HPE Universal Internet of Things Platform

Onboarding Protocols User Guide
Release 1.2.1

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 developing and deploying new device controllers based on ActiveMQ DC SDK, It also provides prerequisites, Device profile design, implementation/build, deployment, basic testing, and HA (high availability) configuration and support.

The document also details the functions in which the UIoT Platform NIP and DC (device controller) are involved, such as device authentication, communication retry, and load-balancing between DC and NIP, load balancing between DC and devices, handling message burst, device sessions, session affinity, and so on.

HPE UIoT Platform supports interworking proxies (in conformance with the oneM2M CSE architecture), based on the ActiveMQ messaging broker.

Audience

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

HPE UIoT 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 the UIoT Platform DSM GUI.
Installation and Configuration Guide	Describes the steps to install and configure the HPE UIoT Platform.
External Interfaces Guide	Describes the interface definition between an external client application and the HPE UIoT Platform.
Feature Descriptions	Describes the features of the HPE UIoT Platform.
Solution Description	Describes the HPE UIoT Platform, its components, modules and features.
High Level Design	Explains the high level design of the HPE UIoT Platform.
Performance and Sizing Guide	Provides guidelines to extract high performance from the UIoT Platform.
Companion Product Descriptions	Describes the HPE products that are packaged with the HPE UIoT Platform
Solution Brief	Briefly introduces the HPE UIoT Platform and its components.
Deployment High Level Design	Explains the deployment architecture.
Sizing Guide	Provides guidelines for sizing virtual machines.
Onboarding a new device	Provides instructions to onboard a new device into the HPE UIoT Platform.
Onboarding a new protocol	Provides instructions to onboard a new protocol into the HPE UIoT Platform.

Onboarding a new tenant/application	Provides instructions to onboard a new tenant into the HPE UIoT Platform.
Onboarding a new application	Provides instructions to onboard a new application into the HPE UIoT Platform.
Reporting platform traffic	Describes the platform traffic report of the HPE UIoT Platform.
Security Guide	Describes the security features implemented in the HPE UIoT Platform.

Document history

Table 1: Document history

Edition	Product version	Date	Description
1.0	HPE UIoT Platform 1.2	June 2016	First version
2.0	HPE UIoT Platform 1.2.1	July 2016	Added Appendix A and Appendix B

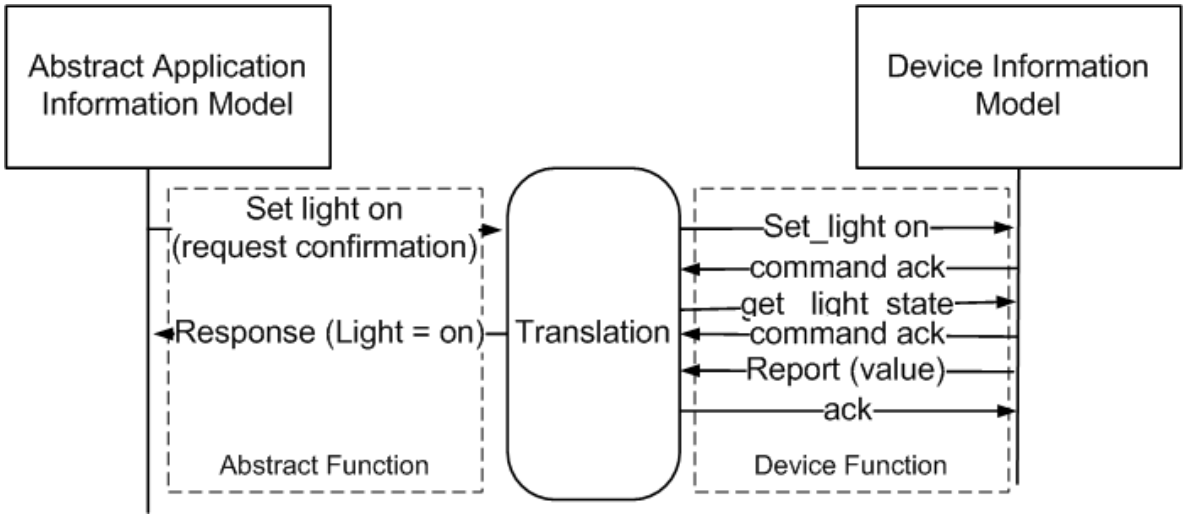
Chapter 1
Overview

UIoT Platform allows a southbound SDK style interface to devices, which does not support OneM2M devices and services.

In this case, OneM2M resources and operations are mapped to protocol specific resources or objects and operations using an adapter layer known as Interworking Proxy Entity (IPE) in OneM2M terminology. This IPE is also known as a device controller in HPE UIoT Platform architecture.

Protocol specific device controller resources and operations are mapped to UIoT Platform SDK operations covering all the life cycles of devices and services.

HPE UIoT Platform supports NIP device controller SDK, which includes a sample device controller code.



1.1 Design considerations

Two major design considerations impact the way device controllers are implemented.

1. How should UIoT Platform resources be mapped to devices and their data elements?
2. How should identifiers of devices be mapped to UIoT Platform resource model? This involves constructing UIoT Platform URIs of the resources dynamically (just like REST-based resources)

1.2 High level steps for implementing and deploying a device controller

1. Ensure prerequisites for deploying a device controller are in place.
2. Make a clone of Device Controller SDK
3. Implement the protocol specific device controller and map the device and device data to the UIoT Platform data model:
4. Implement device controller to NIP message flows
5. Implement NIP to device controller message flows
6. Map resource IDs based on the resource model. This involves creating and parsing resource URIs dynamically from device identifiers.
7. Create a device profile and container profile on UIoT Platform using DSM.
8. Provision profiles for the device on DSM.

9. Deploy Device Controller on Wildfly 9.0 application server. The end point of the device controller should be accessible.
10. Register the device controller on UIoT Platform. This step creates the queues required to establish the channel of communication between UIoT Platform and the device controller for asynchronous communication and registers a synchronous end point for the device controller.

1.2.1 Steps to implement protocol-specific device controller

- Implement REST interfaces for synchronous communication.
- Create or convert oneM2M data models. These provide standard interface for creating or converting the oneM2M primitives `RequestPrimitive` and `ResponsePrimitive`.
- Reuse subscribe and publish adaptors for message brokers.
- Replace placeholders with protocol-specific implementation.
- Use Maven-based scripts to compile, build, and deploy device controllers on the application server. SDK sample code has mock code for sensors to immediately build, deploy, and integrate with the UIoT Platform.

The SDK is intended mainly for two scenarios:

- Requests initiated from UIoT Platform to devices and devices.
- Requests and notifications from devices to UIoT Platform

1.3 Mapping UIoT Platform resources with devices and data elements

UIoT Platform supports OneM2M resource model and roughly the following should be considered for mapping.

OneM2M Resource Name	Real time/DC resource
CSEBase	Represents UIoT Platform itself. It's the root node of all the resources and is always a constant value.
remoteCSE	Represents a gateway. CSEBase can have gateways.
Node	Represents a physical device. Node is usually used during management operations. Otherwise, AE is used to represent a device.
Application Entity (AE)	Represents an application running on a device (sensor or gateway).
container	Represents a logical grouping for device data.
container Instance	Represents an independent data unit or command that flows from a device to UIoT Platform or vice versa.
Notification	Represents a notification from a device to UIoT Platform or vice versa.
Subscription	Represents a message type to subscribe to a resource such as AE or container, in order to receive notifications.

For the complete set of supported OneM2M resources, refer to the UIoT External Interfaces Guide.

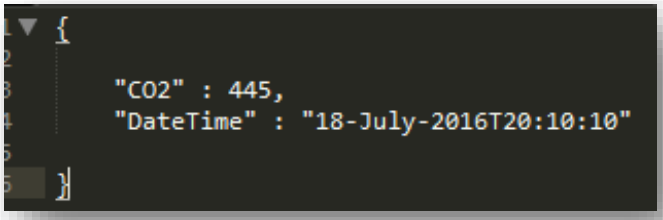
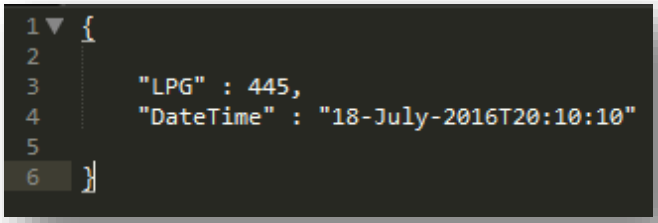
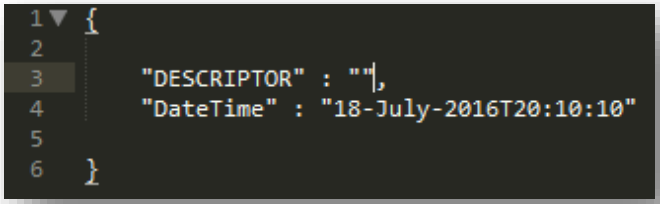
- A developer of device controllers needs to create a design of how the above resources are mapped to devices and data containers.
- Developer of device controllers may choose to model device data as separate containers of sensor data such as temperature, CO2, and humidity. Alternatively, they may choose to store all these sensor data as a single payload.
- Container structure can be defined and attached to device profiles. When you provision a device on UIoT Platform, containers for that device are automatically created.

