IoTivity Programmer's Guide
– Things Manager for Android

1 CONTENTS

2 THINGS MANAGER

Things Manager is one of the primitive services provided by Iotivity framework to manage a group of things in the network. Things Manager also provides rich functionalities to manage multiple things. With Things Manager functionalities and its offering SDK APIs, developers can easily implement a variety of applications to find candidate devices to form a group, create a group of found devices, create a group action for the group, and execute the group action. Also configuration and diagnostics function for multiple things can be supported by these APIs. The purpose of this guide is to provide the details for developers to understand and fully utilize given SDK APIs and how the Things Manager works to support the APIs. Lastly, this guide introduces explanatory scenarios for a group formation, group action, group synchronization, things configuration and things diagnostics and shows a series of procedure of sample applications to describe the scenarios for a better understanding on Things Manager.

3 TERMINOLOGY

3.1 GROUP

A set of devices in an IoTivity local network and remote networks for accomplishing the specific goal. Using several kinds of criteria, devices can be a member of a specific group. However, basically those member devices don't have any information about the group. Only the device that creates this group can have and maintain the information about this group.

Currently, resource type can be used as criteria for group formation and more criteria's will be provided in future.

3.2 ACTIONSET

A set of action descriptions needed by remote devices as the member of a specific group. For a particular group, multiple action sets can be assigned. One action set can have multiple actions and one action should be assigned to one specific member devices' characteristic. Currently only resource type can be used as device's characteristic.

To create an action set, one may need to know the Delimiter serialization. With the Delimiter, one specifies an action set as below.

```
movieTime*10 1*uri=coap://10.251.44.228:49858/a/light|power=on*
uri=coap://10.251.44.221:49858/a/light|power=on
```

A first segment before the first asterisk(*) is an action set name. The second segment indicates a time-related information which is used for scheduled/recursive group action features. The first digit in the segment is either delay or a step of time delay and the second digit is a type of group action, e.g., a normal/scheduled/recursive group action. If the type indicates a normal group action, the first digit will be ignored. If the type does a scheduled group action, the first digit will be utilized as a time delay such as *after* 10 seconds. If the type does a recursive group action, the first digit will be utilized as a step of time delay such as *every* 10 seconds.

The third segment goes before a next asterisk. In the above example, “uri=coap://10.251.44.228:49858/a/light|power=on” is the segment. This can be also divided into two sub segments by a vertical bar(|): URI and a pair of attribute key and value. The remained string from the second asterisk is same as the third segment.

3.3 Things Manager

A software service which helps to create a specific group and maintain that group. Group action feature -creating, maintaining and executing group action related with this group .

Richer API regarding configuration of multiple things and diagnostics of multiple things are provided by Things Manager.

A more detailed description of Things Manager and its relevant components will be provided later in this guide.

3.4 Things Configuration

Things Manager provides APIs to access a Configuration resource’s value to get/update a system parameter. The extent of what a Configuration resource covers could be all system-specific. On the server side, bootstrapping requisite information (i.e. system configuration parameters) from a bootstrap server to access other IoT services. On the client side, getting/updating the system configuration parameters from/to multiple remote things.

3.5 Things Diagnostics

The purpose of a Things Manager in terms of diagnostics is to request a system command (e.g., Factory Reset, Reboot) with a diagnostic purpose to a resource server by accessing a Diagnostics resource’s value from a client that is remotely located.

4 ANDROID SDK API

SDK API is the facet of Things Manager to applications as shown in the Figure 1.

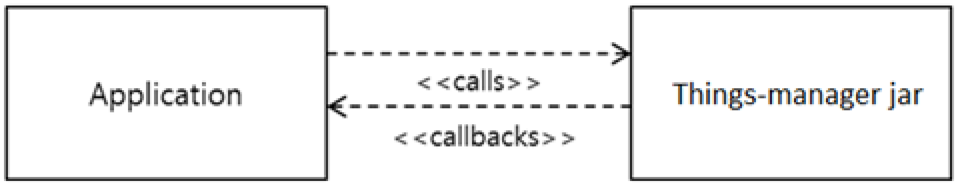


Figure 1 Things Manager SDK APIs and Application

4.1 Group Manager API

This APIs provides methods for application to find appropriate devices (i.e. things) in network, create a group of the devices, check presence of member devices in the group, and actuate a group action in a more convenient way.

The operations provided in the SDK are listed below:

- findCandidateResources
- subscribeCollectionPresence
- findGroup
- createGroup
- joinGroup
- leaveGroup
- deleteGroup
- getGroupList
- addActionSet
- executeActionSet
- cancelActionSet
- getActionSet
- deleteActionSet
- bindResourceToGroup

findCandidateResource() is to request Things Manager to find resources in network. The findCandidateResource() function has a key parameter to give a condition to find appropriate resources out of all available resources in network. One of conditions provided is a resource type.

With the parameter, developers can indicate a list of resource types they want to find and create a group.

Prototype:

- OCStackResult findCandidateResources(Vector<String> resourceTypes)
  [Without WaitTime]
- OCStackResult findCandidateResources(Vector<String> resourceTypes, int waitTime)
  [With WaitTime]

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setFindCandidateResourceListener(IFindCandidateResourceListener listener)

Callback for this API :

- public void onResourceCallback(Vector<OCresource> resources)

Note: Listeners are provided to receive responses asynchronously, users can register using **Set listener** methods provided by SDK.

Note that for registering a callback function to get notification when resources are discovered in the network, setFindCandidateResourceListener() is used.

subscribeCollectionPresence() is to request Things Manager to subscribe for the presence state of the resource. The only parameter is *OCResource* which indicates a target child resource.

Prototype:

- OCStackResult subscribeCollectionPresence(OCResource resource)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setSubscribePresenceListener(ISubscribePresenceListener listener)

Callback for this API

- public void onPresenceCallback(String resource, OCStackResult result)

Note that for registering a callback function to get child resource presence status, setSubscribePresenceListener() is used.

findGroup() is to request Things Manager to find a specific remote group when a resource tries to join a group. The only parameter is *collectionResourceTypes* which is the resource types of a group to find and join.

