IoTivity Programmer’s Guide – Resource Encapsulation

1 CONTENTS

2 REVISION HISTORY

Revision	Author(s)	Comments
2015061901	Arya Kumar	Initial Release
2015070601	Jay Sharma	Updated as per Review Comments
2015070901	Markus Jung	Resource Container documentation update Resource Bundle APIs and Project templates
2015071001	Jay Sharma	Updated as per Review comments and added sample application section
2015071401	Jay Sharma	Updated as per Review comments
2015072001	Jay Sharma	Added missed APIS, Added more description to the APIs.
2015072201	Jay Sharma	Changed name from Resource Manipulation to Resource Encapsulation , Updated as per review comments.
2015072202	Minji Park	Resource Container documentation update
2015072301	Minji Park	Updated as per Review Comments
2015072401	Jay Sharma	Added RCSAddress class for Discover method & added prefix to Resource client classes.
	Jay Sharma	Added RCS prefix to classes
2015073101	JungHo Kim	Remove all API references (Duplication of reference doc.)

3 Terminology

Resource Encapsulation

It is an abstract layer which consists of common resource function modules.

Resource Broker

It is function module of Resource Encapsulation layer which monitors the presence status of the Resource of Interest.

Resource Cache

It is the function module of Resource Encapsulation layer which manages the caching of Resource data.

Resource Client

It is the common API layer for the Resource Cache and Resource Broker module.

Server Builder

It is module which provides easy creation of resource with flexibility of handling the request either internally by module itself or at application level.

Resource Container

It provides the APIs for integration of non-OIC resources into OIC ecosystem.

Resource Bundle

It contains the resource information of non-OIC devices, which is used to create OIC resources.

4 Introduction to Resource Encapsulation

Resource Encapsulation is an abstract layer which consists of common resource function modules. It provides functionalities for both the client and server side to ease the work of developers.
For **Client side** it provides Cache and Resource Broker functionalities (monitoring the presence of resource in the network). For **server side** it provides the simple and direct way to create the resource and to set the properties and attributes. For handling the request from client it provides flexibility to developer either auto control of request by the layer itself or developer control the request in the application. **Resource Container** module of Resource Encapsulation layer provides integration of non-OIC devices into OIC devices.

Resource Encapsulation provides the common function modules to make developer's life easy.

4.1 Overall Architecture

This is an abstract view of the IoTivity architecture including the Resource Encapsulation layer of the IoTivity service.

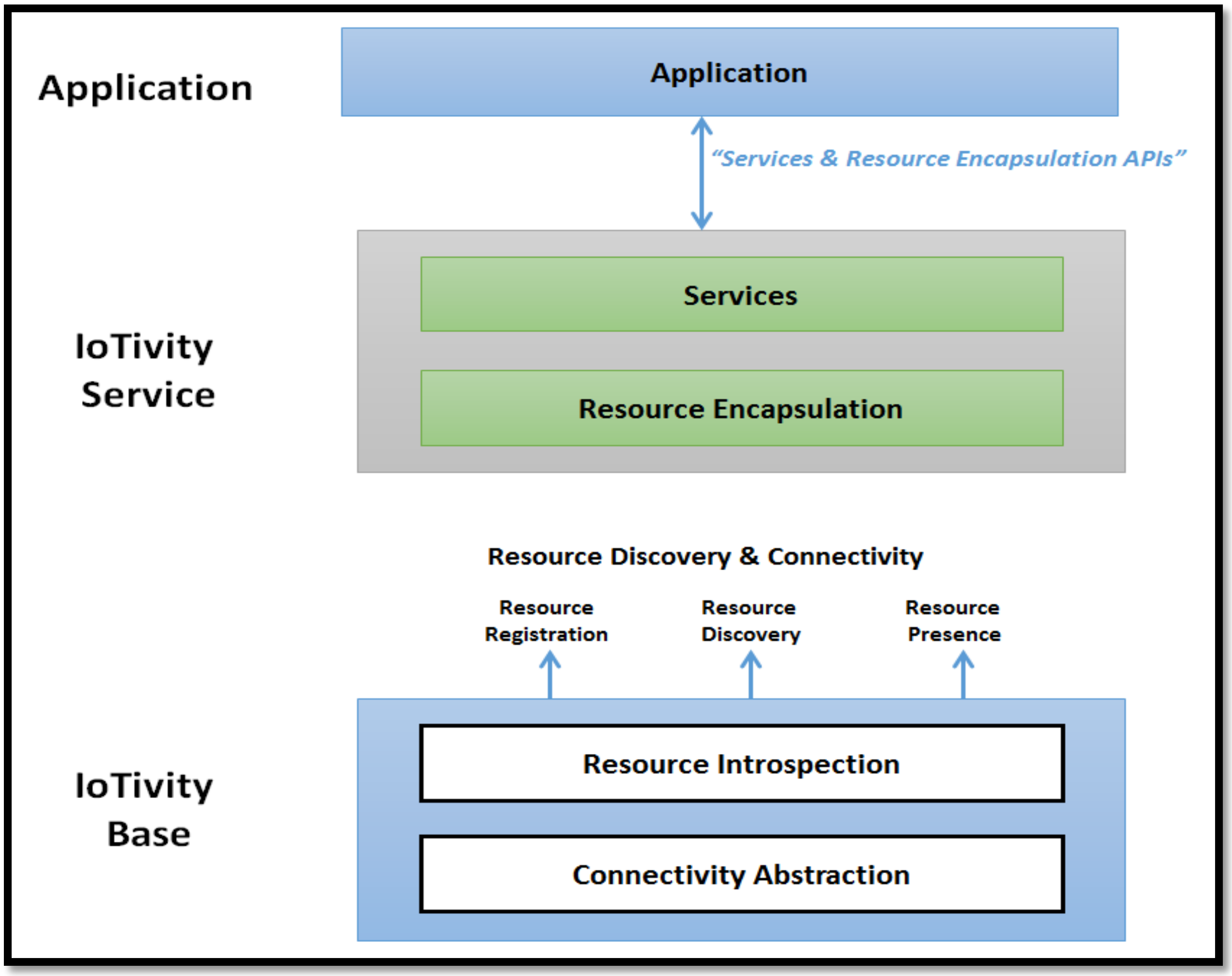


Figure 1 : IoTivity architecture depicting IoTivity services layer

4.2 IoTivity Service Components

The IoTivity service layer consists of two sub layers:

- **Service**:
  This layer contains service modules which in-turn uses the functional modules of RE layer.
- **Resource Encapsulation (RE):**
  This layer consists of common functional modules as shown in **Figure 2**.

The difference between both these layers is that the service layer has resource(s) to represent their features whereas RE layer do not have any resources.

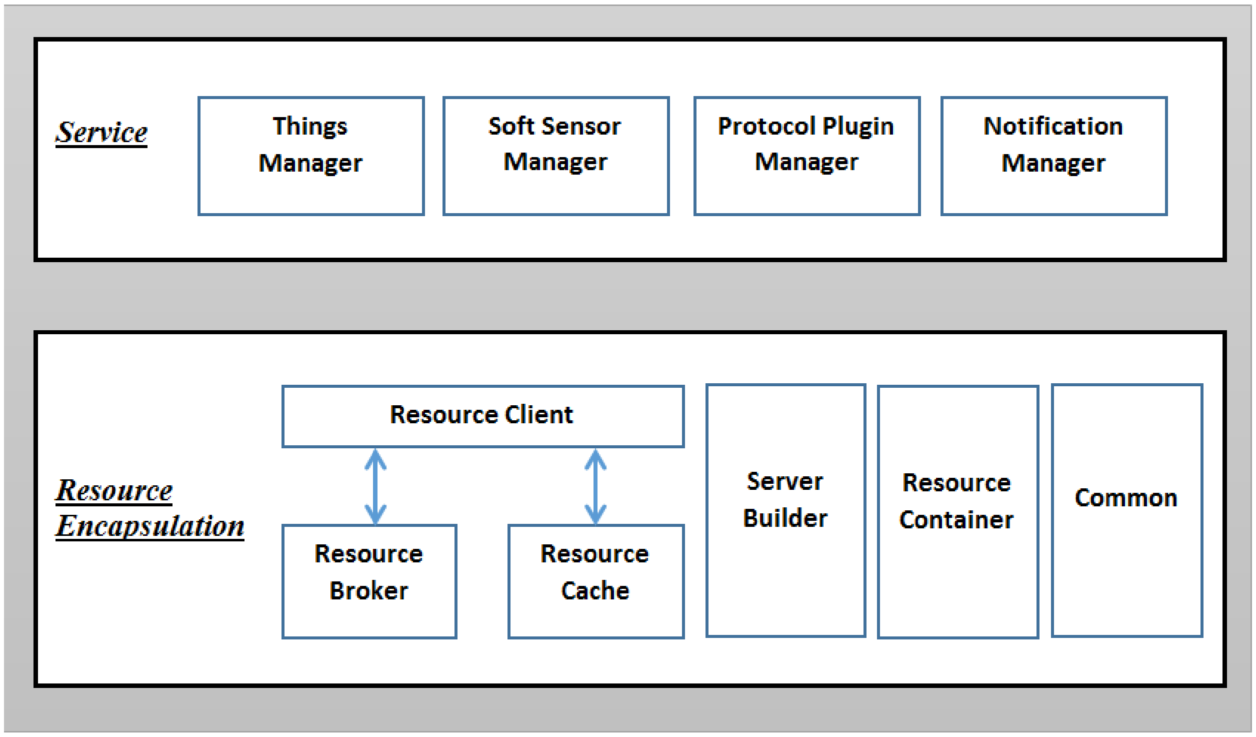


Figure 2: Iotivity service architecture depicting service modules and function modules

Here the **Resource Broker** and **Resource Cache** are functional modules which provide the client side functionalities for the IoTivity services. The **Resource client** is an API layer on these functional modules to provide these functionalities to the developer in an abstract way.