Example

The sensor payload looks as follows:

```
{

    SensorData {
        "CO2" : 445,
        "LPG" : 200,
        "DESCRIPTOR" : "",
        "DateTime" : "18-July-2016T20:10:10"
    }
}
```

Now, you want to organize the data into three containers by exploding data based on json path

Container Name	Data
C02	
LPG	
DESCRIPTOR	

By default, the following container structure gets created automatically for every device registered on DSM.

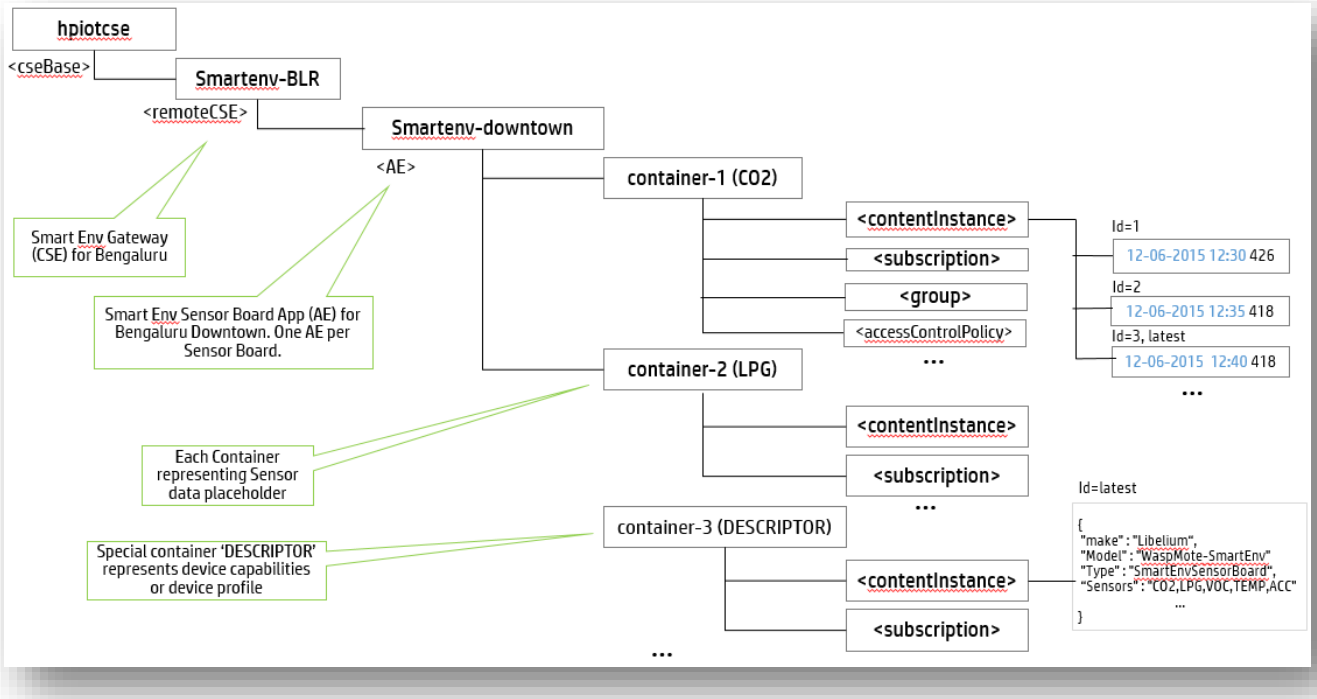
```
{

    SensorData {
        "CO2" : 445,
        "LPG" : 200,
        "DESCRIPTOR" : "",
        "DateTime" : "18-July-2016T20:10:10"
    }
}
```

Then, user needs to define a Container profile as below and provision on DSM and link it to devices from DSM portal.

NOTE: The default container stores unmodified sensor payload and is always created for a device when registering a device on DSM.

```
<?xml version="1.0" standalone="yes"?>
<ContainerProfile xmlns="http://www.hp.com/schema/m2m/">
    <ContainerProfileName>SmartEnvDevicesCP</ContainerProfileName>
    <Containers>
        <Container acp="READ_ONLY" name="CO2" parent="" sequence="1">
                            <Payload type="" value="" xPath="SensorData.LPG"/>
                <Payload type="" value="" xPath="SensorData.DateTime"/>
        </Container>
        <Container acp="READ_ONLY" name="LPG" parent="" sequence="2">
            <Payload type="" value="" xPath="SensorData.LPG"/>
            <Payload type="" value="" xPath="SensorData.DateTime"/>
        </Container>
        <Container acp="READ_ONLY" name="DESCRIPTOR" parent="default" sequence="3">
            <Payload type="" value="" xPath="SensorData.DESCRIPTOR"/>
            <Payload type="" value="" xPath="SensorData.DateTime"/>
        </Container>
    </Containers>
</ContainerProfile>
```

The following example depicts how data can be organized on smart Environment devices.



1.4 Mapping identifiers of devices

UIoT Platform uses SP Relative path to address resources such as containers and application entities. For example,

If a device controller needs to post sensor (AX123456) payload to a Co2 container, the developer should set the To and From fields as shown in the following table.

Field	Format
To	/<CSEBase>/<Assetname>/<containerName> Example: /HP_IoT/AX123456/CO2
From	<DeviceIdentifietType>:<DeviceIdentifier> Example: SerialNumber: AX123456

If a device controller needs to post sensor (AX123456) payload to a Co2 container under a gateway, the device controller developer needs to set the To and From fields on Request Primitive as shown in the following table.

Field	Format
To	/<CSEBase>/<remoteCSE>/<Assetname>/<containerName> Example: /HP_IoT/rmCSE_AX872762/AX123456/CO2
From	<DeviceIdentifietType>:<DeviceIdentifier> Example: SerialNumber: rmCSE_AX872762

The following table describes the terms in the field format.

URL Term	Description
CSEBase	Context of UIoT Platform.
Assetname	Asset/Device-ID assigned while registering on DSM.
containerName	Name of the container assigned while creating it.
GatewayName or remoteCSE	Asset/Gateway-ID assigned while registering on DSM.

Refer to the following table when creating To/From fields to address the resources on UIoT Platform.

Direction	RequestPrimitive	ResponsePrimitive	Remarks
DC ==> DAV	To : /<CSEBase>/<AssetName>/<containerName> From: imei:9988099880 Queue : <DCName>_Normal_Notf_ReqQ	To : null From: HPE_IoT Queue : <DCName>_Normal_Notf_ResQ	Next patch will have structured path in To
DC ==> DAV	To : /HP_IoT/<GatewayName>/<AssetName>/<containerName> From: imei:9988099880 Queue : <DCName>_Normal_Notf_ReqQ	To : null From: HPE_IoT Queue: <DCName>_Normal_Notf_ResQ	Gateway names always needs to be prefixed with "rmCSE_"
DAV ==> DC	To : /<AE_Resource_ID> From: /HP_IoT Queue : <DCName>_Normal_ReqQ	To : /<AE_RESOURCE_ID> From: /HP_IoT Queue : Queue : <DCName>_Normal_ResQ	Notification, Current Implementation. It's recommended to retrieve AEName (device_ID) RequestPrimitive.Content. ContentInstance.resource ID
DAV ==> DC	To : /HP_IoT/<AssetName>	To : /HP_IoT/<AssetName>/<containerName>	Notification, Will be changes to this format in next patch.

	From: /HP_IoT Queue : <DCName>_Normal_ReqQ	From: imei:9988099880 Queue : Queue : <DCName>_Normal_ResQ	

1.5 Prerequisites

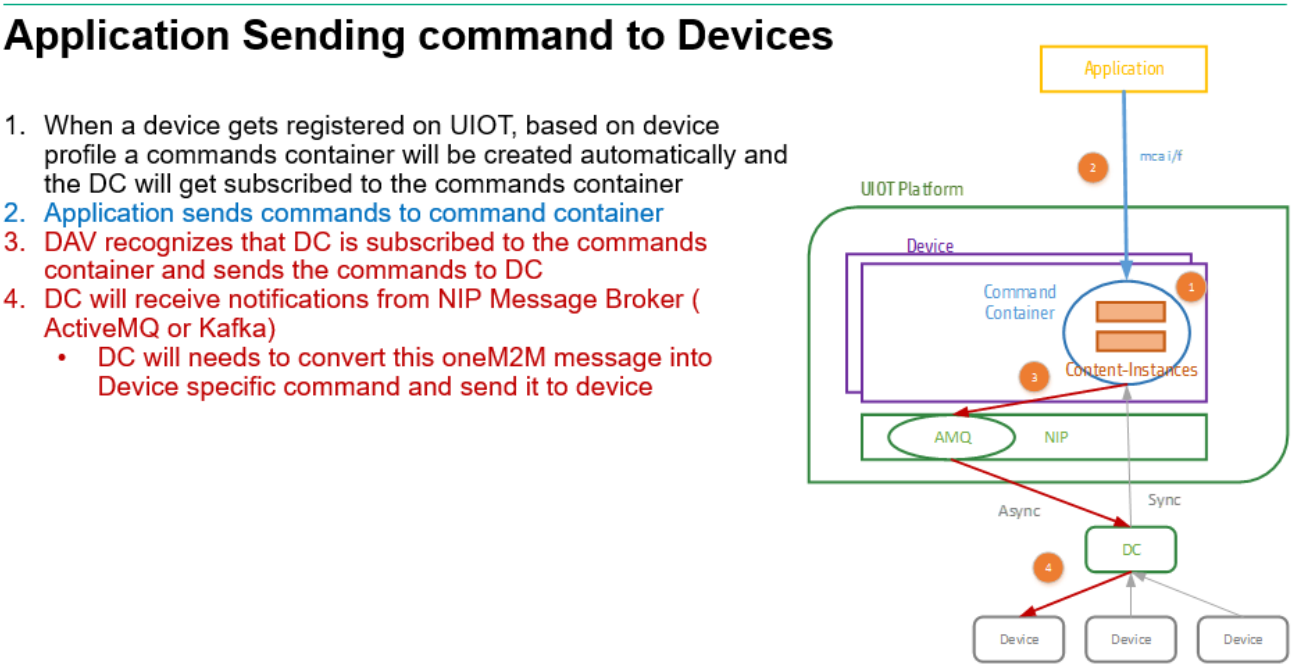
Following are the prerequisites for onboarding protocols to UIoT Platform:

- All commercial agreements between HPE and the customer using the protocol are in place.
- The IP connectivity between the UIoT Platform aaS system and the customer enterprise is already established.
- 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*.
- Development Environment
  - Desktop with at-least 4GB RAM and quad core
  - Oracle JDK 1.7, Maven 3.0+ and Wildfly 9.0 Application server
  - Eclipse IDE (preferably)
- Operational requirements for tasks such as exposing device controller over public IP, creating load balancer configurations, etc.

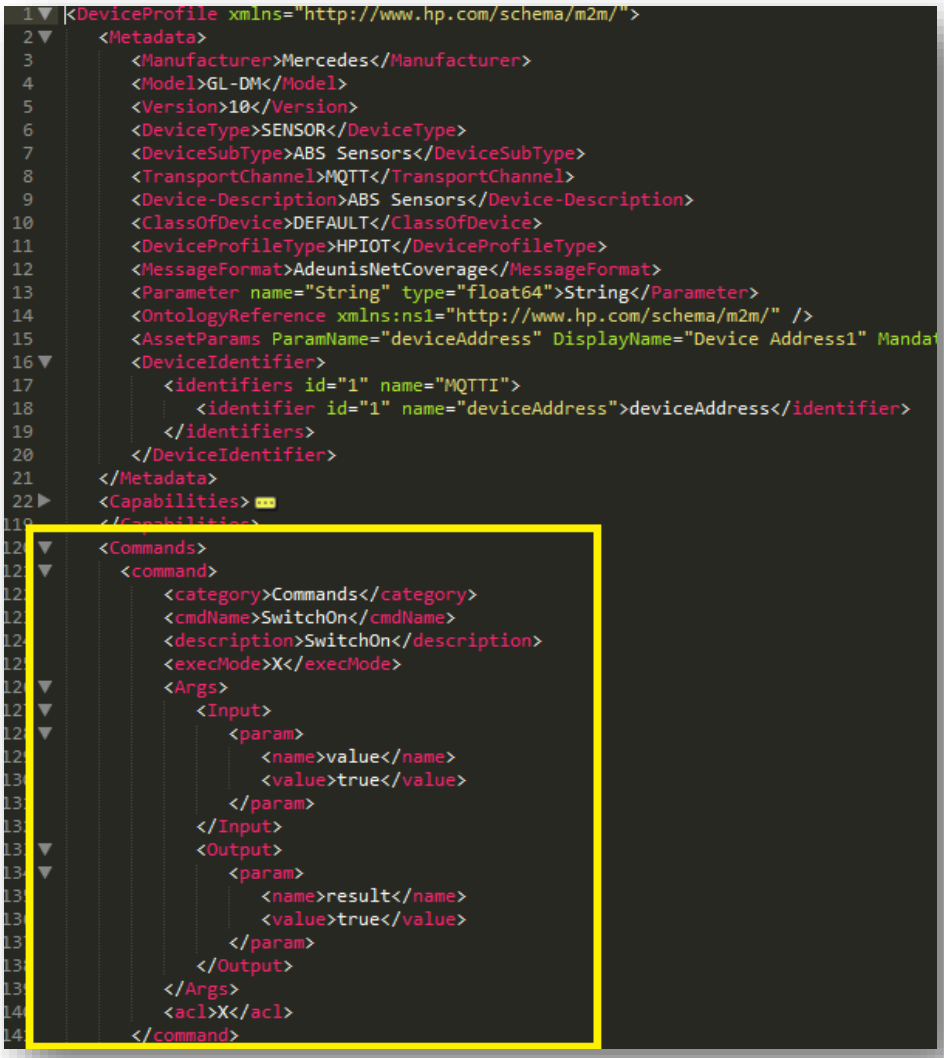
1.6 App/IOT initiated requests towards devices

These are requests from an external application or UIoT Platform initiated to devices through a device controller.

In the case of an application sending a command to a device, the application creates a RequestPrimitive object (with POST as operation type) and sends it to the "command" container hosted on UIoT platform. A device controller subscribed to the "command" container gets the command as notification and processes it. Refer to the following diagram for details.



If the device profile of the device to be provisioned has a `commands` section, then DSM creates a container with the name "command" internally and the device controller is automatically subscribed to this "command" container. If any application posts a message containing a command message to this container, the device controller will get a notification. The device controller developers need to retrieve the container-Instance containing commands from the notification object and process these commands towards devices.



```
<DeviceProfile xmlns="http://www.hp.com/schema/m2m/">
    <Metadata>
        <Manufacturer>Mercedes</Manufacturer>
        <Model>GL-DM</Model>
        <Version>10</Version>
        <DeviceType>SENSOR</DeviceType>
        <DeviceSubType>ABS Sensors</DeviceSubType>
        <TransportChannel>MQTT</TransportChannel>
        <Device-Description>ABS Sensors</Device-Description>
        <ClassOfDevice>DEFAULT</ClassOfDevice>
        <DeviceProfileType>HPIOT</DeviceProfileType>
        <MessageFormat>AdeunisNetCoverage</MessageFormat>
        <Parameter name="String" type="float64">String</Parameter>
        <OntologyReference xmlns:ns1="http://www.hp.com/schema/m2m/" />
        <AssetParams ParamName="deviceAddress" DisplayName="Device Address1" Mandat
        <DeviceIdentifier>
            <identifiers id="1" name="MQTTI">
                <identifier id="1" name="deviceAddress">deviceAddress</identifier>
            </identifiers>
        </DeviceIdentifier>
    </Metadata>
    <Capabilities> ...
    <Commands>
      <command>
          <category>Commands</category>
          <cmdName>SwitchOn</cmdName>
          <description>SwitchOn</description>
          <execMode>X</execMode>
          <Args>
              <Input>
                  <param>
                      <name>value</name>
                      <value>true</value>
                  </param>
              </Input>
              <Output>
                  <param>
                      <name>result</name>
                      <value>true</value>
                  </param>
              </Output>
          </Args>
          <acl>X</acl>
      </command>
```

The supported modes of request are Synch and Asynch. The communication of how Synch and Asynch happen between UIoT Platform and the device controller are abstracted from a device controller and handled by the device controller SDK.

- Sync—In a synchronous request, UIoT Platform receives the response immediately in the same request thread. This synchronous process is also considered as the blocking mode. The resources are blocked till the UIoT Platform gets a response.
  Synchronous communication happens from NIP to device controllers through HTTP interfaces. The device controller SDK has inbuilt HTTP interfaces for communication.
- Asynch—In an asynchronous mode of communication, UIoT Platform sends a request to the device controller. The platform does not require an immediate response from the device controller. The device controller sends a response when the devices respond. This non-blocking communication is the preferred method in UIoT Platform. Asynchronous communication happens through Messaging Broker (ActiveMQ).

The device controller SDK has inbuilt JMS implementations for sending and receiving requests from NIP messaging broker queues.