Prototype:

- OCStackResult findGroup(Vector<String> resourceTypes)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setGroupListener(IFindGroupListener listener)

Callback for this API :

- public void onGroupFindCallback(OCResource resource)

Note that for registering a callback function to get group discovery status, setGroupListener() is used.

createGroup() is to request a Things Manager to create a group that is not existing. The only parameter is *collectionResourceType* which is the resource type of a group to create.

Prototype:

- OCStackResult **createGroup** (String *collectionResourceType*)

  OCStackResult :   Returns OC_STACK_OK on Success , Error status code on Failure.

joinGroup() is to request Things Manager to join an existing group. This is an overloaded function. One is used when a resource that has a group tries to find a specific remote resource and makes it join a group. The first parameter is *collectionResourceType* which is the resource type of a group to join. The second parameter is *resourceHandle* which is the resource handle to join a group.

Prototype:

- OCStackResult joinGroup(String collectionResourceType, OCResourceHandle resourceHandle)

  OCStackResult :   Returns OC_STACK_OK on Success , Error status code on Failure.

On the other hand, the other one is used when a resource that doesn't have a group tries to find and join a specific remote group. The first parameter is *resource* which is the group resource pointer to join. The second parameter is *resourceHandle* which is the resource handle to join a group.

Prototype:

- OCStackResult joinGroup(OCResource resource, int resourceHandle)

  OCStackResult :   Returns OC_STACK_OK on Success , Error status code on Failure.

leaveGroup() is to request Things Manager to leave a joined group. The first parameter is *collectionResourceType* which is the resource type of a group to leave. The second parameter is *resourceHandle* which is the resource handle to leave a group.

Prototype:

- OCStackResult leaveGroup(String collectionResourceType, OCResourceHandle resourceHandle)

  OCStackResult :   Returns OC_STACK_OK on Success , Error status code on Failure.

deleteGroup() is to request Things Manager to delete a created group. The only parameter is *collectionResourceType* which is the resource type of a group to delete.

Prototype:

- OCStackResult deleteGroup(String collectionResourceType)

  OCStackResult :   Returns OC_STACK_OK on Success , Error status code on Failure.

getGroupList() is to request Things Manager to get a list of joined groups. This has no parameter and returns the map with the resource type of a group and group resource handle.

Prototype:

- Map<String, int> getGroupList()

addActionSet() is to request Things Manager to add a specific action set belonging to a specific resource. The first parameter is a *resource* which a new action set will be added onto. The second parameter is a *newActionSet* which is a target instance of ActionSet class to be added.

Prototype:

- int addActionSet(OcResource resource, ActionSet actionSet)

Register a callback for this API:

- public void setPutResourceListener (IOCResourcePutCallback listener)

Callback for this API :

- public void onPutResponseCallback(Vector <OCHeaderOption> headerOptions,
  OCRepresentation rep, int errorValue)

Note that for registering a callback function for the requested actions, setPutResourceListener() is used.

executeActionSet() is to request Things Manager to execute a specific action set belonging to a specific resource. This is a kind of overloaded functions and has two types. One is used when one wants to execute an action set immediately. The first parameter is a *resource* which is a target resource. The second parameter is an *actionsetName* which is a string indicating an action set name to be executed.

Prototype:

- OCStackResult executeActionSet(OCResource resource, String actionsetName)

  OCStackResult :   Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setPostResourceListener (IOCResourcePostCallback listener)

Callback for this API :

- public void onPostResponseCallback (Vector<OCHeaderOption>headerOptions,
  OCRepresentation rep, int errorValue)

Note that for registering a callback function for the requested actions, setPostResourceListener() is used.

executeActionSet() with delay parameter is used for execute an action set *after* a given delay in seconds. The first and second parameters are same as those of the above function. The third parameter is a *delay* which is a time delay used to postpone an execution of action set. The unit of this delay is a second. Note that even if you did not create the group action as a scheduled group action, the group action will be executed after the delay by using this parameter.

Prototype:

- OCStackResult executeActionSet(OCResource resource, String actionsetName, long delay)

  OCStackResult :   Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setPostResourceListener (IOCResourcePostCallback listener)

Callback for this API :

- public void onPostResponseCallback (Vector<OCHeaderOption>headerOptions,
  OCRepresentation rep, int errorValue)

CancelActionSet() is to request Things Manager to cancel the execution of action set.

Prototype:

- OCStackResult cancelActionSet(OCResource resource, String actionsetName)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setPostResourceListener (IOCResourcePostCallback listener)

Callback for this API :

- public void onPostResponseCallback (Vector<OCHeaderOption>headerOptions,

  OCRepresentation rep, int errorValue)

getActionSet() is to request Things Manager to get an existing action set belonging to a specific resource. The first parameter is a *resource* which is a target resource to be retrieved. The second parameter is *actionsetName* which is a string indicating an action set name. When the callback function is called, an action set string in Delimiter format will be returned.

Prototype:

- OCStackResult getActionSet(OCResource resource, String actionsetName)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setGetResourceListener (IOCResourceGetCallback listener)

Callback for this API :

- public void onGetResponseCallback (Vector<OCHeaderOption>headerOptions,

  OCRepresentation rep, int errorValue)

Note that for registering a callback function for the requested actions, setGetResourceListener() is used.

deleteActionSet() is to request a Things Manager to delete an existing action set belonging to a specific resource. The first parameter is a *resource* which is a target resource. The second parameter is an *actionsetName* which is a string indicating an action set name to be deleted.

