase decode(CodecIOBase codecIOBase) throws CodecException {
    logger.info("Entered third party decoding Module");
    Map<String, String> Payload = codecIOBase.getCodecStringInputOutput();
    String value = Payload.get("LORA-FRMPayloadClearText");
    // String value ="0102020302010A010203";
    int len = value.length();
    byte[] byteArray = new byte[len / 2];
    char a, b;

    // Start: Convert the Hex String into a Byte Array
    for (int i = 0; i < len; i += 2) {
```

```
        a = value.charAt(i);
        b = value.charAt(i + 1);
        byteArray[i / 2] = (byte) ((Character.digit(a, 16) << 4) + Character
                        .digit(b, 16));

      }
      // End: Convert the Hex String into a Byte Array
      mbytearray = byteArray;

      try {
            jsonObject = CaptureData();
            logger.info("Json object is" + jsonObject);

      } catch (JSONException e) {
            // TODO Auto-generated caCapturedObjectstch block
            e.printStackTrace();
      }
      logger.info("Converting the JsonObject to String");
      jsonString = jsonObject.toString();
      codecStringInputOutput = new HashMap<String, String>();
      codecStringInputOutput.put("CapturedObjects", jsonString);
      codecIOBase.setCodecStringInputOutput(codecStringInputOutput);
      return codecIOBase;
}

public boolean setUniqueCodecIdentifier() {

      return true;

}

public JSONObject CaptureData() throws JSONException {

      int type = mbytearray[0];
      switch (type) {

      case 1:

            JSONObject sensorinfo = new JSONObject();
            sensorinfo.put("Data", "sensor");
            int framecounter = mbytearray[1] >> 4;

            sensorinfo.put("Framecounter", framecounter);
            sensorinfo.put("SensorType1", mbytearray[2]);

            int res = ((mbytearray[3] & 0xff) << 16)
                  | ((mbytearray[4] & 0xff) << 8) | ((mbytearray[5] & 0xff));

            sensorinfo.put("SensorType1-Value", res);

            sensorinfo.put("SensorType2", mbytearray[6]);
            int res1 = ((mbytearray[7] & 0xff) << 16)
                  | ((mbytearray[8] & 0xff) << 8) | ((mbytearray[9] & 0xff));

            sensorinfo.put("SensorType2-Value", res1);
            return sensorinfo;

      case 2:
            System.out.print("Pulse Counter Index");
            // Decoding for Pulse Counter

            break;

      case 3:
            System.out.print("AMR configuration");
```

```
                break;

        case 4:

                System.out.print("pulse counter nÃ‚Â° 1 configuration");
                break;

        default:
                System.out.print("Invalid Type");
                break;

        }
        return null;
    }

}
```

Appendix C
Sample of CodecIOBase input/output

The following is a CodecIOBase input/output sample for encode. This sample considers that the input and output are clear text. The decoded values are JSON strings.

Table 10 CodecIOBase input/output sample

UniqueCodevIndetifier (String)	Input Key(String)	Input Value(String)	Position	Output key(String)	Output value(String)	Position

InO_ILD	LORA-FRMPayload ClearText	7ef700f35426b3110a00520000410c000000000000000000000000	1st record in the map	CapturedObjects	{
					BinaryInputEndpoint0: 12
					}
						1st record in the map

				ZCLReportAttributes	{
					Endpoint: 17,
					CommandId: 10,
					ClusterId: 82,
					AttributeId: 0,
					AttributeType: 65,
					AttributeValue: 12
					}
						2nd record in the map