Sample Request Primitive with Notification Arriving at DC.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:requestPrimitive xmlns:ns2="http://www.onem2m.org/xml/protocols">
   <operation>5</operation>
   <to>HPE_IoT/rmCSE_GW_S2/9804060846</to>
   <from>HPE_IoT</from>
   <requestIdentifier>1320093516245414958</requestIdentifier>
   <name>NOTIF</name>
```

```
   <content>
         <Notification>
               <notificationEvent>
                     <representation xsi:type="xs:string"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">{"ContentInstance":{"resourceType":4,"resourceID":"HPE_IoT/rmCSE_GW_S2
/9804060846/commands/7d43086be9b","parentID":"HPE_IoT/rmCSE_GW_S2/9804060846/com
mands","creationTime":"2016-07-21 12:16:20.984","lastModifiedTime":"2016-07-21
12:16:20.984","labels":[],"name":"7d43086be9b","expirationTime":"2016-07-21
12:16:20.974","announceTo":[],"announcedAttribute":[],"stateTag":0,"creator":nul
l,"contentInfo":"text/plain:0","contentSize":200,"ontologyRef":null,"content":"{
\"COMMAND\":\"18, TS\":\"2015-06-27T23:12:25.770+05:30\"}"}}</representation>
                     <resourceStatus>1</resourceStatus>
                     <operationMonitor>
                           <operation>5</operation>
                           <originator>-5062010536225455657</originator>
                     </operationMonitor>
               </notificationEvent>
               <verificationRequest>false</verificationRequest>
               <subscriptionDeletion>false</subscriptionDeletion>

   <subscriptionReference>1818221799257686553</subscriptionReference>
               <creator>-5062010536225455657</creator>
         </Notification>
   </content>
   <responseType>2</responseType>
</ns2:requestPrimitive>
```

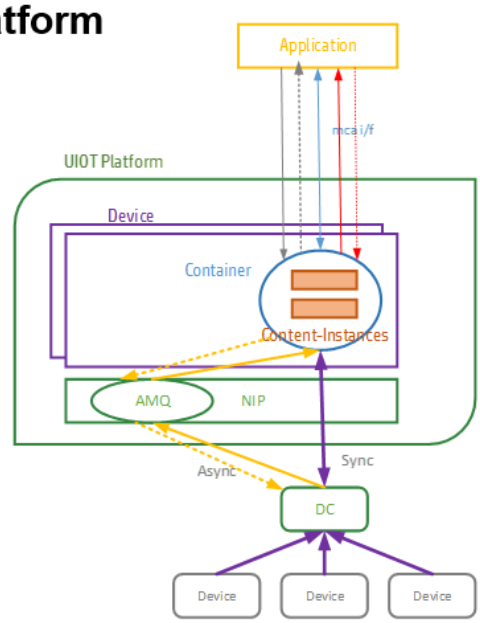
1.7 Sensor or gateway requests to UIoT Platform

These are requests initiated from connected sensors/gateways to UIoT Platform. Examples of this request are:

1. Sensor posting its data to UIoT Platform
2. Notifications from a sensor to UIoT platform

Devices Sending Sensor data to UIOT Platform

- Devices can send sensor data to UIOT Platform via DC
- DC converts non-OneM2M request to OneM2M and sends to UIOT platform for storing and sharing it with applications
- If DC sends blocking POST sensor readings request (purple arrow) to UIOT platform, UIOT gets responds back in the same connection that it has accepted the request.
- If DC sends non-blocking POST sensor readings request (orange arrow) to UIOT platform over message broker,
  - Message Broker forwards message to DAV
  - DAV Processes message and sends response back to message broker
  - Message Broker routes the message to corresponding DC
  - DC will receives this response message and processes it



This communication also happens in the synchronous and asynchronous modes.

1. Synchronous: In this mode, a device posts data to the device controller and the device controller directs it to NIP/UIoT platform. This is blocking communication till the UIoT Platform gives a response back.
2. Asynchronous: In this mode, the device posts to the device controller and the device controller acknowledges immediately. Then device controller will post this request to UIoT Platform in the same way. The responses from UIoT platform are received later through the "sendResponseToDevice" interface.

While posting data to UIoT Platform, it is the same interface "sendRequestToIOT" that is used for synchronous and asynchronous communication. The difference is in the 'responseType' attribute of the requestPrmitive. The responseType 2 is used for asynchronous commiunication and 3 for synchronous. The device controller SDK abstracts how synch and asynch are posted to NIP.

Sample Request Primitive with Device Payload towards IOT from DC

Message Sent On Queue: Acme_Normal_Notf_ReqQ

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:requestPrimitive xmlns:ns2="http://www.onem2m.org/xml/protocols">
   <operation>1</operation>
   <to>HP_IoT/9988099880/default</to>
   <from>IMEI:9988099880</from>
   <requestIdentifier>3afb7396-1562-46e1-85e4-40fbb766a3d1</requestIdentifier>
   <resourceType>4</resourceType>
   <name>9988099880</name>
   <content>
         <ns2:contentInstance>
               <contentInfo>text/plain:0</contentInfo>
               <contentSize>50</contentSize>
               <content xsi:type="xs:string"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">{"ACC":"10;11;30","BAT":"81","TEMP":"27","CO2":"411","VOC":"47","HUMA"
:"69","LUX":"103","LPG":"410","GAS_UNIT":"PPM","TS":"2014-11-
03T10:58:00.148+05:30"}</content>
         </ns2:contentInstance>
   </content>
   <responseType>2</responseType>
</ns2:requestPrimitive>
```

Sample Request Primitive with Gateway Payload towards IOT from DC

Message Sent On Queue: Acme_Normal_Notf_ReqQ

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:requestPrimitive xmlns:ns2="http://www.onem2m.org/xml/protocols">
   <operation>1</operation>
   <to>HPE_IoT/rmCSE_GW_11/990000862471860/default</to>
   <from>IMEI:990000862471860</from>
   <requestIdentifier>074e3430-4b8b-4405-a857-9e28197ff7e5</requestIdentifier>
   <resourceType>4</resourceType>
   <name>rmCSE_GW_11/990000862471860</name>
   <content>
         <ns2:contentInstance>
               <contentInfo>text/plain:0</contentInfo>
               <contentSize>50</contentSize>
               <content xsi:type="xs:string"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">{"ACC":"10;11;30","BAT":"81","TEMP":"27","CO2":"411","VOC":"47","HUMA"
:"69","LUX":"103","LPG":"410","GAS_UNIT":"PPM","TS":"2014-11-
03T10:58:00.148+05:30"}</content>
         </ns2:contentInstance>
   </content>
   <responseType>2</responseType>
```

```
   <notificationUri>rmCSE_GW_11/990000862471860</notificationUri>
</ns2:requestPrimitive>
```

1.8 Onboarding device controllers

Do the following to onboard device controllers.

1. Configure project in Maven with basic details.
   The required basic configuration is already inbuilt and hence the procedure is simple to follow.
2. Implement an ObjectMapper Interface.
3. Implement a publisher interface for sending requests to device.
4. Implement a subscriber interface for receiving request from device.
5. Build and deploy.
6. Onboard devices on the UIoT Platform.

1.9 Configure pom.xml

Build the project using Maven. The POM `groupId` and `artifactId` are the following:

```
<groupId>com.hpe</groupId>
   <artifactId>AcmeDC</artifactId>
   <name>AcmeDC</name>
   <packaging>war</packaging>
   <version>1.0</version>
```

The required Java version is 1.7 and Spring version is 4.2.3:

```
<java-version>1.7</java-version>
<org.springframework-version>4.2.3.RELEASE</org.springframework-version>
```

For a complete list of dependencies, refer to the full `pom.xml` file, which is part of the device controller SDK.

1.10 Configure web.xml

Web.xml has two listeners.

- Spring ContextLoaderListener for dispatcher servlet loading.
- Log4jConfigListener for log4j.

Do the following to configure `web.xml`:

1. Configure the path for `log4j.xml`

```
    <context-param>
        <param-name>log4jConfigLocation</param-name>
        <param-value>file:$IOT_HOME/config/{DcName}/log4j.xml</param-value>
    </context-param>
```

2. Configure the refresh interval.

```
    <context-param>
        <param-name>log4jRefreshInterval</param-name>
        <param-value>60000</param-value>
    </context-param>
```

1.11 Configure log4j.xml

Log4j is fully configurable at runtime using external the configuration file `log4j.xml`. The following minimal configuration is required for a new device controller.

Change the log file name.

```
<appender name="DC" class="org.apache.log4j.DailyRollingFileAppender">
   <param name="File" value="${jboss.server.log.dir}/AcmeDC.log"/>
   <param name="DatePattern" value="'.'yyyy-MM-dd"/>
   <param name="Append" value="true"/>
   <layout class="org.apache.log4j.PatternLayout">
   <param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%c{1}] %m%n"/>
   </layout>
</appender>
```

1.12 Configure servlet-context.xml

This file is intended for the Spring dispatcher servlet configuration. More configuration beans can be added here.

Minimum requirements are the following:

1. Scan the class path.

   ```
   <annotation-driven />
         <context:component-scan base-package="com.hpe.iot" />
   ```

   Base path scanning is enabled by default. If the package does not follow the above mentioned package structure, add the new package to be scanned.

2. Configure the property file.

   ```
   <beans:bean id="propertyConfigurer"
   class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer
   ">
         <beans:property name="ignoreUnresolvablePlaceholders" value="true" />
         <beans:property name="locations">
               <beans:list>
                     <beans:value>file:$IoT_HOME/config/{DcName}/iot-
   queue.properties</beans:value>
               </beans:list>
         </beans:property>
   </beans:bean>
   ```

By default, the `iot-queue.properties` file contains the pre-configured properties available for device controllers. You can add more properties if required.

1.13 Configure iot-queue.properties

This `iot-queue.properties` file is one of the important files for developing a new device controller. As part of onboarding a new device controller, eight new queues should be configured in UIoT Platform.

The required configurations are the following:

1. NIP hosting address.