Server Builder is the functional module which provides server side APIs for easy creation of Resource and handling of requests.

Resource Container provides the easy integration of non-OIC resource to OIC resource.

Common provides the common APIs from both client and server side.

4.3 COMPONENTS OF RESOURCE ENCAPSULATION LAYER

4.3.1 Resource Broker

This is a function module in the resource encapsulation layer. It monitors the presence status of the resource of interest. It guarantees the presence status of the remote server (resource) selected & asked by application.

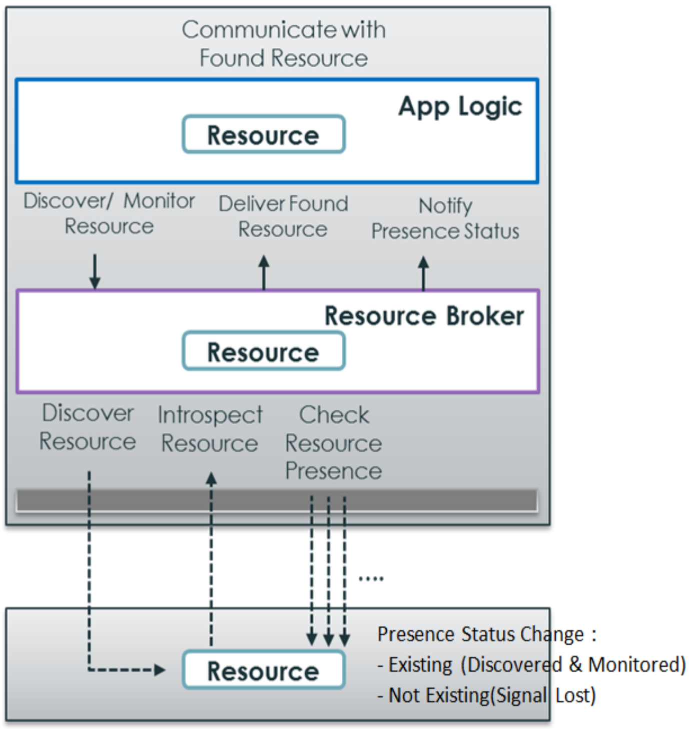


Figure 3: Resource Broker usage

4.3.2 Resource Cache

This is another function module in the resource client side. It caches the attribute data of the resource of interest. It guarantees the delivery of the resource data selected & asked by application. It has different methods of caching the resource data as per developer's requirements. These methods are specified in API section.

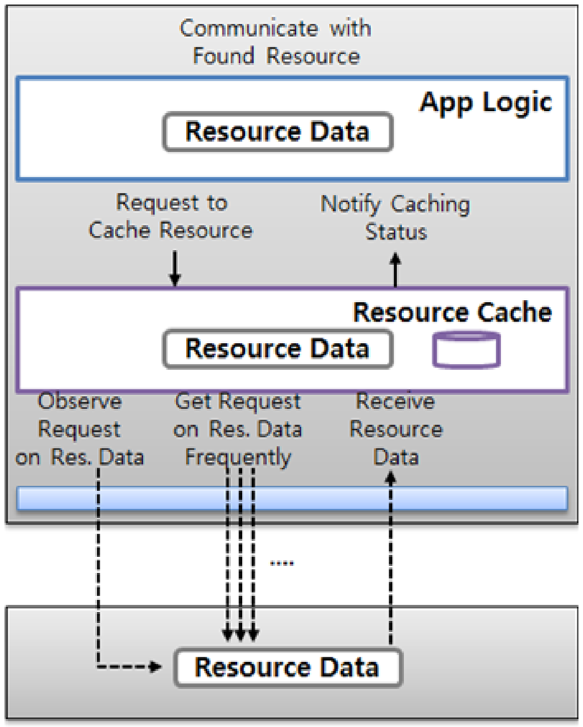

Communicate with Found Resource
App Logic
Resource Data
Request to Cache Resource
Notify Caching Status
Resource Cache
Resource Data
Observe Request on Res. Data
Get Request on Res. Data Frequently
Receive Resource Data
....
Resource Data

Figure 4: Resource Cache usage

4.3.3 Resource Client

Resource Client is a common API layer for the developer to use Resource Cache and Resource Broker functionalities. It provides the APIs of “Resource Broker” and “Resource Cache” to the developers.

4.3.4 Resource Container

The main purpose of this functional module is to:

- Manages the life-cycle of the resources.
- Provides APIs for integration of non-OIC resources into OIC ecosystem.
- Dynamic loading of resources bundles. One resource bundle can contain multiple resources. Bundle is activated by the container and bundle registers its resources at the resource container re-using the features of the other RE layer components.
- Provides common resource templates and configuration mechanism for resource bundles. It deals with OIC specific communication features, and provides common functionalities in a generic way.

It provides APIs to activate and deactivate resource instance(s) dynamically on demand.

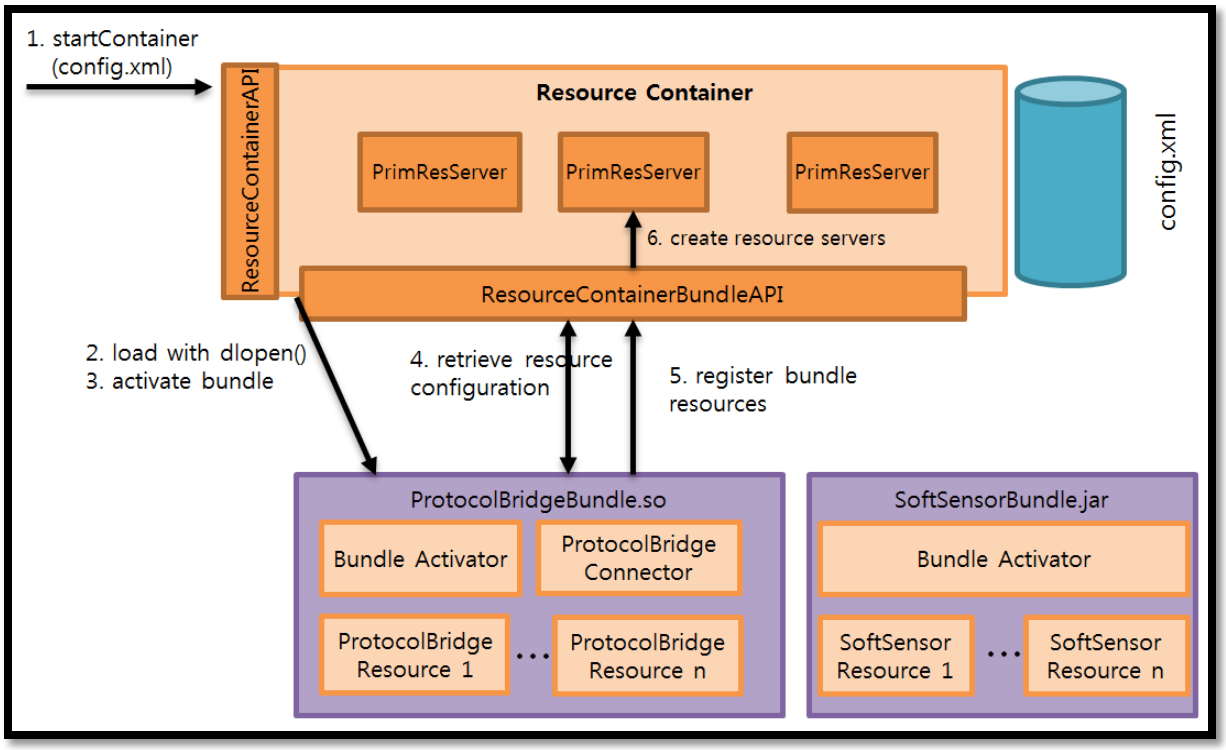
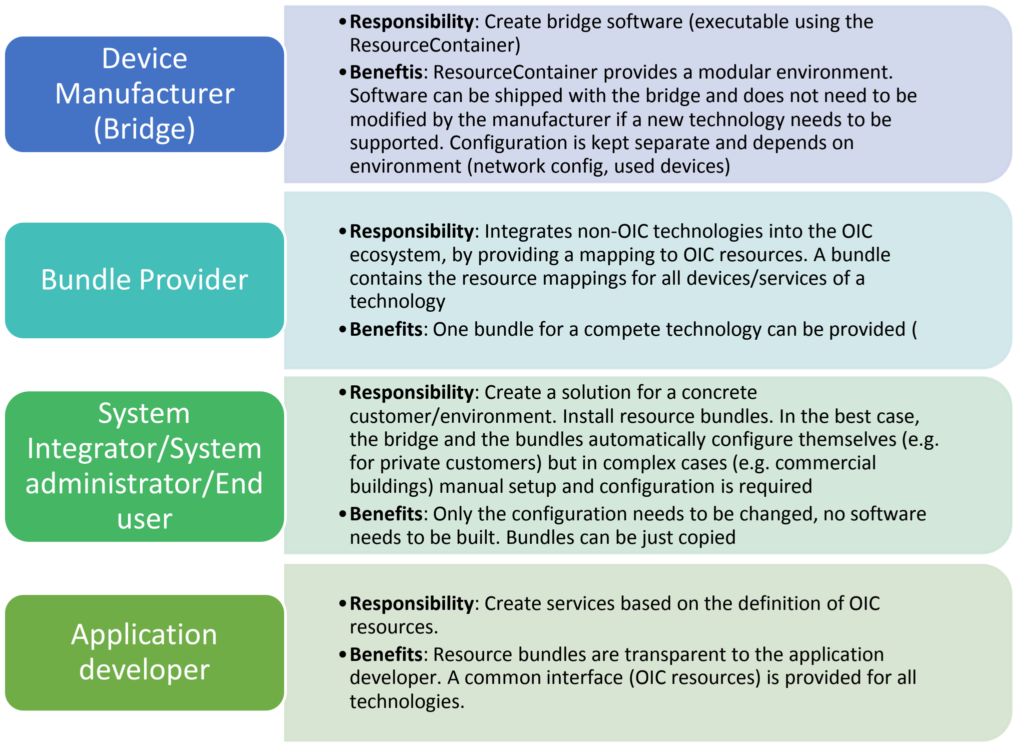