Prototype:

- OCStackResult deleteActionSet(OCResource resource, String actionsetName)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setGetResourceListener (IOCResourceGetCallback listener)

Callback for this API :

- public void onGetResponseCallback (Vector<OCHeaderOption>headerOptions,

Note that for registering a callback function for the requested actions, setGetResourceListener() is used.

bindResourceToGroup() is to request a Things Manager to bind a found resource to group. The first parameter is a *resource* which is the found resource in the network. The second parameter is a *collectionHandle* which is a resource handle representing a group resource which the found resource is about to join to. Resource handle of resource which bound to given group/collection will be returned upon API success otherwise null.

Prototype:

- OCResourceHandle bindResourceToGroup(OCResource resource, OCResourceHandle collectionHandle)

  OCResourceHandle : Resource handle which is bound to given collection.

In addition to Group Manager APIs, ActionSet class provides two useful APIs to action set information translation. The action set information can be presented in two forms: ActionSet class form and a string form in Delimiter format.

toString() is to request ActionSet class to provide a string serialized in Delimiter format.

Prototype:

- String toString()

toActionSet() is to request ActionSet to translate a string serialized in Delimiter format into an instance of ActionSet class and get the instance. The only parameter is an *actionsetString* which is a target string serialized in Delimiter format.

Prototype:

- ActionSet toActionSet(String actionsetString)

4.2 THINGS CONFIGURATION & DIAGNOSTICS API

4.2.1 Things Configuration

There are two main usages of this class:

- On the **server** side, bootstrapping requisite information (i.e. system configuration parameters) from a bootstrap server to access other IoT services.
- On the **client** side, getting/updating the system configuration parameters from/to multiple remote things.

On the server side, there is only one API for **bootstrapping** :

Prototype:

- OCStackResult doBootstrap()

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setConfigurationListener(IDiagnosticsListener listener)

Callback for this API :

- public void onBootStrapCallback(Vector<OCHeaderOption> headerOptions,

Note: Listeners are provided to receive responses asynchronously, user can register using **Set listener** methods provided by SDK.

Note that for registering a callback function to receive asynchronous response for configuration feature APIs, setConfigurationListener() is used.

On operating at first stage, a resource server should fetch a bunch of configuration information from a bootstrap server to configure itself to access other IoT services. The information includes a system time information(e.g., time zones), network information (e.g., IP Address), and security information (i.e., access control list). After fetching, the information has been stored in forms of resources, namely Configuration resource.

On the client side, there are two APIs for getting/updating a resource value from/to resource server(s) : updateConfiguration() and getConfiguration()

Prototypes:

- OCStackResult updateConfigurations(OCResource resource,
  Map<String, String> configurations)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setConfigurationListener(IDiagnosticsListener listener)

Callback for this API :

- public void onUpdateConfigurationsCallback(Vector<OCHeaderOption> headerOptions,
  OCRepresentation rep, int errorValue)

Prototypes:

- OCStackResult getConfigurations(OCResource resource, Vector<String> configurations)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setConfigurationListener(IDiagnosticsListener listener)

Callback for this API :

- public void onGetConfigurationsCallback(Vector<OCHeaderOption> headerOptions,
  OCRepresentation rep, int errorValue)

Note that for registering a callback function to receive asynchronous response for configuration feature APIs, setConfigurationListener() is used.

The first parameter, *resource*, is a pointer of resource instance of Configuration resource. The resource pointer can be acquired by performing findResource() function with a dedicated resource type, "**oic.con**". Note that, the resource pointer represents not only a single simple resource but also a collection resource composing multiple simple resources. In other words, using these APIs, developers can send a series of requests to multiple things by calling the corresponding function at once.

The second parameter, *configurations*, represents an indicator of which resource and attribute key developers want to access and which value developers want to update. For convenience to access the wanted value, *ConfigurationName* in *configurations* parameter should be an attribute key of configuration resource, e.g., loc(Location), st(System Time), c(Currency), and r(Region) attribute keys. And *ConfigurationValue* only used in updateConfiguration() function indicates a value to be updated. Note that, only one configuration parameter is supported in this release. Multiple configuration parameters will be supported in future release.

As mentioned briefly, to access a resource with the above APIs, developers should know a ConfigurationName replacing a resource URI. Things Configuration provides an additional APIs for this;

Prototype:

- String getListOfSupportedConfigurationUnits(void)

When calling this function, developers get to know which *Configuration-Names* are supported and their brief descriptions. This information is provided in JSON format.

4.2.2 Resource Model of Things Configuration

Configuration resource currently has several attributes: location, system time, currency, and region. The detail descriptions for those attributes are followed below.

Resource Name	URI	Resource Type
Configuration	/oic/con	oic.con

Attribute	Attribute Name (key)	Value Type	Access Modes	Description
Location	loc	Json	R, W	Provides location information where available
System Time	st	Datetime	R	Reference time for the device
Currency	c	String	R, W	Indicate the currency that is used for any monetary transactions
Region	r	String	R, W	Indicate the current region in which the device is located geographically

4.2.3 Things Diagnostics

There are two functionalities in Things Diagnostics:(1) FactoryReset to restore all configuration parameters to default one, and (2) Reboot to request a system rebooting.

The prototypes of APIs to provide these functionalities are as following;

Prototype:

- OCStackResult factoryReset(OCResource resource)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setDiagnosticsListener(IDiagnosticsListener listener)

Callback for this API :

- public void onFactoryResetCallback(Vector<OCHeaderOption> headerOptions,

Prototype

- OCStackResult reboot(OCResource resource)

  OCStackResult : Returns OC_STACK_OK on Success , Error status code on Failure.

Register a callback for this API:

- public void setDiagnosticsListener(IDiagnosticsListener listener)

Callback for this API :

- public void onRebootCallback(Vector<OCHeaderOption> headerOptions,

Note that for registering a callback function to receive asynchronous response for diagnostics feature APIs, setDiagnosticsListener() is used.

The first function, factoryReset(), is used to restore all configuration parameters to default one. All configuration parameters refers Configuration resource, which they could have been modified for various reasons (e.g., for a request to update a value). If developers on the client want to restore the parameters, just use the factoryReset() function.