```
iot.naming.provider.url=failover://(tcp://localhost:61616)?initialReconnectDe
lay=200
```

The details are provided by the UIoT Platform hosting system.

2. Four normal queues and four priority queues used when asynchronous communication occurs between UIoT Platform and the device controller.

   For example, if your device controller name is AcmeDc, the suggested names for queues are the following:

```
AcmeDC_Normal_ReqQ, AcmeDC_Normal_ResQ, AcmeDC_Normal_Notf_ReqQ,
AcmeDC_Normal_Notf_ResQ

AcmeDC_Normal_ReqQ, AcmeDC_Normal_ResQ, AcmeDC_Normal_Notf_ReqQ,
AcmeDC_Normal_Notf_ResQ
```

   These names are auto generated on the UIoT Platform DSM portal GUI when onboarding a device controller. However, you can edit them and use those updated names when onboarding device controller to DSM portal.

3. HTTP endpoint for NIP to communicate during synchronous operation.

```
iot.http.endpoint.uri=http://{host}:{port}/davc/rest/postRequestPrimitive
```

   Represents the DAV hosting address. This address is used during the synchronous communication between UIoT Platform and device controllers.

4. In the property file:

```
# Description      : Enable/Disable ActiveMQ.
# Property Type    : Mandatory
# Example          : true/false
iot.jms.activemq.enabled=true
```

```
# Description      : NIP Connection details.
# Property Type    : Mandatory
# Example          : tcp://<host_name>:61616 or remote://localhost:4447 or
nio://<host_name>  :61616
iot.naming.provider.url=failover://(tcp://localhost:61616)?initialReconnectDe
lay=2000
```

```
# Description      : Number of JMS consumers for each QUEUE - NORTH BOUND
# Property Type    : Mandatory
# Example          : QUEUE_NAME
iot.jms.consumer.number=50
```

```
# Description      : Queue Name for request to NIP
# Property Type    : Mandatory
# Example          : QUEUE_NAME
iot.other.request.queue= AcmeDC_Normal_Notf_ReqQ
```

```
# Description      : Queue Name for response from NIP
# Property Type    : Mandatory
# Example          : QUEUE_NAME
```

```
iot.other.response.queue= AcmeDC_Normal_Notf_ResQ
```

```
# Description      : Queue Name for request to DC
# Property Type    : Mandatory
# Example          : QUEUE_NAME
iot.this.request.queue= AcmeDC_Normal_ReqQ
```

```
# Description      : Queue Name for response from DC
# Property Type    : Mandatory
# Example          : QUEUE_NAME
iot.this.response.queue= AcmeDC_Normal_ResQ
```

```
# Description      : Priority Queue Name for request from NIP
# Property Type    : Mandatory
# Example          : QUEUE_NAME
iot.other.request.queue.priority= AcmeDC_Priority_Notf_ReqQ
```

```
# Description      : Priority Queue Name for response to NIP
# Property Type    : Mandatory
# Example          : QUEUE_NAME
iot.other.response.queue.priority= AcmeDC_Priority_Notf_ResQ
```

```
# Description      : Priority Queue Name for notification request to NIP
# Property Type    : Mandatory
# Example          : QUEUE_NAME
iot.this.request.queue.priority= AcmeDC_Priority_ReqQ
```

```
# Description      : Priority Queue Name for notification response from NIP
# Property Type    : Mandatory
# Example          : QUEUE_NAME
iot.this.response.queue.priority= AcmeDC_Priority_ResQ
```

```
# Description      : DAVC End point for NIP to process sync request
# Property Type    : Mandatory
# Example          : localhost:8080/davc
iot.http.endpoint.uri=http://{host}:{port}/davc/rest/postRequestPrimitive
```

```
# Description      : Enable/Disable use of shortNamesin oneM2M primitives
# Property Type    : Mandatory
# Example          : true/false
iot.onem2m.shortnames.enabled=true
```

```
# Description      : Flag for proxy to understand it is DC or DAV
```

```
# Property Type     : Mandatory
# Example           : true/false
iot.proxy.is.HttpDc=true
```

1.14 Additional libraries

All required libraries are available on Maven, except the following. The UIoT Platform support team will provide the latest versions of these libraries.

```
<dependency>
   <groupId>m2m-common</groupId>
   <artifactId>m2m-common</artifactId>
   <version>1.0</version>
</dependency>

<dependency>
    <groupId>com.hpe</groupId>
    <artifactId>iot-proxy</artifactId>
    <version>1.0</version>
</dependency>
```

Add these libraries to the local repository before building the deployable files:

```
mvn install:install-file -Dfile=iot-proxy-1.0.jar -DgroupId=com.hpe -
DartifactId=iot-proxy -Dversion=1.0 -Dpackaging=jar
```

1.15 High level architecture

Asynchronous communication occurs between device controller and NIP through JMS over ActiveMQ. The device controller SDK has inbuilt plug-in for listening and sending requests to the corresponding queues. All complications are abstracted from a device controller developer.

Synchronous communication takes place between device controller and UIoT Platform over HTTP. Like in the asynchronous communication, the device controller SDK has inbuilt server and client side implementation for HTTP communication too. The developer should implement the south bound interface and apply the conversions to oneM2M primitives.

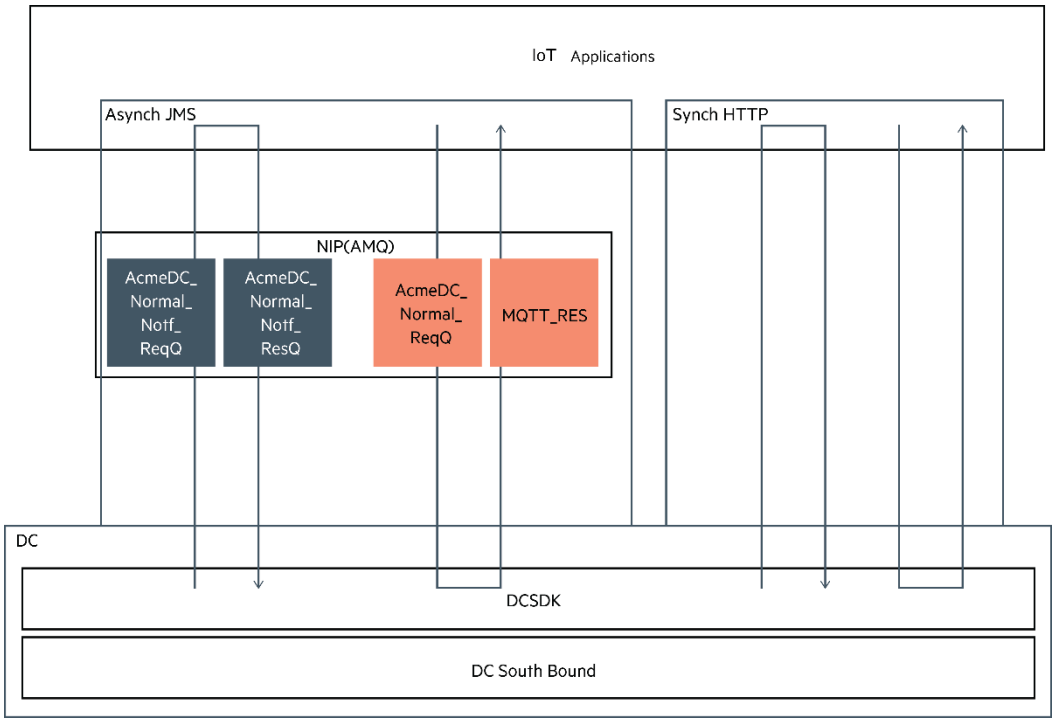


Figure 1 High-level architecture DC SDK and integration with NIP

1.16 Object converter

The first step in the device controller protocol implementation is creating a mapping between the protocol-specific request or response object and the oneM2M requestPrimitive/responsePrimitive parameters.

All requests between UIoT Platform and the device controllers are sent through standard oneM2M requestPrimitives and responsePrimitives. The requestPrimitive conforms to the oneM2M standard and holds all the necessary information about the request. For every request, a similarly defined response is generated and sent through the responsePrimitive with the same request identifier.

1.17 Request Primitive

The following diagram shows the structure of Request primitive.

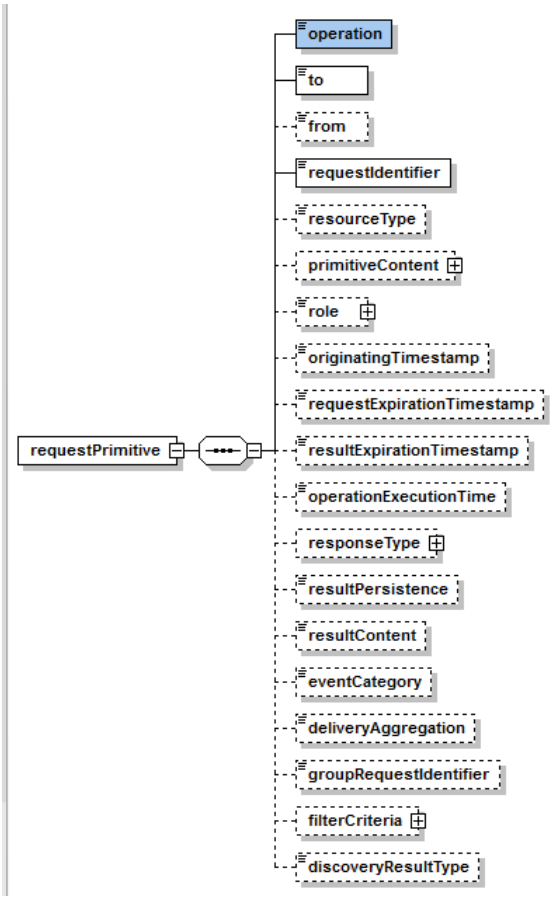


Figure 2: Request primitive structure

The mandatory fields of request primitive are defined with examples in the following sections.

The following table describes fields mandatory for the request primitive:

Table 2 Parameters for request primitive

Parameter	Description
OPERATION Type	The following operations are possible: • 1(create)

<table>
<tr><td></td><td>

2(retrieve)
3(update)
4(delete)
5 (Notify)

Example: Device controller developer needs to send a RequestPrimitive with operation value = 1, for posting sensor data to create a content instance on UIoT Platform.
</td></tr>
<tr><td>To</td><td>
The targeted device identifier.

1. To. This is the targeted device/Gateway Identifier.

The To field format follows the structured path as defined by oneM2M spec.

Generic format is as follows:

<table>
<tr><th>Resource</th><th></th></tr>
<tr><td>Direct sensors</td><td>/<IoTPlatformName>/<DeviceName>/<ContainerName></td></tr>
<tr><td>Sensors under GW</td><td>/< IoTPlatformName>/<GWName><DeviceName>/<ContainerName></td></tr>
<tr><td>Sensors with nested containers</td><td>/<IoTPlatformName>/<DeviceName>/<ParentContainerName>/<ChildContainerName></td></tr>
<tr><td></td><td></td></tr>
</table>

IoTPlatformName -> IoT Platform Name
DeviceName -> Name of the Device defined during Asset registration in DSM
GWName -> Name of the Gatewy defined during GW registration in DSM. In 1.2GA Its required to prefix string rmCSE_ with GW name while contstructing URL for device under GW.
ContainerName -> Name of the container

Note : DC Developer is expected to know the containers mapped to device
</td></tr>
<tr><td>From</td><td>
Name of the device.

The name can be a unique <code>key:value</code> for each device or the registered ID in UIoT Platform.

Generic format is as follows:

<code>DeviceIdentifierName:DeviceIdentifierValue</code>

DeviceIdentifierName -> Device Identifier name defined in DP
DeviceIdentifierValue -> Device Identifier value defined during Asset registration in DSM.

Example,

<code>MACADDRESS:11.22.33.44</code>
<code>POA:gw1/light123 or 234234234.</code>

</td></tr>
<tr><td>Request Identifier</td><td>A unique Id for each request on UIoT Platform. The response to the request will have the same identifier as the request. The recommended way is to create a UUID and pass it to be sure that it's a unique identifier for each request.</td></tr>
<tr><td>Resource Type</td><td>Shows the type of operation.</td></tr>
</table>

	Example: • 4 for Content Instance • 12 for management command
Content	OneM2M content objects. The `Content` of the message is the part where the sensor payload resides. This content should be available in JSON format.