Figure 5: Resource Container Architecture

Figure 5 illustrates the architecture of the resource container. It offers a container API that can be used to start the container. A common XML configuration file is used for all resource bundles. The configuration contains parameters specific to the bundle but also to every resource instance. A resource bundle contains an activator and bundle resources. A bundle resource can be the definition of a soft sensor resource that contains an algorithm to derive new knowledge and offer it as a resource and protocol bridge resources which map other technologies to OIC resources. The bundles only contain the mapping logic, whereas the actual creation of OIC resource servers happens in the resource container. The bundle provider is agnostic of the base and resource encapsulation layer APIs of IoTivity and only needs to adhere to the bundle API.

The resource container is a component used by many different stakeholders. The main stakeholders are device manufacturers that provide a device bridge to other technologies, a bundle provider providing bundles that map other technologies to IoTivity or offer software-defined resources (e.g. algorithms, sensor fusioning), system integrator or end-user which configure the resource container and its bundles for a concrete environment. For an application developer the resource container is transparent and the developer has only to adhere to the OIC specified interfaces. A stakeholder overview is given in ***Figure 6*** and the interaction flow for the activation of a bundle is shown in ***Figure 7*.**



Device Manufacturer (Bridge)

- **Responsibility**: Create bridge software (executable using the ResourceContainer)
- **Beneftis**: ResourceContainer provides a modular environment. Software can be shipped with the bridge and does not need to be modified by the manufacturer if a new technology needs to be supported. Configuration is kept separate and depends on environment (network config, used devices)

Bundle Provider

- **Responsibility**: Integrates non-OIC technologies into the OIC ecosystem, by providing a mapping to OIC resources. A bundle contains the resource mappings for all devices/services of a technology
- **Benefits**: One bundle for a compete technology can be provided (

System Integrator/System administrator/End user

- **Responsibility**: Create a solution for a concrete customer/environment. Install resource bundles. In the best case, the bridge and the bundles automatically configure themselves (e.g. for private customers) but in complex cases (e.g. commercial buildings) manual setup and configuration is required
- **Benefits**: Only the configuration needs to be changed, no software needs to be built. Bundles can be just copied

Application developer

- **Responsibility**: Create services based on the definition of OIC resources.
- **Benefits**: Resource bundles are transparent to the application developer. A common interface (OIC resources) is provided for all technologies.

Figure 6: Resource container stakeholder overview

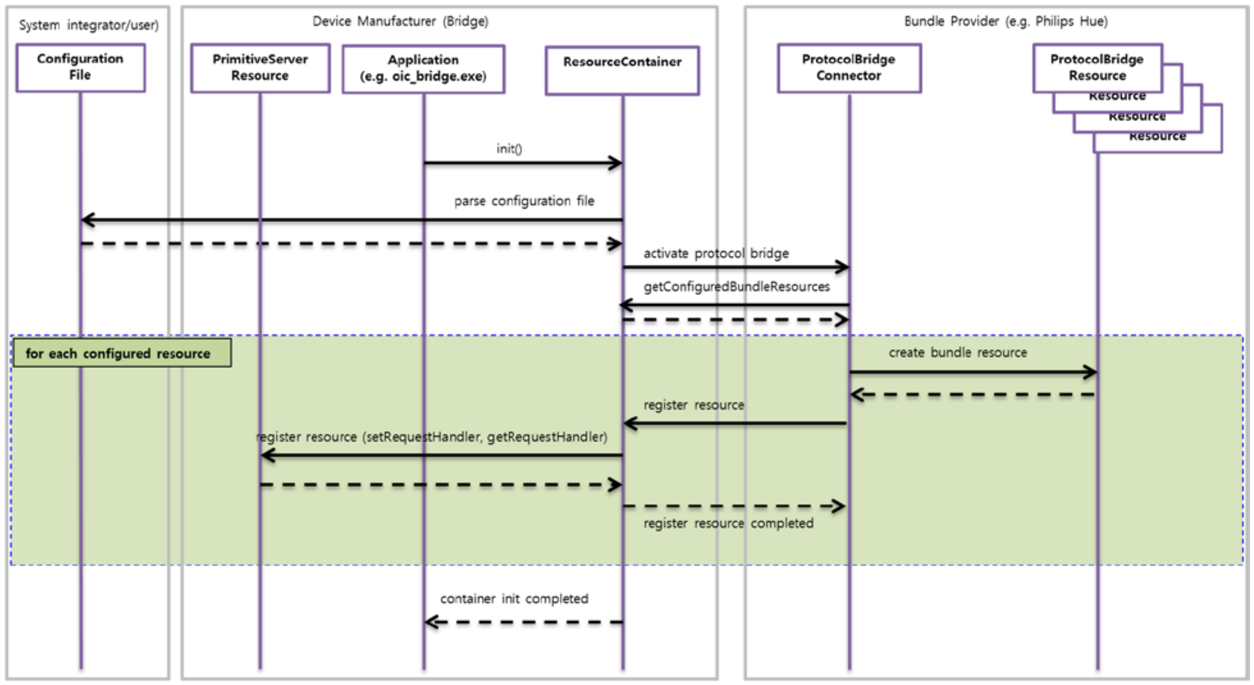


Figure 7: Resource container interaction flow

4.3.5 Server Builder

It is a functional module which handles the simplified creation of resources. In this module the developer does not need to deal with the details of CoAP communication, request and response handling. It provides APIs to ease the definition of resource types. The resources are defines based on the properties and developer has to provide the getter/setter methods. The developer does not have to worry about the request handling as it is taken care of internally in this module.

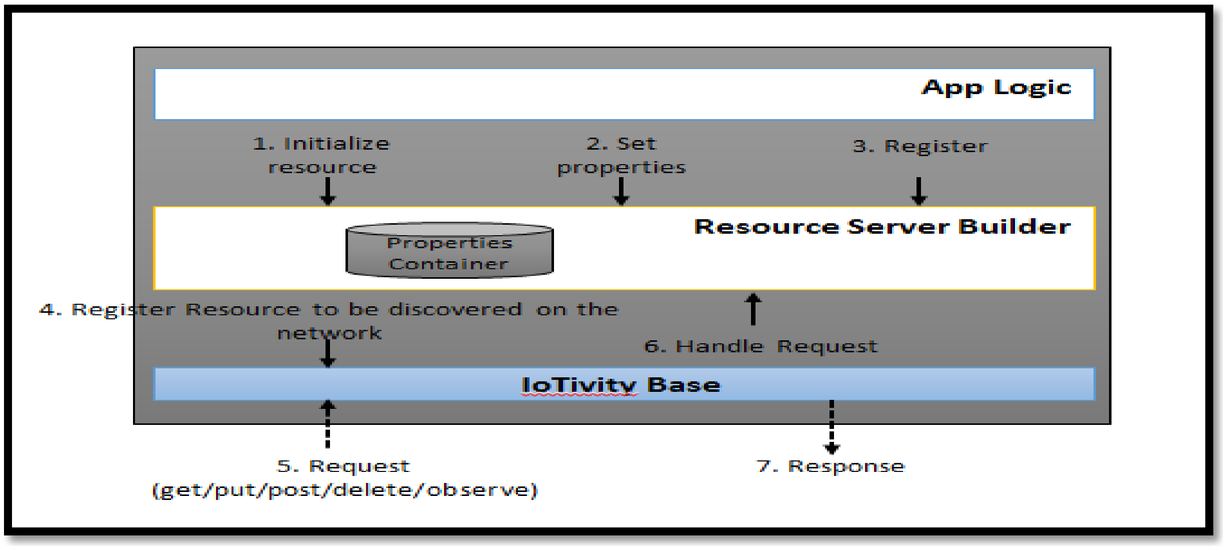


Figure 8: Server builder usage

5 BUILD INSTRUCTIONS

This section covers Cloning of Iotivity code and build instructions for Iotivity code.

5.1 CLONING IOTIVITY CODE

5.1.1 Tools and Libraries

The following tools and libraries are necessary to build IoTivity code in Linux machine for Linux platform. The commands and instructions provided in this section are specifically for Ubuntu LTS 12.04. Open the terminal window use the following instructions to install all the necessary tools and libraries to build an IoTivity project.

Ubuntu LTS 12.04

Ubuntu LTS version 12.04 is the supported OS for building the IoTivity stack. The instructions may be different for other versions of Ubuntu and Linux.