Additionally, for the purpose of storing default configuration parameters on a resource server side, the server maintains a FactorySet resource which stores all default configuration parameters. If the server receives a request of FactoryReset() from the client, the server refers a *FactorySet* resource to restore all parameters of Configuration resource. Note that, a client can access a *FactorySet* resource to read default parameters but cannot modify any parameter of the resource.

The second function, reboot(), is used to send a request to a server to be rebooted. On receiving the request, the server attempts to reboot itself in a deterministic time.

4.2.4 Resource Model of Things Diagnostics and FactorySet

Diagnostics resource currently has three attributes: factory reset, reboot, and start stat collection. The detail descriptions for those attributes are followed below.

Resource Name	URI	Resource Type
Diagnostics	/oic/diag	oic.diag

Attribute	Attribute Name (key)	Value Type	Access Modes	Description
Factory_Reset	fr	bolean	R, W	0 – No action (Default) 1 – Start Factory Reset. After factory reset, this value shall be changed back to the default value
Reboot	rb	boolean	R, W	0 – No action (Default) 1 – Start Reboot, After reboot, this value shall be changed back to the default value
StartStatCollection	ssc	boolean	R, W	0 – No collection of statistics 1 – Start collection statistics. Toggles between collecting and not collecting any device statistics depending on the value being 0 or 1.

Diagnostics resource has three simple resources: Factory Reset, Reboot, and Start Collection. The Factory Reset and Reboot resources are responsible for FactoryReset and Reboot functionalities in Things Diagnostics, respectively. The other resource, Start Collection, is to start collecting device-related statistics itself when the resource's value is set to *TRUE.*

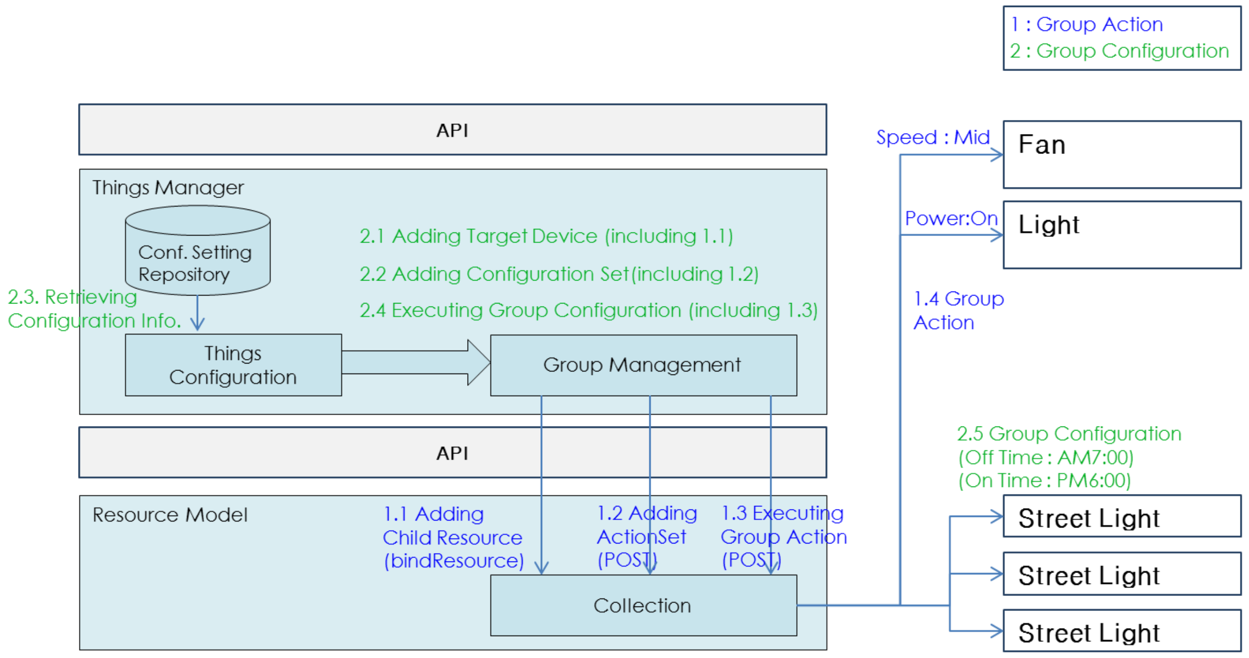
FactorySet resource is identical to Configuration resource except URI and resource type. It is only used to store default attribute values of Configuration resource which would be used to restore all values of Configuration resource to be default values. Thus, a list of attributes maintained in FactorySet resource is same as one in Configuration resource.

Resource Name	URI	Resource Type
FactorySet	/factorySet	factoryset

5 THINGS MANAGER ARCHITECTURE

5.1 CONTEXT DIAGRAM

Things Manager service operates in the IoTivity Base messaging environment as shown in the Figure 5.



1 : Group Action
2 : Group Configuration
API
Things Manager
Conf. Setting Repository
2.1 Adding Target Device (including 1.1)
2.2 Adding Configuration Set(including 1.2)
2.4 Executing Group Configuration (including 1.3)
2.3. Retrieving Configuration Info.
Things Configuration
Group Management
API
Resource Model
1.1 Adding Child Resource (bindResource)
1.2 Adding ActionSet (POST)
1.3 Executing Group Action (POST)
Collection
Speed : Mid
Fan
Power:On
Light
1.4 Group Action
2.5 Group Configuration (Off Time : AM7:00) (On Time : PM6:00)
Street Light
Street Light
Street Light