```
RequestPrimitive requestPrimitive = new RequestPrimitive();

RequestPrimitive requestPrimitive = new RequestPrimitive();

         //Create(1), Retrieve(2), Update(3), Delete(4), Notify (5);
         requestPrimitive.setOperation(BigInteger.valueOf(1l));

         //Targeted Relative path of the device
//Egs: /HPE_IoT/990000862471854/default
requestPrimitive.setTo("/HPE IoT/990000862471854/default");

         //Device/Gateway/resource/application Id
         requestPrimitive.setFrom("IMEI:990000862471854");

         //Unique ID for each request
         requestPrimitive.setRequestIdentifier(UUID.randomUUID().toString());

         //Asynch(2), Synch(3)
         requestPrimitive.setResponseType(BigInteger.valueOf(1l));

         //If ResponseType==2,Notification URL for status update.Keep null if NA
         requestPrimitive.setNotificationUri(null);

         ////Device/Gateway/resource/application name
         requestPrimitive.setName("990000862471854");

         //oneM2M resource Type. ContentInstance(4), mgmt command(12) etc.
         requestPrimitive.setResourceType(BigInteger.valueOf(4l));

         //Real Content as per ResourceType
         requestPrimitive.setContent(contentInstance);
```

The following table describes fields mandatory for response primitive.

Table 3 Parameters for response primitive

Parameter	Description
Request Identifier	The request identifier of the response should be the same as that of the corresponding request.
Response Status code	Status of the request.
	Example: • Success (2000) • Accepted (2002) For a complete list of status codes, refer to the oneM2M *TS-0004_Service_Layer_Core_Protocol_Specification-V_1_0_1* document.

```
ResponsePrimitive responsePrimitive = new ResponsePrimitive();

   //Response status code as per oneM2M.
   //Refer 6.6.3 of TS-0004_Service_Layer_Core_Protocol_Specification-V_1_0_1
   responsePrimitive.setResponseStatusCode(BigInteger.valueOf(1000));

   //Request unique Identifier
   responsePrimitive.setRequestIdentifier(value);

   //Device/Gateway/resource/application name
   responsePrimitive.setTo(value);

   //Device/Gateway/resource/application name
responsePrimitive.setFrom(value);
```

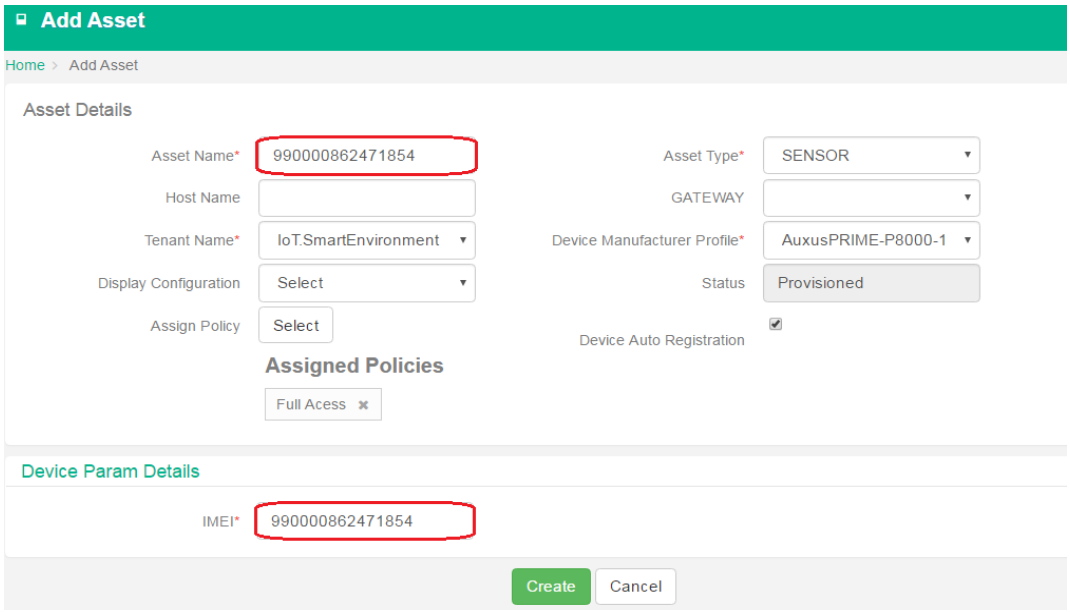
1.18 Steps to construct To/From URL

1. Device manufacturer profile is the XML, which defines device capabilities and also defines the identifier of the device as highlighted in the following diagram. This needs to be uploaded to DSM portal and attached to the devices while provisioning them on UIoT Platform.

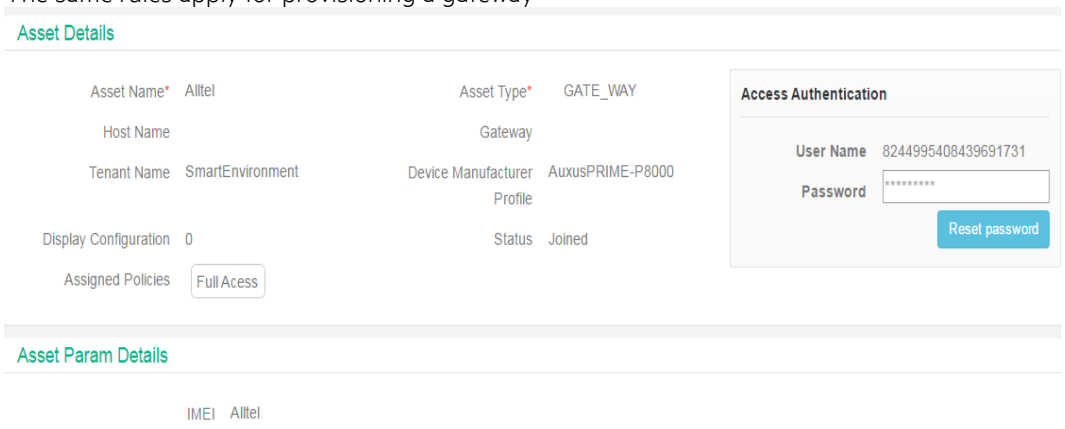
```
<DeviceProfile xmlns="http://www.hp.com/schema/m2m/">
   <Metadata>
      <Manufacturer>AuxusPRIME</Manufacturer>
      <Model>P8000</Model>
      <Version>10</Version>
      <DeviceType>SENSOR</DeviceType>
      <DeviceSubType>ABS Sensors</DeviceSubType>
      <TransportChannel>MQTT</TransportChannel>
      <Device-Description>ABS Sensors</Device-Description>
      <ClassOfDevice>DEFAULT</ClassOfDevice>
      <DeviceProfileType>HPIOT</DeviceProfileType>
      <MessageFormat>AdeunisNetCoverage</MessageFormat>
      <Parameter name="String" type="float64">String</Parameter>
      <OntologyReference xmlns:ns1="http://www.hp.com/schema/m2m/" />
      <AssetParams ParamName="IMEI" DisplayName="IMEI" Mandatory="true" ReadOnly="false" DataType="String" Category="IoT" Decoding="none" DefaultValue="" />
      <DeviceIdentifier>
         <identifiers id="1" name="MQTTI">
            <identifier id="1" name="IMEI">IMEI</identifier>
         </identifiers>
      </DeviceIdentifier>
   </Metadata>
         .
         .
         .
```

The device Identifier section in this DP sample defines the identifier expected from the device. It's referred as Asset Parameter in the platform.

2. While provisioning a device on UIoT platform, make sure that Asset Name and Device ID (it can be IMEI, Serial number, etc as defined in Device Profile) should be the identifier of the device as highlighted by red boxes in the following diagram.



The same rules apply for provisioning a gateway



NOTE: The user must enter the same value in the Asset Name and Device Identifier fields for 1.2 GA release.

To URL:

a) Readings post to Default Container :
/HPE_IoT/990000862471854/default

HPE_IoT -> IoT Platform Name (Constant)
990000862471854 -> Device Name
default-> Default Container Name (Constant)

b) Readings post to sepcific container :
/HPE_IoT/990000862471854/Temperature

Temperature -> Device controller is expected to know the specific containers mapped to the device.

c) Readings post to nested containers :
HPE_IoT/990000862471854/default/location

location -> Device controller is expected to know the specific containers mapped to the device.

d) Device under GW
HPE_IoT/rmCSE_Alltel/990000862471854/default

rmCSE_Alltel -> Alltel is the name of the gateway registered in DSM. In 1.2GA, it is required to prefix the string rmCSE_ to the gateway name while constructing the URL for the device under the gateway.

From URL:

IMEI:990000862471854

IMEI -> Device Identifier name defined in DP

990000862471854 -> Device Identifier value defined during Asset Registration in DSM.

1.19 Constructing Response Primitive

The mandatory fields of request primitive are:

- **Request Identifier**: The request identifier of the response should be the same as that of the corresponding request.
- **Response Status code**: Status of the request.

  Ex: Success (2000), Accepted (2002).

  Refer to the document, *TS-0004_Service_Layer_Core_Protocol_Specification-V_1_0_1*, for the complete list of status codes.