Git

Git is source code management software. Git is necessary to gain access to the IoTivitiy source code. Use the following command to download and install git:

```
$ sudo apt-get install git-core
```

ssh

Secure Shell is required to connect to the git repository to check out the IoTivity source code. Secure Shell is typically part of the base operating system and should be included. If for any reason it is not available, it can be installed by running the following command in your terminal window:

```
$ sudo apt-get install ssh
```

SCons

SCons is build tool used for compiling IoTivity source code. Please refer to the following link to install SCons.

http://www.scons.org/doc/production/HTML/scons-user.html#chap-build-install

Doxygen

Doxygen is a documentation generation tool used to generate API documentation for the IoTivity project. Download and install doxygen by running following command in your terminal window.

```
$ sudo apt-get install doxygen
```

5.1.2 Checking out the Source code

Gerrit is a web-based code review tool built on top of the git version control system. Gerrit's main features are side-by-side difference viewing and inline commenting, streamlining code review. Gerrit allows authorized contributors to submit changes to the git repository after reviews are done. Contributors can have code reviewed with little effort, and get their changes quickly through the system.

The following five steps describe how to check out the source code on the development machine.

Note: skip Step 1 to use existing ssh keys.

Step 1: Create ssh Key

On the terminal, type the following (replace "your name <your_email_address>" with your name and email address):

```
$ ssh-keygen –t rsa –C “your name<your_email_address_here>”
```

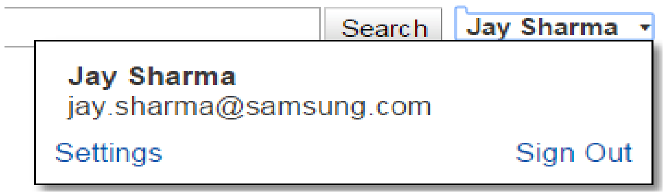
For example: Jay Sharma with an email address jay.sharma@samsung.com would type:

```
$ ssh-keygen –t rsa –C “Jay Sharma jay.sharma@samsung.com”
```

After pressing the **Enter** key at several prompts, an ssh key-pair will be created at ~/.ssh/id_rsa.pub.

Step 2: Upload and register an ssh public key

a. Log in to OIC Gerrit.
b. Click on **Settings** on the top right side as shown here:



c. Click on **SSH Public Keys** and add key.
d. Open ~/.ssh/id_rsa.pub, copy the content, and paste the content in the “Add SSH Public Key” window.
e. Click **Add**.

Step 3: Setting up ssh

a. Open ~/.ssh/config in a text editor.
b. Add the following lines:

   Host iotivity gerrit.iotivity.org
   Hostname gerrit.iotivity.org
   IdentityFile ~/.ssh/id_rsa
   User [Insert_your_username_here]
   Port 29418

c. To connect behind the proxy, add the following line after IdentityFile ~/.ssh/id_rsa with the appropriate proxy address and port:

   ProxyCommand nc –X5 –x <proxy-address>:<port> %h %p

Step 4: Verify your ssh connection

Execute the following command in the terminal window:

```
$ ssh gerrit.iotivity.org
```

Upon successful connection, the following message should appear indicating proper ssh and configuration connection.

******   Welcome to Gerrit Code Review   ******

If the connection is not established, check for the proxy and use the proxy settings described in Step 3.

Step 5: Cloning the project source

To build the IoTivity resource stack:

a. Using your terminal window, browse to the directory where code will be checked out.
b. Execute the following command in the terminal window to clone the iotivity repository:

```
$ git clone ssh://gerrit.iotivity.org/iotivity
```

The above command clones the repository in your current working directory.

5.2 BUILD IOTIVITY CODE FOR LINUX PLATFORM

To run the Linux Sample application of “Resource Encapsulation”, the Iotivity code should be built for Linux platform.

5.2.1 Build Procedure

To build the whole project, including the core, C SDK, C++ SDK, Resource Encapsulation samples:

- Navigate to the root of the iotivity directory using the terminal.
- Execute the **scons** command from the iotivity directory in the terminal:

```
 $ scons
```

If the build is successful you will see an out/linux folder in Iotivity directory.

6 RESOURCE ENCAPSULATION APIS

We can classify the Resource Encapsulation APIs in three categories : Client side APIs, Serve Side APIs and Common APIs.

6.1 CLIENT SIDE APIS

Resource Broker and Resource Cache provide the client side APIs. There is a common API layer “Resource Client” which provides APIs of Resource Broker and Resource Cache modules to developers. The Resource Client APIs are provided by two classes: **RCSDiscoveryManager** and **RCSRemoteResourceObject.**

6.2 SERVER SIDE APIS

Server side APIs are provided by the “**Server Builder**” and “**Resource Container”** module of Resource Encapsulation layer. It provides simple APIs to construct resources in a generic way and other utility APIs to perform server related operation on the resource(i.e. construct a server, host a server).

6.3 COMMON APIS

RCSResourceAttributes class represents the attributes for a resource. It overloaded the various operators, e.g., *==*, *[]*, *=*. It provides the APIs like begin, end and size from which *std::Map* provides. This API also provides two kinds of iterator to iterate over the attributes.

The common APIs has the following netsted-classes:

Value : for values of the attribute.

Type : for data type of the attributes

7 Resource Bundle APIs and Project Templates

The IoTivity resource container dynamically loads resource definitions from external libraries. The library consists of multiple bundle resource classes which can represent either a software defined sensor or a protocol bridge resource. A protocol bridge resources maps the interaction between OIC-based communication and arbitrary protocols. Third-parties can use this mechanism to integrate their technologies in the OIC eco-system. A so-called bundle combines multiple resource type definitions. A bundle has to provide a bundle activator, which is responsible to create resource instances and to register the resource instances at the resource container. The container offers an API to retrieve the bundle and resource configuration and to register/unregister resources.

The resource container supports C++ and Java libraries. According APIs and project templates ease the development of bundles.

7.1 C++ Bundle APIs and Project Template

A C++ bundle is a shared object library (.so) which is dynamically loaded by the resource container. To create a C++ bundle the header files located in the *resourceContainer/bundle-api* folder have to be included by a bundle developer. A sample C++ template project is provided in *resourceContainer/examples/HueSampleBundle*. To create a bundle the header files of the bundle API need to be included. A bundle has then to provide a bundle activator and its resource definitions. A protocol bridge bundle shall further provide a connector.

7.1.1 Project template

A project template to create a C++ resource bundle is given in *resourceContainer/examples/HueSampleBundle*.

A resource bundle has to define an activater, a connector and resource classes. The following external functions need to be defined, and trigger the creation and the execution of the activator.

```
extern "C" void externalActivateBundle(ResourceContainerBundleAPI *resourceContainer,
                                       std::string bundleId)
{
    bundle = new HueSampleBundleActivator();
    bundle->activateBundle(resourceContainer, bundleId);
}

extern "C" void externalDeactivateBundle()
{
    bundle->deactivateBundle();
    delete bundle;
}

extern "C" void externalCreateResource(resourceInfo resourceInfo)
{
    bundle->createResource(resourceInfo);
}

extern "C" void externalDestroyResource(BundleResource *pBundleResource)
{
    bundle->destroyResource(pBundleResource);
}
```

7.1.2 Build instructions

To build the bundle API and a C++ resource bundle, include the header files of the bundle-api into a new project and provide the resource container library.

Scons can be used to create a resource library. The Sconscript of the re sourceContainer shows how to build a resource bundle.

7.2 JAVA BUNDLE APIS AND PROJECT TEMPLATE

A Java bundle offers the capability to reuse existing Java-based communication libraries and integrate your protocols. The Java bundle mechanism can also be used in an Android environment. A Java bundle developer only needs to take care about the mapping between the OIC resource representation and the integrated technology. The required interfaces to create a Java bundle can be found in the *resourceContainer/bundle-java-api* directory. The API consists of a set of interfaces and abstract classes. Most important is the *BaseActivator* class, which has to be extended by a bundle provider. It offers the methods for retrieving configuration parameters and for resource registration. The second important abstract class is the *BundleResource*. A developer has to extend this class for concrete resource types and implement the mapping for reads and writes on the resource attributes.

7.2.1 Project Template

A Java example project template is provided in the folder *examples/HueJavaSampleBundle/hue*.

To create a new Java bundle, copy this folder and modify the Maven pom.xml. Source files can be put in an arbitrary

7.2.2 Build instructions

For building the Java Bundle APIs and the bundle projects the Maven[1] build system is used.

Note (using Maven behind firewall or proxy):
If you are behind a firewall or proxy it might be required to ease the security settings. Provide the following parameters to your Maven commands if external dependencies are downloaded.
-Dmaven.wagon.http.ssl.insecure=true
-Dmaven.wagon.http.ssl.allowall=true
-Dmaven.wagon.http.ssl.ignore.validity.dates=true

Bundle API

First step is to build the Java Bundle API.

```
resourceContainer/bundle-java-api> mvn compile
resourceContainer/bundle-java-api> mvn install
```

This compiles the bundle api and installs it in the local maven repository. All bundle projects can use the library then.

A project can include declare a dependency on the Java bundle API. An example pom.xml can be found in the *resourceContainer/examples/HueJavaSampleBundle* folder.

Java resource bundle

To build a Java bundle all dependencies need to be included. In order to package all dependencies the assembly plugin is used.

8 SAMPLE APPLICATION : RESOURCE CLIENT & SERVER BUILDER

This section describes about the sample applications using the **Resource Client** and **Server Builder** APIs.

8.1 WORKING FLOW

This section describes the working flow of the **SampleResourceClient** and **SampleResourceServer** Linux Applications.