Figure 5. Things Manager Context Diagram

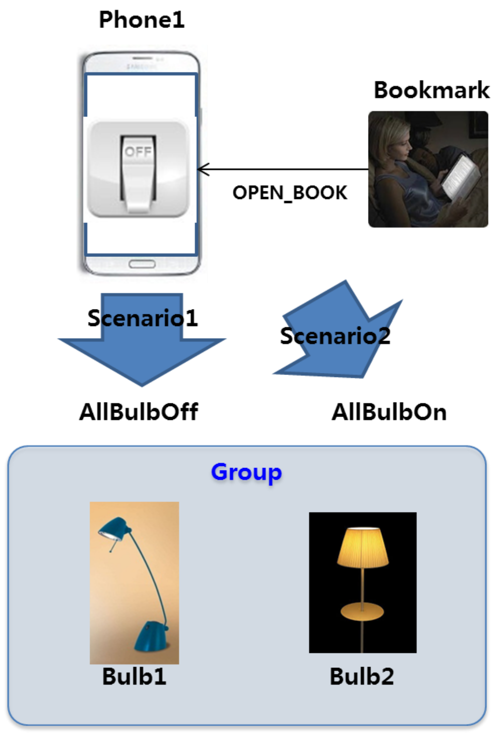
6 EXAMPLE : GROUP FORMATION & GROUP ACTION

This section is to show the functionality of group management and group action. With example, it can be understood how group can be made and how can we use it.

A Group can have homogeneous or heterogeneous member things. Multiple groups can be handled simultaneously.

Group Action can be triggered by user's action (i.e. switch control) or by registered action. User application can subscribe this kind of action to particular group. Example Scenario:

6.1 EXAMPLE SCENARIO:



Role		Scenario1	Scenario2
Actuator	Bulb1	On->Off	Off->On
	Bulb2	On->Off	Off->On
Sensor	Bookmark	N/A	Open
Command UI & TM	Phone1	Things Graph Manager working	
		Switch On->Off	N/A

Figure 6. Example Scenario for testing Group Formation/Action Feature

- Scenario 1: Bulb Control Service

There are mainly two entities composing "Bulb Control System": one mobile phone and two bulbs. This scenario shows that users can easily manage their at-home light bulbs with their mobile phone. One of convincing situations for the scenario is the moment that users leave home for work; they may want to turn all bulbs at home off with their phone.

a) At initial stage, we assume that two bulbs are already on.
b) After an application to control the bulbs executed on the phone, the phone discovers the bulbs around it and sets them as "Group" for Bulb Control Service.
c) After "Group" made, user can see a switch to control.
d) Once clicking the switch to off, the bulbs are going to turning off.

- Scenario 2: Help Your Reading Service (using bookmark)

a) At initial stage, we assume that the bulbs are already off and the bookmark is put in a book.

b) The phone discovers the bookmark and bulbs. First, when all bulbs are found, “Group” can be made. And when the bookmark is found, the phone requests the bookmark to observe a book’s openness. After that, the bookmark will notice its status whenever it changes. (open→close, close→open)

c) A book reader opens his/her book.

d) Then the bulbs are going to turn.

6.2 WORKING FLOW

This section introduces an experimental example of the scenarios where light bulbs and bookmark run on Ubuntu platform and the ***TM sample app*** on Android Mobile phone. This section describes how Things (i.e. bulbs and bookmark) can make group communication and what kind of features can be provided by Things Manager, specifically Group Manager.

6.2.1 Run all light bulb applications

First of all, users execute light bulb applications corresponding to actuators (called resources from here).

To execute all light bulb applications, enter as follows:

```
~/deviceB/service/things-manager/build/linux/release$ ./lightserver

Resource URI : /a/light

     Resource Type Name : core.light

     Resource Interface : oc.mi.def

     Resource creation is successful with resource handle : 0x1dd1de0

…

~/deviceC/service/things-manager/build/linux/release$ ./lightserver

…
```

As seeing above, users can know details of the registration for the light bulb resource including resource URI, resource type, and resource interface.

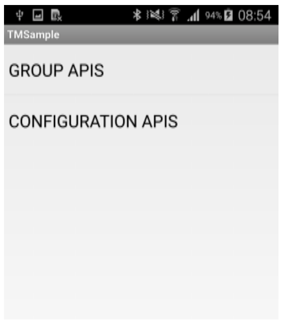
Next, users execute a bookmark application. To do, enter as follows:

```
~/deviceD/service/things-manager/build/linux/release$ ./bookmark

Resource URI : /core/bookmark

     Resource Type Name : core.bookmark

     Resource Interface : oc.mi.def

     Resource creation is successful with resource handle : 0xfd6da0
```

6.2.2 Run the TM sample app to control a group

Now, users execute **Android** application, called a "*TM sample app*". Once the application is executed, it shows two options: **GROUP APIS** & **CONFIGURATION APIS**.

MAIN MENU SCREEN:

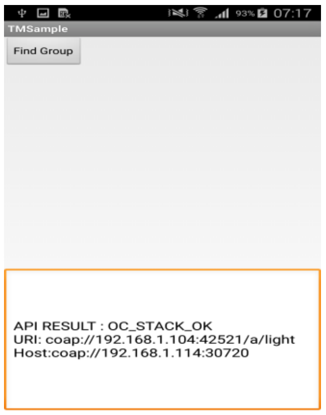


- **GROUP APIS:** For verifying Group Manager APIs.
- **CONFIGURATION APIS:** For verifying Things Configuration and Diagnostic APIs.