```
ResponsePrimitive responsePrimitive = new ResponsePrimitive();

            //Response status code as per oneM2M.
            //Refer 6.6.3 of TS-
0004_Service_Layer_Core_Protocol_Specification-V_1_0_1

      responsePrimitive.setResponseStatusCode(BigInteger.valueOf(1000));

            //Request unique Identifier
            responsePrimitive.setRequestIdentifier(value);

            //Device/Gateway/resource/application name
            responsePrimitive.setTo(value);

            //Device/Gateway/resource/application name
responsePrimitive.setFrom(value);
```

1.20 Implement convert method

Design your model objects and implement the convert methods.

1. Create model objects
   Create the protocol-specific model object under the `com.hpe.iot.dc.south.model` package.
2. Implement the interface ObjectConverter.

```
public class ConverterImpl implements ObjectConverter<AcmeRequest,
AcmeResponse>
{
      @Override
      public RequestPrimitive from(AcmeRequest requestobject) {
            // TODO Code to convert Protocol object to RequestPrimitive
            // TODO Return the converted object
            return null;
}

      @Override
      public AcmeRequest from(RequestPrimitive requestPrimitive) {
            // TODO Code to convert Request primitive to protocol specific
object
// TODO Return the converted object
            return null;
      }

      @Override
      public ResponsePrimitive to(AcmeResponse response) {
```

```
            // TODO Code to convert to Protocol response to ResponsePrimitive
            // TODO Return the converted object
            return null;
      }

      @Override
      public AcmeResponse to(ResponsePrimitive responsePrimitive) {
            // TODO Code to convert Response primitive to protocol specific
response.
// TODO Return the converted object
            return null;
      }


}
```

3. Define the converter class as a Spring bean with `@Service` annotation

```
@Service
public class ConverterImpl implements ObjectConverter< AcmeRequest,
AcmeResponse > {
…
…

}
```

1.21 Implement device controller publisher

This publisher is a protocol specific client/publisher to send requests to devices. Import your required libraries and implement the `DcPublisher` interface.

The following is an example of the implementation.

```
@Service
public class MqttPublisher implements DcPublisher< AcmeRequest, AcmeResponse >{


   @Override
   public AcmeResponse sendRequest(AcmeRequest request) {
         // TODO Protocol Specific code here for posting the request
         // TODO Return response after the caller
         return null;
   }


   @Override
   public void sendResponse(AcmeResponse response) {
         // TODO Protocol Specific code here for posting the response

   }

}
```

Here, `AcmeRequest` and `AcmeResponse` are your custom protocol specific request/response model objects. This class also is made to be a spring bean.

1.22 Implement device controller subscriber

This subscriber is the protocol-specific server/subscriber to receive requests from devices or gateways. Import your required libraries and implement the interface `DcSubscriber`.

The following is an example of the implementation.

```
@Service
public class MqttSubscriber implements DcSubscriber< AcmeRequest, AcmeResponse
>{


   @Override
   public MqttResponse recieveRequest(AcmeRequest request) {
         // TODO Protocol Specific code here for receiving the request
         // TODO Return response after the caller

         return null;
   }


   @Override
   public void receiveResponse(AcmeResponse response) {
         // TODO Protocol Specific code here for receiving response.

   }

}
```

Here, `AcmeRequest` and `AcmeResponse` are your custom protocol specific request/response model objects. This class is also defined as a Spring bean.

1.23 Build and deploy

1. Build the project using Maven.

   ```
   mvn clean install
   ```

   This command generates an `acmeDc.war` file in the `Target` folder.

2. Deploy the `WAR` file.
   Copy it to the JBoss deployment folder `$IOT_HOME/jboss/standalone/deployments`

1.24 Onboard and enable

To onboard the new device controller, register it on UIoT Platform using the DSM portal.

1. Log into the UIoT Platform DSM portal.

2. Click the **DC Mgmt** tab on the left pane and click the **Add DC** link.

   The **DC Management** window appears with the **Display Profiles** pane.

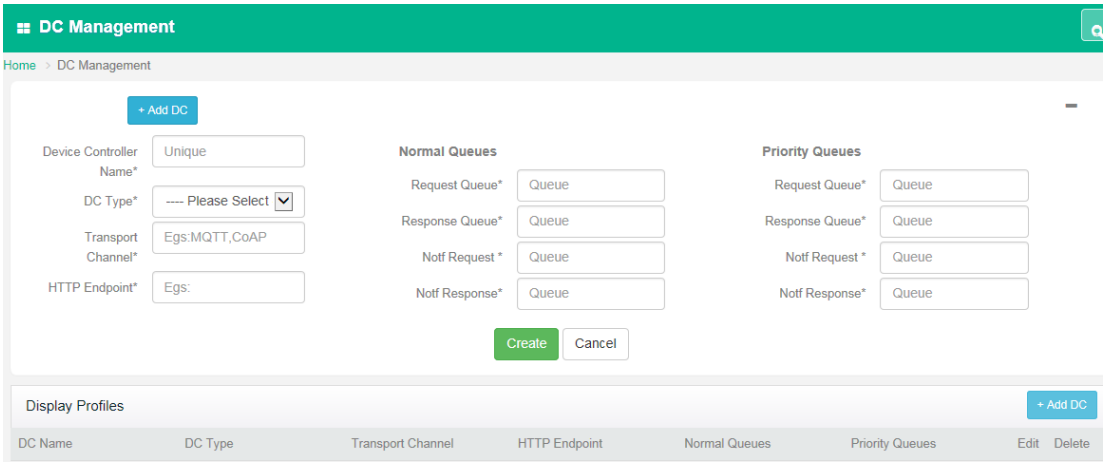


Figure 3 Add device controllers

3. Enter the following details.

Table 4 Add device controllers

Field	Description
Device Controller Name	Enter a unique name for the device controller.
DC Type	Select a type of controller from the drop-down list.
Transport Channel	Enter a unique name for each type of device controller. The name should also indicate the kind of protocol used. This ID is used in the device profile to identify the device controller for each device. NIP uses this name to route the traffic to the respective device controller.
HTTP Endpoint	Enter the URL used for sync communication from DAV to DC. Use the following URL if you are using the DC SDK: http://{host}:{port}/{dcname}/rest/postRequestPrimitive"
Normal Queues and Priority Queues	The UI populates the suggested names for queues according to the device controller name. These names should match the queue name provided in the device controller properties file.

4. Click the **Create** button.
   A confirmation appears. Restart the UIoT Platform to complete the onboarding process of the new device controller.

Chapter 2
Verify device controllers

2.1 Verify the installation

After completing the deployment and onboarding the device controller, you can verify the device controller using the ActiveMQ admin console.

1. Login to ActiveMQ Admin console.
2. Navigate to the **Queue** section.

   This section displays all the queues that are configured.
3. Verify whether the JMS consumers are initiated for all these queues.
   The number of consumers are configured in the property file.

2.2 Test the device controller

1. Create a device profile with a new Transport Channel.