These sample applications show the functionalities provided by **Resource Client** (Common API layer over **Resource Broker** and **Resource Cache**) and **Serverbuilder** to ease the life of developer.

We have two linux applications :

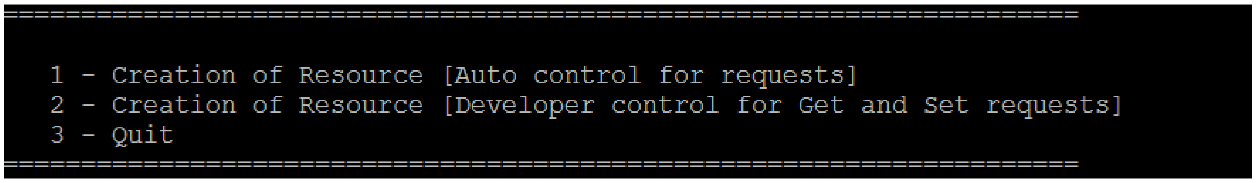
[1]maven.apache.org

SampleResourceClient & **SampleResourceServer** (act as **Temperature Sensor**)

First run the **SampleResourceServer**:

```
~/iotivity/service/resource-encapsulation/examples/linux $ ./SampleResourceServer
```

Following logs will be shown:



```
=====================================================================


   1 - Creation of Resource [Auto control for requests]
   2 - Creation of Resource [Developer control for Get and Set requests]
   3 - Quit
=====================================================================
```

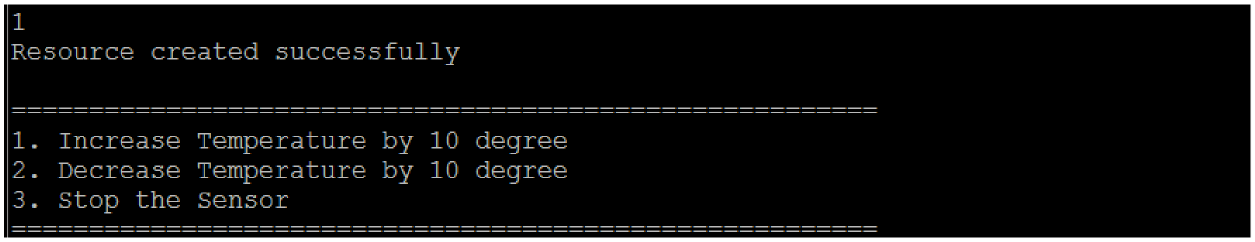
Sample Application is providing two options for creation of resource.

If we select the first option, it will create the resource and handling of all requests from client will be taken care internally by **ResourceBuilder** module.
If we select the second option, it will create the resource and handling of get and set request will be done by the application.

8.1.1 Server with Auto Control of Request

In this section, We will create the resource using the first option i.e. request handling will be done internally. Application will just create the resource, set the resource properties and Attributes.

Once we select 1st option, the following logs will be shown:

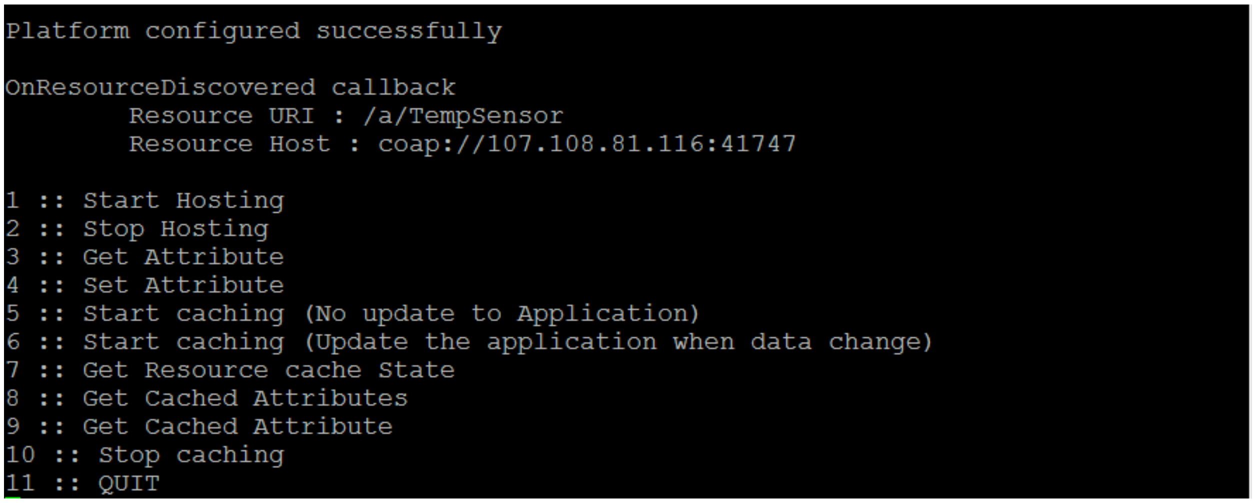


```
1
Resource created successfully

========================================================
1. Increase Temperature by 10 degree
2. Decrease Temperature by 10 degree
3. Stop the Sensor
========================================================
```

Now we will run the **SampleResourceClient**:

```
~/iotivity/service/resource-encapsulation/examples/linux $ ./SampleResourceClient
```

Following logs will be shown:

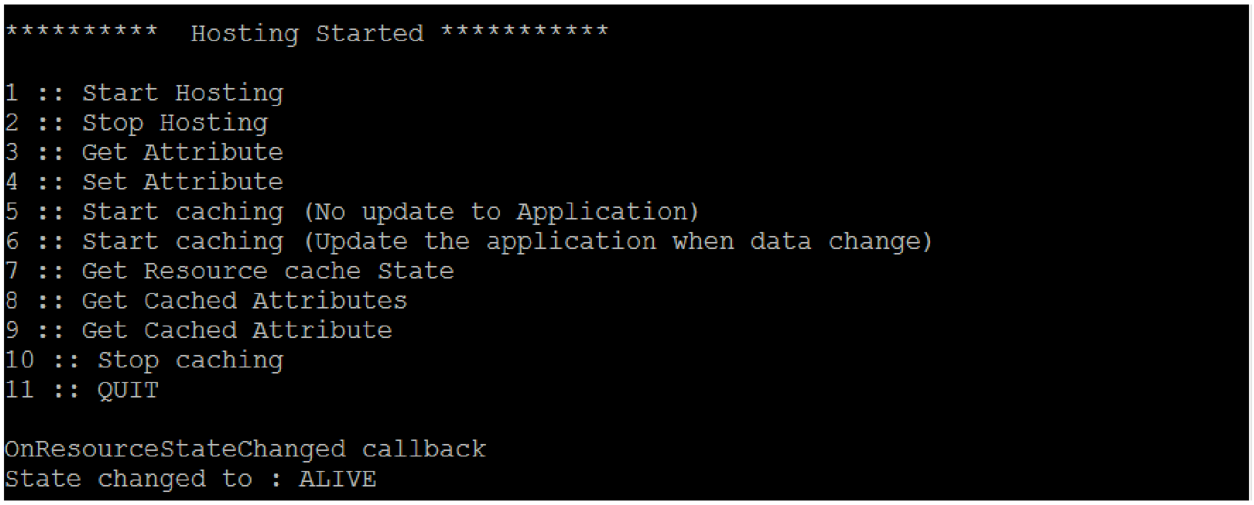


```
Platform configured successfully

OnResourceDiscovered callback
        Resource URI : /a/TempSensor
        Resource Host : coap://107.108.81.116:41747

1 :: Start Hosting
2 :: Stop Hosting
3 :: Get Attribute
4 :: Set Attribute
5 :: Start caching (No update to Application)
6 :: Start caching (Update the application when data change)
7 :: Get Resource cache State
8 :: Get Cached Attributes
9 :: Get Cached Attribute
10 :: Stop caching
11 :: QUIT
```

So, we have discovered the temperature sensor. we can test the ResourceClient APIs.

- If we select option 1 in the "SampleResourceClient" it will start to monitor the resource and notify the application the presence status of the resource.

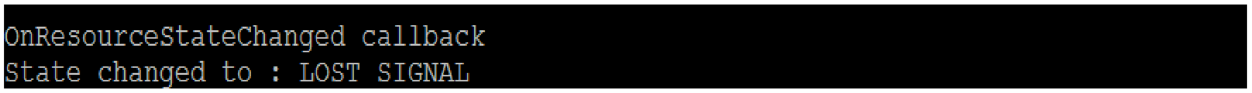
The following logs will be shown:



```
**********  Hosting Started ***********

1 :: Start Hosting
2 :: Stop Hosting
3 :: Get Attribute
4 :: Set Attribute
5 :: Start caching (No update to Application)
6 :: Start caching (Update the application when data change)
7 :: Get Resource cache State
8 :: Get Cached Attributes
9 :: Get Cached Attribute
10 :: Stop caching
11 :: QUIT

OnResourceStateChanged callback
State changed to : ALIVE
```

As the server is running we are getting the resourceState as "ALIVE".

Now if we stop the server the following logs will be shown in ResourceClient Application.