Once you select the first option the below screen will be shown:

If resource is running in background, then resource information will be displayed as shown in the below screenshot

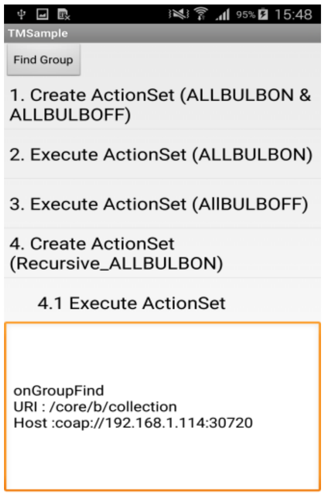
GROUP APIS SCREEN:



```
API RESULT : OC_STACK_OK
URI: coap://192.168.1.104:42521/a/light
Host:coap://192.168.1.114:30720
```

- **Find Group:** Clicking Find Group button invokes findGroup API for the resource of the type "b.collection"

After that, click **Find Group** button to find the group resource. Once we find the group resource, a list of actions will be displayed which can be performed. Group comprising the found light resources. Group information will be displayed as shown in the below screenshot

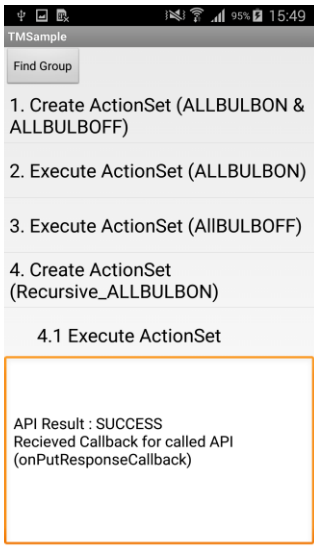

TMSample
Find Group
1. Create ActionSet (ALLBULBON & ALLBULBOFF)
2. Execute ActionSet (ALLBULBON)
3. Execute ActionSet (AllBULBOFF)
4. Create ActionSet (Recursive_ALLBULBON)
4.1 Execute ActionSet

```
onGroupFind
URI : /core/b/collection
Host :coap://192.168.1.114:30720
```

6.2.3 Create group action set,"AllBulbsOn" and "AllBulbsOff"

The next step is to define group action sets for the group. In this example, there are two predefined group action sets: "AllBulbsOn" and "AllBulbsOff".

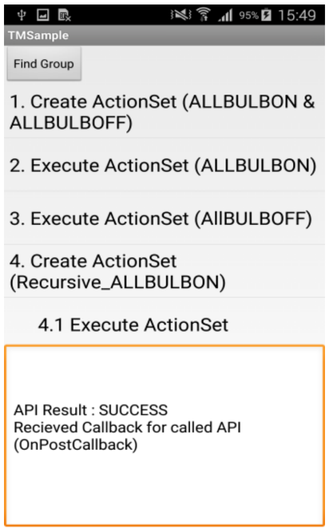
To create action set "AllBulbsOn" & "ALLBULBOFF" click on the 1st option in the app


TMSample
Find Group
1. Create ActionSet (ALLBULBON & ALLBULBOFF)
2. Execute ActionSet (ALLBULBON)
3. Execute ActionSet (AllBULBOFF)
4. Create ActionSet (Recursive_ALLBULBON)
4.1 Execute ActionSet

```
API Result : SUCCESS
Recieved Callback for called API
(onPutResponseCallback)
```

6.2.4 Execute a group action

To execute action set "AllBulbsOn", click on the 2nd option in the App.

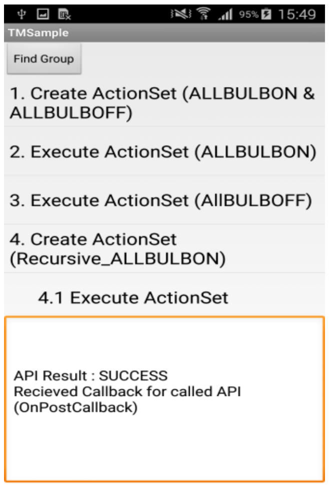

TMSample
Find Group
1. Create ActionSet (ALLBULBON & ALLBULBOFF)
2. Execute ActionSet (ALLBULBON)
3. Execute ActionSet (AllBULBOFF)
4. Create ActionSet (Recursive_ALLBULBON)
4.1 Execute ActionSet

```
API Result : SUCCESS
Recieved Callback for called API
(OnPostCallback)
```

After that, users can notice that all light bulb applications have an "On" event as follows:

```
In execution of ".∕lightserver

…

In entity handler wrapper:

     In Server CPP entity handler:

             requestFlag : Request

                     requestType : PUT

                             power: on
```

To execute action set "AllBulbsOff", click on the 3rd option in the App.


TMSample
Find Group
1. Create ActionSet (ALLBULBON & ALLBULBOFF)
2. Execute ActionSet (ALLBULBON)
3. Execute ActionSet (AllBULBOFF)
4. Create ActionSet (Recursive_ALLBULBON)
4.1 Execute ActionSet

```
API Result : SUCCESS
Recieved Callback for called API
(OnPostCallback)
```

Then, users can notice that all light bulb applications have an "Off" event as follows:

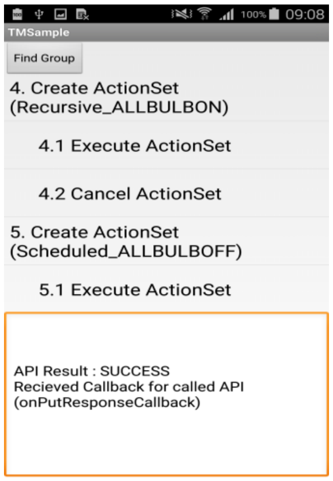
```
In execution of "./lightserver"

…

In entity handler wrapper:

     In Server CPP entity handler:

             requestFlag : Request

                     requestType : PUT

                             power: off
```

6.2.5 Create, Execute and Cancel ActionSet (RECURSIVE_ALLBULBON)

User can create an action set that can be triggered recursively.

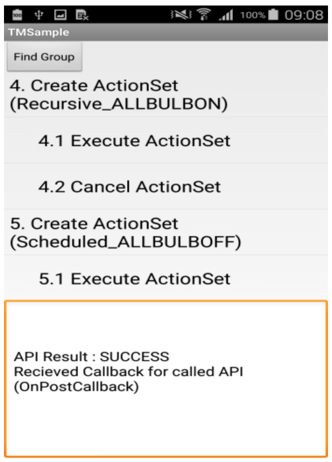
In sample application we are giving time duration of 5 seconds .

To create a recursive action set "ALLBULBON" click on the 4th option in the app.



```
API Result : SUCCESS
Recieved Callback for called API
(onPutResponseCallback)
```

To execute the recursive action set "ALLBULBON" click on the 4.1 option in the App.