```
<Metadata>
    <Manufacturer>Transponder</Manufacturer>
    <Model>Vehicle Transponder</Model>
    <Version>1</Version>
    <DeviceType>SENSOR</DeviceType>
    <DeviceSubType>ECU</DeviceSubType>
    <TransportChannel>AcmeDC</TransportChannel>
    <Device-Description>Vehicle Sensors</Device-Description>
    <ClassOfDevice>DEFAULT</ClassOfDevice>
    <DeviceProfileType>HPIOT</DeviceProfileType>
    <MessageFormat>Transponder test</MessageFormat>
    <Parameter name="String" type="long-unsigned">
      </Parameter>
    <OntologyReference xmlns:ns1="http://www.hp.com/schema/m2m/SEM/"/>
</Metadata>
```

2. Create a device with a new device profile.
3. Post a request to the device controller with the unique ID of the device
   (POST Content Instance)
4. Verify the values in DSM.

Appendix A
Supported protocols

Sl. No	Protocol	UIoT Platform Version	Supported Protocol and version	Protocol Operations	Unsupported Protocols / Protocol Operations	DC Request/ ResponsePrimitive Supported oneM2M Operations {Create(1), Retrieve(2), Update(3), Delete(4), Notify (5)}	Details
1.	CoAP DC	1.2 GA	CoAP (v RFC7252)	The DC implements a CoAP client (The device/gateway should act as a CoAP Server)			Communication direction: Downlink and Uplink Leveraging org.eclipse.californium CoAP implementation <version>1.0.0-M2</version> Interworking bindings based on: oneM2M - CoAP Protocol Binding (oneM2M TS-0008 version 1.3.2 Release 1) Documentation: Available at UIoT Platform Reference Desk
				GET : - Will deliver the list of attributes for a specific resource like a JSON response. POST: - Will check the content format of the request and	Open points: - Asynchronous CoAP request - CoAP Security	RETRIEVE CREATE, NOTIFY	

				will validate it with the value that the CoAP Resource is configured to use. - Will add the list of attributes from the payload request to the list of attributes of the CoAP Resource. - Will notify all the CoAP clients that have establish an observe relation with this CoAP Resource that the state has changed by reprocessing their original request that has established the relation. PUT: - Will check the content format of the request and will validate it with the value that the CoAP Resource is configured to use. - Will change the list of attributes of the CoAP Resource with the list of attributes from the payload request. - Will notify all the CoAP clients that have establish an observe relation with this CoAP Resource that the state has changed by reprocessing their original request		UPDATE	

				that has established the relation. DELETE: - Will remove the list of attributes from the current CoAP Resource. - Will remove all observe relations to CoAP clients and notify them that the observe relation has been canceled.		DELETE	
2.	LWM2M DC	1.2 GA		The DC implements a LWM2M Server based on the LWM2M spec from: OMA-TS-LightweightM2M Candidate Version 1.0 – 26 Nov 2014 (The Device/Gateway should act as a LWM2M Client)			Leveraging the Open Source Server: provided by Leshan <leshan.version>0.1.11-M6-SNAPSHOT</leshan.version> Support for High Availability is not provided in this release.
				CreateRequest			
				ReadRequest			
				ObserveRequest			
				ExecuteRequest			
					BootstrapRequest		
					DeleteRequest		
					Registerrequest		
					DeregisterRequest		
					DiscoverRequest		
					UpdateRequest		
					WriteAttributesRequest		
					WriteRequest		

3.	MQTT DC	1.2 GA	MQTT v3.1.1	TCP based MQTT Protocol – that receives notifications from the mqtt devices over activemq	MQTT Security	CREATE only	Communication Direction: Downlink & Uplink Leveraging org.eclipse.paho.mqttv3 MQTT implementations Eclipse Paho v1.0.2
4.	HTTP DC	1.2 GA	HTTP v2.0	HTTP based (REST) Device Controller – that communicates with the exposed generic HTTP devices		CREATE only	Communication Direction: Downlink & Uplink Supports oneM2M standard Request and Response (shortnames Enabled). One M2M version 1.6

Appendix B
Device Controllers developed for product demos

Sl. No	Protocol	Compatible with UIoT Platform Version	Supported Protocol and version	Supported Protocol & Protocol Operations	Unsupported Protocols / Protocol Operations	DC Request/ ResponsePrimitive Supported oneM2M Operations {Create(1), Retrieve(2), Update(3), Delete(4), Notify (5)}	Details
1.	ARM DC	1.2 CA (Demo)	HTTP v2.0	HTTP based (REST) Device Controller – that communicates with the exposed ARM Cloud API (https://api.connector.mbed.com)			Communication Direction: Downlink & Uplink Complete ARM MBED API Reference: https://docs.mbed.com/docs/mbed-device-connector-web-interfaces/en/latest/api-reference/
				(1) EndpointsAPI		CREATE	HttpMethod.GET
				(2) EndpointAPI		CREATE	HttpMethod.GET
				(3) ResourceAPI		CREATE & UPDATE	HttpMethod.GET & HttpMethod.PUT
				(4) SubscriptionsAPI		UPDATE & DELETE	HttpMethod.PUT & HttpMethod.DELETE
				(5) NotificationCallbackAPI		UPDATE	HttpMethod.PUT
				(6) ARM Notifications		CREATE	Expecting a PUT on DC_URL/rest/processNotifications
					See link in details for what's not currently covered.		

2.	Trackimo DC	1.2 CA (Demo)	HTTP v2.0	HTTP based (REST) Device Controller – that communicates with the exposed Trackimo Cloud API (https://app.trackimo.com/api/v3)		CREATE only	Communication Direction: Uplink Complete ARM MBED API Reference: https://app.trackimo.com/docs/api/v3/
				Geofence {"/accounts/{account_id}/geozones"}			
				DeviceLocation {/accounts/{account_id}/devices/{device_id}/history"}			
				Settings {"/accounts/{account_id}/settings"}			
				FireEvent {"/accounts/{account_id}/devices/ops/{event_type}"}			
				MovingAlarm, SpeedAlarm, GeofenceAlarm, LeftAlarm, RightAlarm and SosAlarm			
					AccountAPI {"/accounts/{account_id}"}		
					TagAPI {"/accounts/{account_id}/tags"}		
					AuthenticationAPI {"/oauth2/auth"}		
					UserAPI {"/user"}		

					ContactAPI {"/accounts/{account_id}/contacts"}		
					DeviceAPI {"/accounts/{account_id}/devices"}		
3.	Atrack OBD II DC	1. 2 GA (Demo)	TCP (IPv4)	TCP based OBD II Protocol Separate CODEC: AtrackAL1 – decodes to a JSON from Binary or ASCII Payload (see the documentation link for exact description of the format)	Downlink not yet supported	CREATE only	Communication Direction: Uplink only Documentation: Available at the UIoT Platform software Reference Desk
4.	Loriot DC	1.2 CA (Demo)	HTTP v2.0	HTTP based (REST) Device Controller – that receives notifications from the Loriot Cloud API (HTTP Push messages)	MQTT and WebSocket interfaces – not yet supported	CREATE only	Communication Direction: Uplink only Loriot Application API: https://www.loriot.io/home/documentation.html#docu/app-output
5.	Sigfox DC	1.2 CA (Demo)	HTTP v2.0	HTTP based (REST) Device Controller – that communicates with the exposed Sigfox Cloud API	See link in details for what's not currently covered. Open points: The Sigfox API credentials and the DC	CREATE only	Communication Direction: Downlink & Uplink Complete Sigfox MBED API Reference: https://backend.sigfox.com/apidocs/user/5729f84b51e24e

				(https://backend.sigfox.com) Supported Devices: Sensit & GroundMoisture	sigfox callback URL credentials are stored in plain text in a proeprties file.		616df58b66 (needs account) Codec library: Ø Sensit Ø GroundMoisture <TO BE UPDATED WITH: FOLDER CODEC SAMPLES>
				DeviceLocation: Get the messages location on GPS data or with the RSSI method, in reverse chronological order (most recent messages first).			
				DeviceMessages: Get the messages that were sent by a device, in reverse chronological order (most recent messages first)."			
				DeviceRegister: Registers new devices asynchronously by providing a list of identifiers, and associates			

				them to a device type.			
				Callback: Create/Delete and enable/disable callbacks			
				Notifications: Process notifications from devices"			
6.	Smartparking DC	1.2 CA	HTTP v2.0	JSON format transported over HTTP		CREATE only	Communication Direction: Uplink