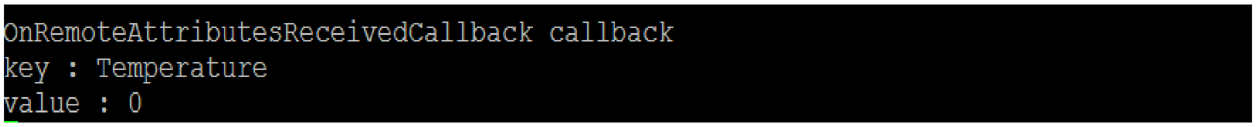
```
OnResourceStateChanged callback
State changed to : LOST SIGNAL
```

If We stop monitoring by selecting the second option in the application it will stop to check the presence of resource in the network. We can stop hosting by selecting the second option in the app. Now if resource goes out of network or destroyed there will be no notification will be there as we stopped the server.

Note: ***As we have stopped the server, we have to again start both the applications in order to test, rest of the functionalities.*** **Start the SampleResourceSever and SampleResourceclient in the same way as we did earlier.**

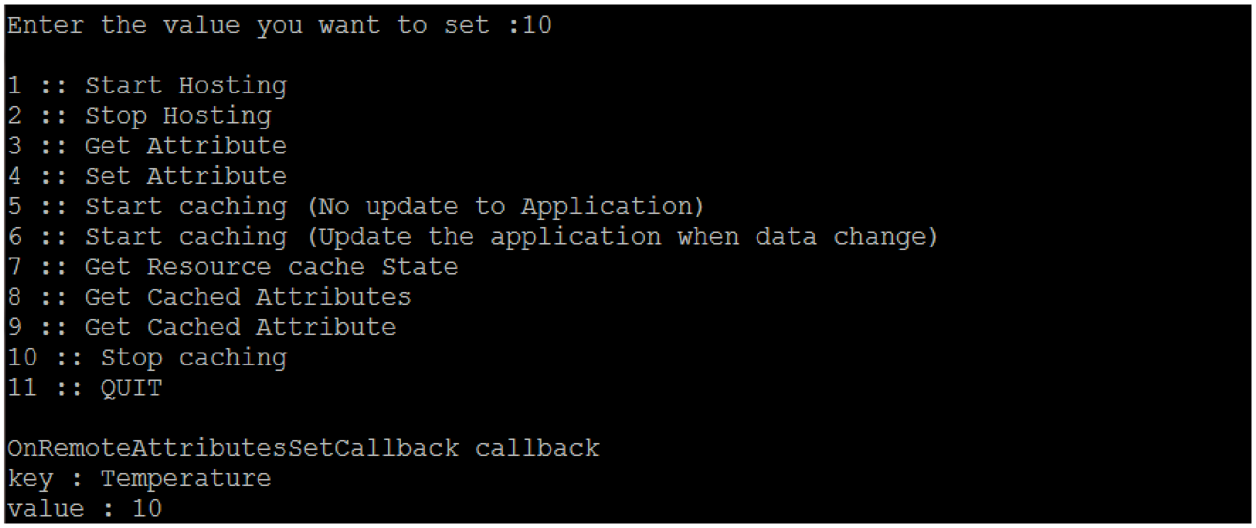
- We will select 3rd option to get the attributes from the server.

The following logs will be shown in the SampleResourceClient application:



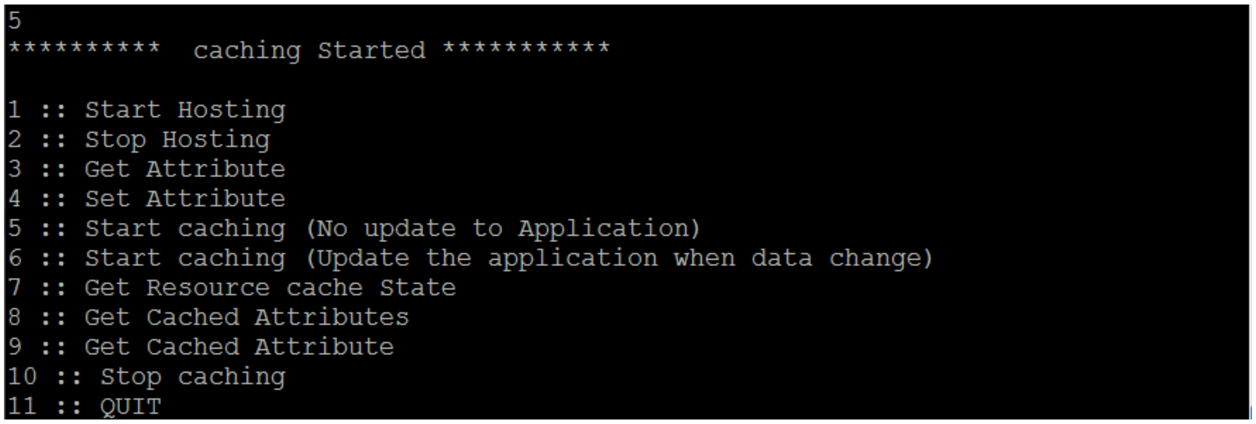
```
OnRemoteAttributesReceivedCallback callback
key : Temperature
value : 0
```

- If we want to set the attributes we can set it by selecting the 4th option in the application. In this example we are setting temperature value to 10. The following logs will be shown.



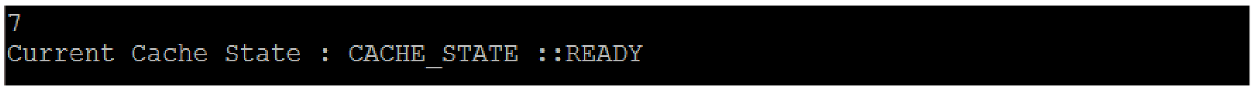
```
Enter the value you want to set :10

1 :: Start Hosting
2 :: Stop Hosting
3 :: Get Attribute
4 :: Set Attribute
5 :: Start caching (No update to Application)
6 :: Start caching (Update the application when data change)
7 :: Get Resource cache State
8 :: Get Cached Attributes
9 :: Get Cached Attribute
10 :: Stop caching
11 :: QUIT

OnRemoteAttributesSetCallback callback
key : Temperature
value : 10
```

- If we want to startCaching we will select 5th option in the application. It will cache the updated data from the server. And provide the cachedAttributes on demand.



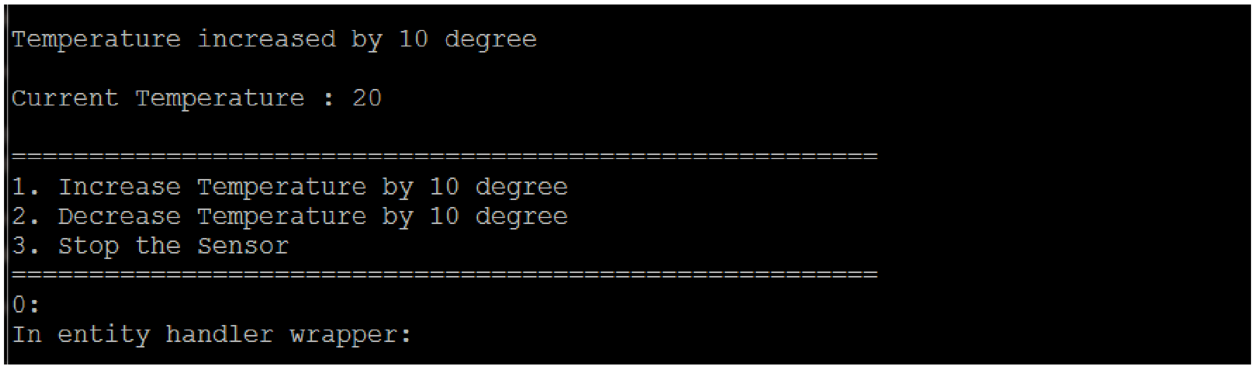
```
5
**********  caching Started ***********

1 :: Start Hosting
2 :: Stop Hosting
3 :: Get Attribute
4 :: Set Attribute
5 :: Start caching (No update to Application)
6 :: Start caching (Update the application when data change)
7 :: Get Resource cache State
8 :: Get Cached Attributes
9 :: Get Cached Attribute
10 :: Stop caching
11 :: QUIT
```

- If select option 7 in the ResourceClient sample app it will show the following logs:



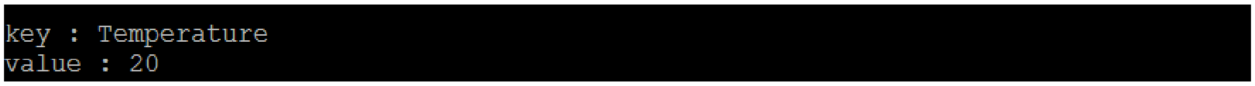
```
7
Current Cache State : CACHE_STATE ::READY
```

- We will change the temperature value on the server by selecting option 1 in the server. The following logs will be shown:



```
Temperature increased by 10 degree

Current Temperature : 20

========================================================
1. Increase Temperature by 10 degree
2. Decrease Temperature by 10 degree
3. Stop the Sensor
========================================================
0:
In entity handler wrapper:
```

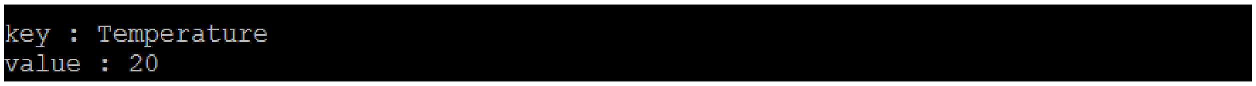
If we select option 8 in the application we will get the updated cached Attributes.



```
key : Temperature
value : 20
```

If we select option 9 in the application we will get the particular cached attribute value. In this example we are getting the cached attribute value corresponding to “Temperature”.