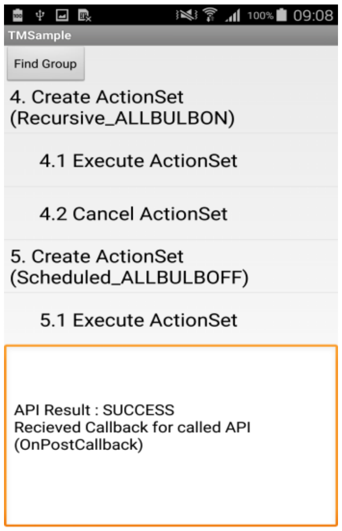


```
API Result : SUCCESS
Recieved Callback for called API
(OnPostCallback)
```

Then users can notice that all light bulb applications have an "On" event after each 5 seconds as follows: (until user cancel it by clicking 4.2 option in the app)

```
In execution of "./lightserver

…

In entity handler wrapper:

     In Server CPP entity handler:

             requestFlag : Request

                     requestType : PUT

                             power: on
```

To cancel the recursive action set "ALLBULBON" click on the 4.2 option in the App. It will stop the recursive action set "ALLBULBON".



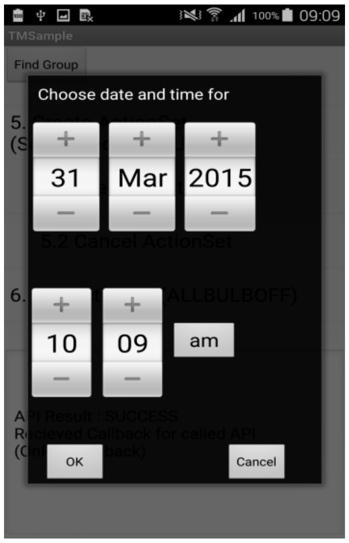
API Result : SUCCESS
Recieved Callback for called API
(OnPostCallback)

6.2.6 Create, Execute and Cancel ActionSet (SCHEDULED_ALLBULBOFF)

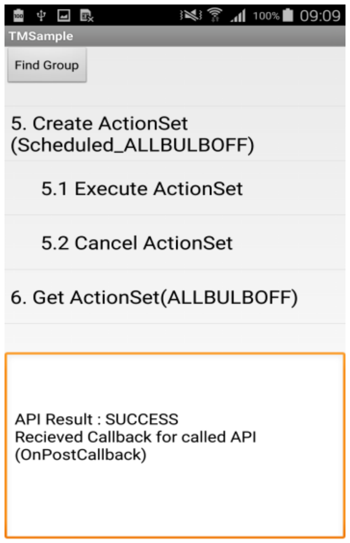
User can create an action set that can be triggered at the given date and time.

To create a scheduled action set "ALLBULBOFF" click on the 5th option in the app.

On clicking the 5th option the following dialog will be shown:



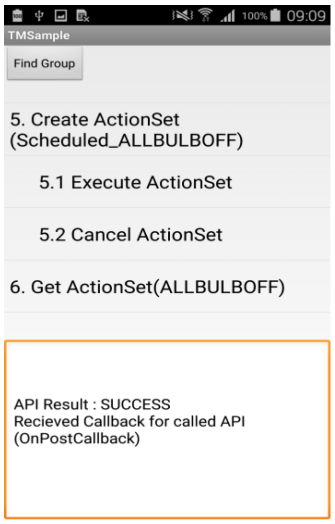
To execute scheduled action set "ALLBULBOFF" click on the 5.1 option in the App.



TMSample
Find Group
5. Create ActionSet (Scheduled_ALLBULBOFF)
5.1 Execute ActionSet
5.2 Cancel ActionSet
6. Get ActionSet(ALLBULBOFF)

API Result : SUCCESS
Recieved Callback for called API
(OnPostCallback)

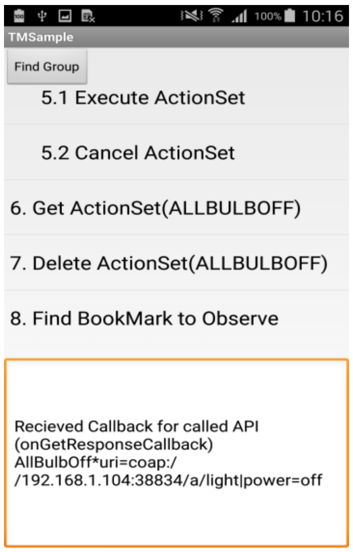
To cancel the scheduled action set “AllBulbsOFF” click on the 5.2 option in the App. It will cancel the scheduled action set that has been executed using 5.1 option.


TMSample
Find Group
5. Create ActionSet (Scheduled_ALLBULBOFF)
5.1 Execute ActionSet
5.2 Cancel ActionSet
6. Get ActionSet(ALLBULBOFF)

```
API Result : SUCCESS
Recieved Callback for called API
(OnPostCallback)
```

6.2.7 GetActionSet (ALLBULBOFF)

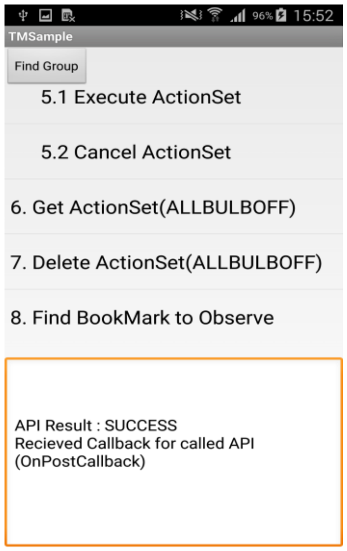
To get an actionset “ALLBULBOFF” that we have created using 1st option of the UI.