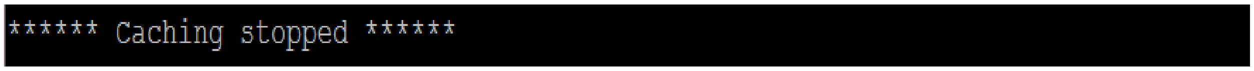
Following logs will be shown:



```
key : Temperature
value : 20
```

- If we want to stop the caching we can do it by selecting the 10th option in the Application. It will stop data caching for the resource that we requested earlier using startCaching option.

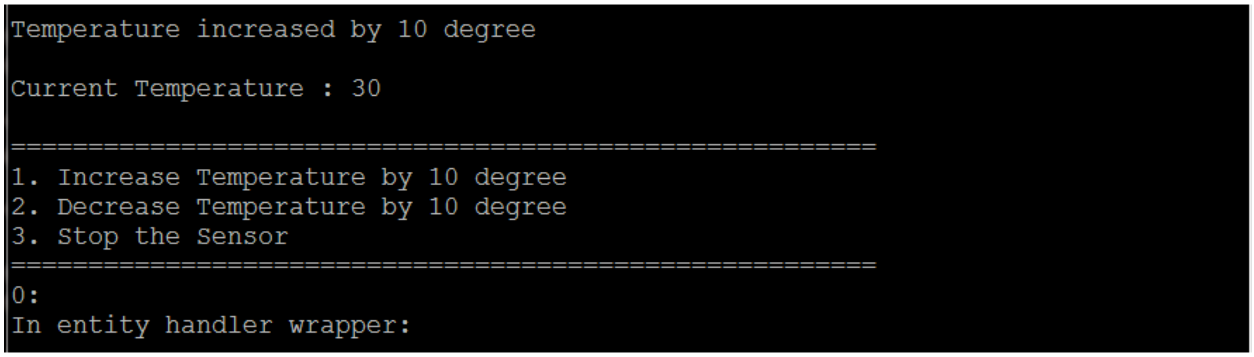
The following logs will be shown:



```
****** Caching stopped ******
```

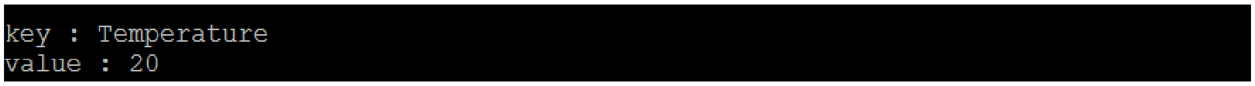
- ***Now, if we update the Temperature value on the server cached attributes will not be updated.***

So, first we will increase the temperature on the server by selecting option 1 in the ResourceServer app. The following logs will be shown:



```
Temperature increased by 10 degree

Current Temperature : 30

========================================================
1. Increase Temperature by 10 degree
2. Decrease Temperature by 10 degree
3. Stop the Sensor
========================================================
0:
In entity handler wrapper:
```

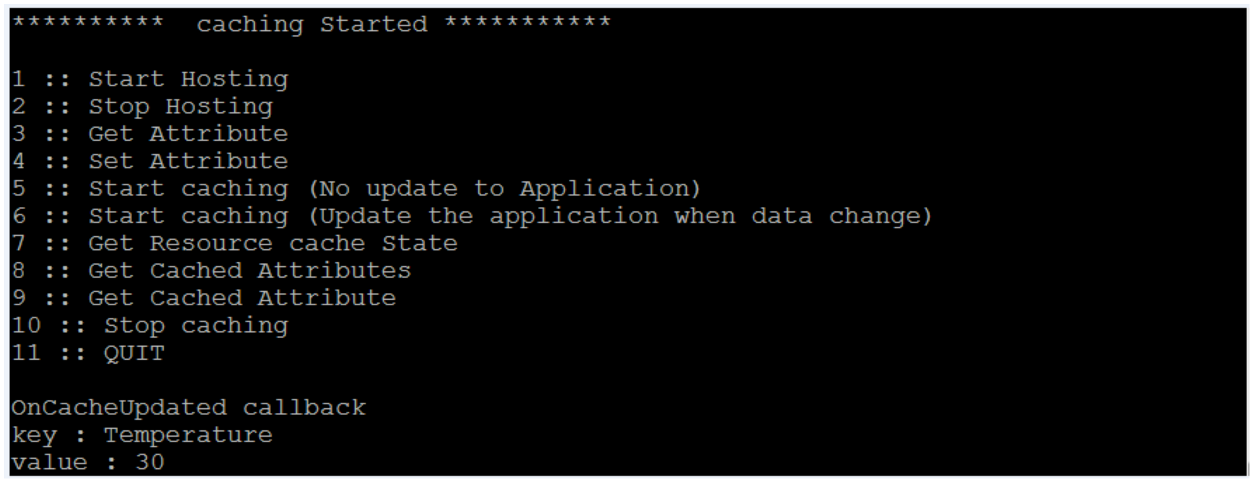
Now, we will select option 8 in the application to get the current CacheAttributes in the cache. The following logs will be shown:



```
key : Temperature
value : 20
```

So, before increasing the temperature on the server we have stopped the caching that is why we are getting cachedAttrbute value as 20 not 30.

- ➢ Now, we startCaching (update the application regarding the data change in the resource) by pressing the 6th option in the SampleResourceClient. The following logs will be show

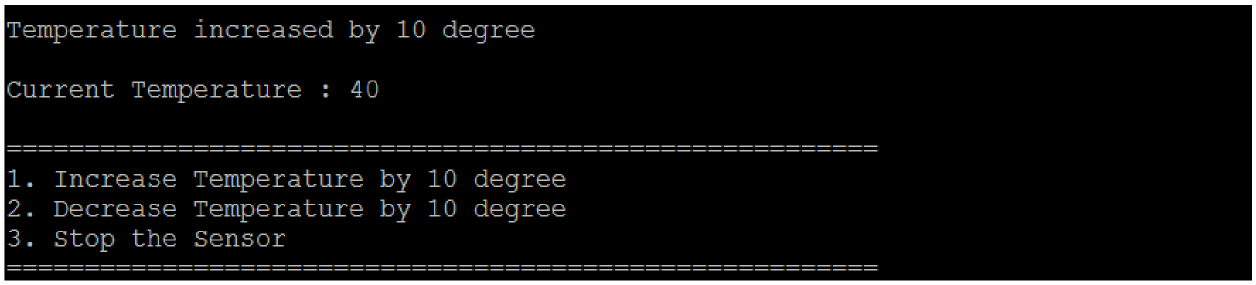


```
**********  caching Started ***********

1 :: Start Hosting
2 :: Stop Hosting
3 :: Get Attribute
4 :: Set Attribute
5 :: Start caching (No update to Application)
6 :: Start caching (Update the application when data change)
7 :: Get Resource cache State
8 :: Get Cached Attributes
9 :: Get Cached Attribute
10 :: Stop caching
11 :: QUIT

OnCacheUpdated callback
key : Temperature
value : 30
```

We have got the current cachedAttribute in the application

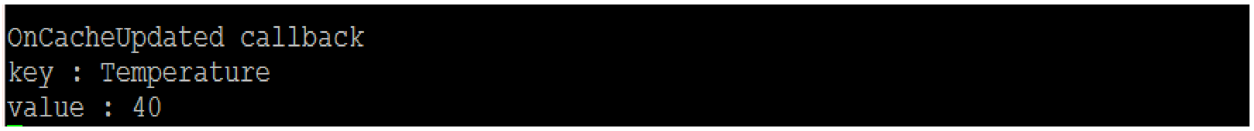
- ➢ ***Now, if we update the Temperature value on the server, application will get the updated CachedAttributes.***

So, first we will increase the temperature on the server by selecting option 1 in the Resource Server app. The following logs will be shown:



```
Temperature increased by 10 degree

Current Temperature : 40

========================================================
1. Increase Temperature by 10 degree
2. Decrease Temperature by 10 degree
3. Stop the Sensor
========================================================
```

Following logs will be shown on SampleResourceClient Application:



```
OnCacheUpdated callback
key : Temperature
value : 40
```

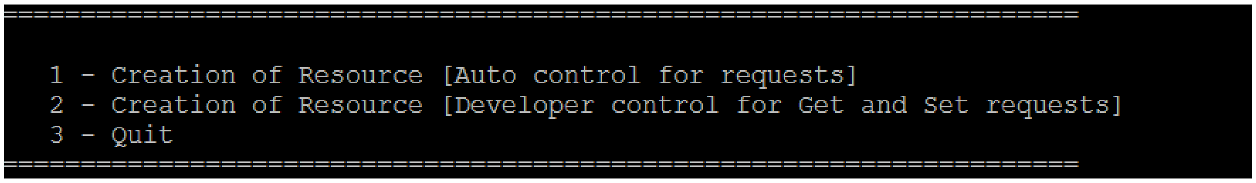
8.1.2 Server with application's Control on Request

Now, we test the resource creation using the 2nd option in the SampleResourceServer linux Application.

So, first we will run the server:

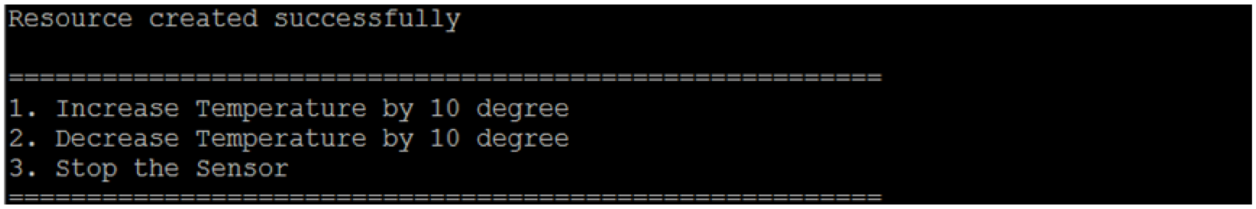
```
~/iotivity/service/resource-encapsulation/examples/linux $ ./SampleResourceServer
```

Following logs will be shown:



```
=====================================================================

   1 - Creation of Resource [Auto control for requests]
   2 - Creation of Resource [Developer control for Get and Set requests]
   3 - Quit
=====================================================================
```

We will select 2nd option i.e. It will create the resource and handling of get and set request will be handled by application. The following logs will be shown:

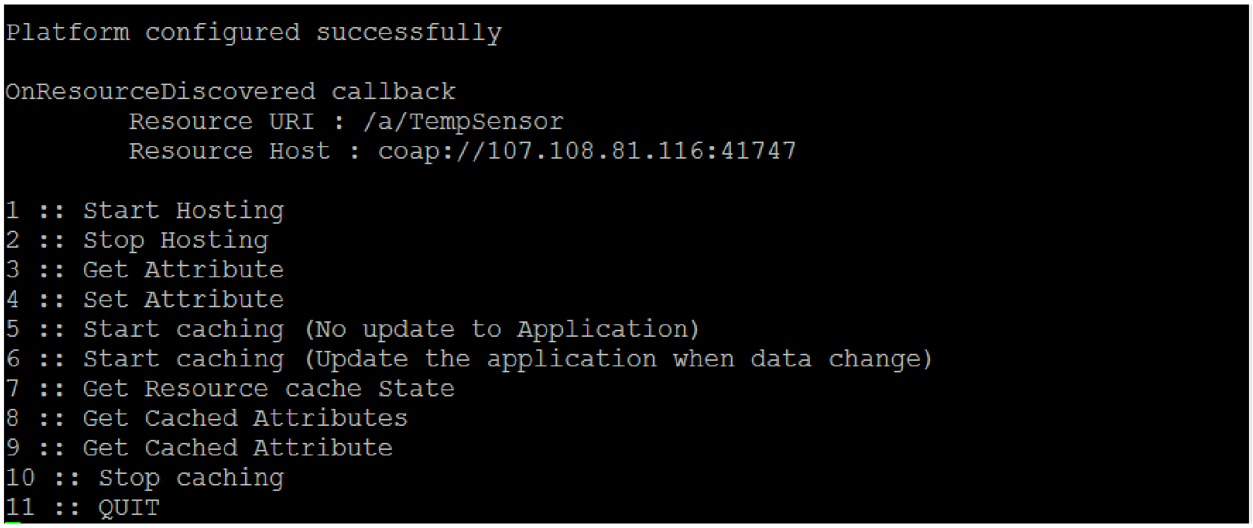


```
Resource created successfully

========================================================
1. Increase Temperature by 10 degree
2. Decrease Temperature by 10 degree
3. Stop the Sensor
========================================================
```

Now we will run the **SampleResourceClient:**

```
~/iotivity/service/resource-encapsulation/examples/linux $ ./SampleResourceClient
```

The following logs will be shown in **SampleResourceClient** application:

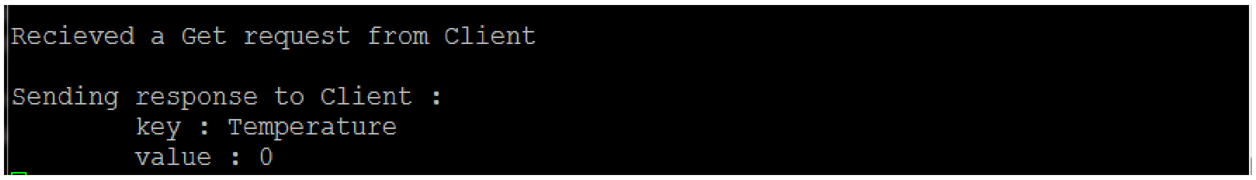


```
Platform configured successfully

OnResourceDiscovered callback
        Resource URI : /a/TempSensor
        Resource Host : coap://107.108.81.116:41747

1 :: Start Hosting
2 :: Stop Hosting
3 :: Get Attribute
4 :: Set Attribute
5 :: Start caching (No update to Application)
6 :: Start caching (Update the application when data change)
7 :: Get Resource cache State
8 :: Get Cached Attributes
9 :: Get Cached Attribute
10 :: Stop caching
11 :: QUIT
```

Here, we will show that how get and set request is handled by the application (server). We will see it using the get and set attributes request from our **SampleResourceClient** Application.

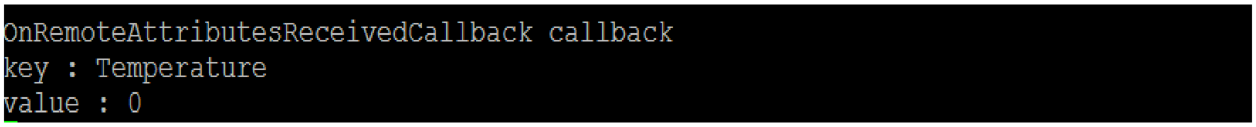
➢ To get the attributes from the server we will select 3rd option in **SampleResourceClient** application.

The following logs will be shown in the **SampleResourceServer** application:



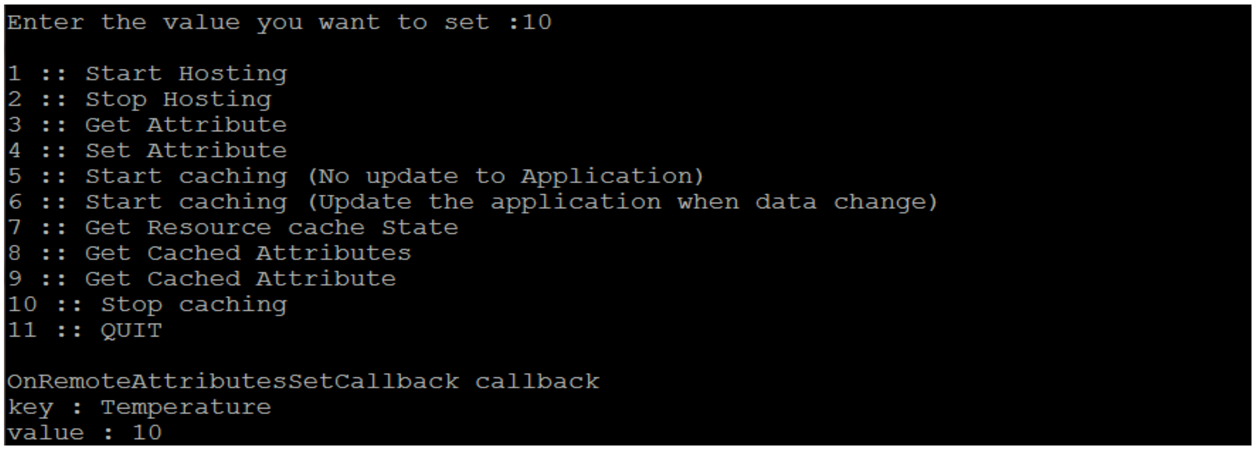
```
Recieved a Get request from Client

Sending response to Client :
        key : Temperature
        value : 0
```

The following logs will be shown in the **SampleResourceClient** application:



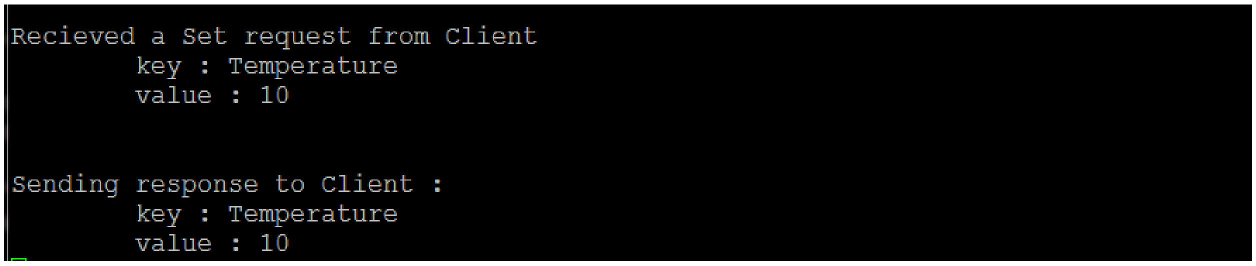
```
OnRemoteAttributesReceivedCallback callback
key : Temperature
value : 0
```

- We will select the 4th option in the ResourceClient Application to set the ResourceAttribute.
  The following logs will be shown in the **SampleResourceClient** application:



```
Enter the value you want to set :10

1 :: Start Hosting
2 :: Stop Hosting
3 :: Get Attribute
4 :: Set Attribute
5 :: Start caching (No update to Application)
6 :: Start caching (Update the application when data change)
7 :: Get Resource cache State
8 :: Get Cached Attributes
9 :: Get Cached Attribute
10 :: Stop caching
11 :: QUIT

OnRemoteAttributesSetCallback callback
key : Temperature
value : 10
```

The following logs will be shown in the **SampleResourceServer** application



```
Recieved a Set request from Client
        key : Temperature
        value : 10


Sending response to Client :
        key : Temperature
        value : 10
```

So, we are handling the get and set request in the application. **Server builder** is providing flexibility for controlling the request either internally or by application.