TMSample
Find Group
5.1 Execute ActionSet
5.2 Cancel ActionSet
6. Get ActionSet(ALLBULBOFF)
7. Delete ActionSet(ALLBULBOFF)
8. Find BookMark to Observe

```
Recieved Callback for called API
(onGetResponseCallback)
AllBulbOff*uri=coap:/
/192.168.1.104:38834/a/light|power=off
```

6.2.8 DeleteActionSet (ALLBULBOFF)

To delete an action set "ALLBULBOFF" that we have created earlier using 1st option of the UI.

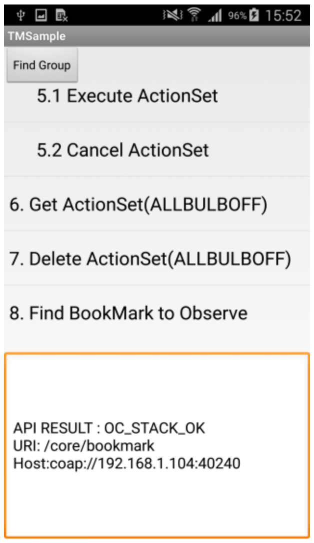

15:52
TMSample
Find Group
5.1 Execute ActionSet
5.2 Cancel ActionSet
6. Get ActionSet(ALLBULBOFF)
7. Delete ActionSet(ALLBULBOFF)
8. Find BookMark to Observe

```
API Result : SUCCESS
Recieved Callback for called API
(OnPostCallback)
```

6.2.9 Request an observe to a bookmark application

For the second scenario mentioned in Section7.1, the "TM sample app" needs to monitor how a bookmark's status changes. To do this, the "TM sample app" utilizes an observe option of CoAP specification.

To request an observe, click on the 8th option i.e. "Find Bookmark to Observe". Then the "TM sample app" discovers a bookmark resource in a network and then sends an observe request to the found resource server.


15:52
TMSample
Find Group
5.1 Execute ActionSet
5.2 Cancel ActionSet
6. Get ActionSet(ALLBULBOFF)
7. Delete ActionSet(ALLBULBOFF)
8. Find BookMark to Observe

```
API RESULT : OC_STACK_OK
URI: /core/bookmark
Host:coap://192.168.1.104:40240
```

If the observe request is successfully delivered to a bookmark resource server, users can see the below prompt.

```
In execution of "./bookmark"


Input a integer(0:opened, 5:close) :
…
```

6.2.10 Change a status of bookmark resource

If users want to change a status of the bookmark resource, simply put a digit "0" or "5" as follows:

```
In execution of "./bookmark"


Input a integer(0:opened, 5:close) : 0
…
```

Once the status changes, the bookmark application sends the notification to the groupserver application. If the notification informs that a book is opened, just execute an "AllBulbOn" action set and send a request for light bulb resource to turn on.

After that, users can notice that all light bulb applications have an "On" event as follows:

```
In execution of "./lightserver"
…
In entity handler wrapper:
     In Server CPP entity handler:
             requestFlag : Request
                     requestType : PUT
                             power: on
```

7 EXAMPLE : THINGS CONFIGURATION & DIAGNOSTICS

This section is to show the functionalities of things configuration and diagnostics. As discussed in Section 5, a things configuration is to get/update system configuration parameters and a things diagnostics is to request a set of specified functions with diagnostic purpose such as a factory reset and system reboot.

7.1 EXAMPLE SCENARIO

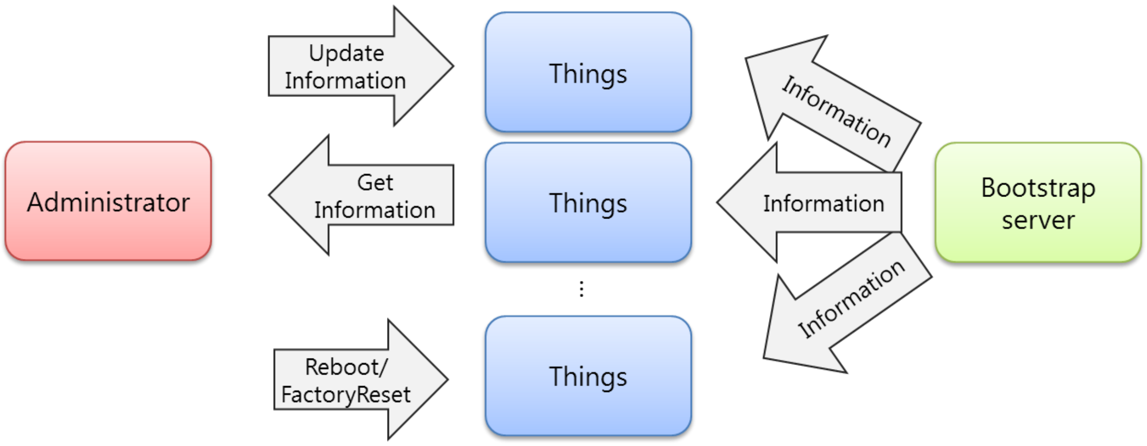


Figure 7. Example Scenario for testing Things Configuration/Diagnostics Feature

Scenario 1: Things Configuration (Get/Update)

We assume that there are two things(i.e. resource servers) to be managed by the administrator(i.e. a resource client). Additionally, we also assume that a simple bootstrap server is installed in purpose that things can fetch essential configuration parameters to access other IoT service. For E.g.: resource servers fetch information about the region and time from boot strap servers.

a) Bootstrap sever and resource server should be running
b) Resource server should get essential configuration parameters from bootstrap server
c) When resource client application starts, it can discover resource server
d) Resource client can get/update configuration parameter from resource server.

Scenario 2: Things Diagnostics (Factory Reset/Reboot)

This scenario shows two requests from an administrator to things: First is a factory reset and second is a system reboot. A factory reset is to restore all system configuration parameters to default one. And a system reboot is just to let the system reboot.

a) Bootstrap sever and resource server should be running
b) Resource server should get essential configuration parameters from bootstrap server
c) When resource client application starts, it can discover resource server

d) Resource client can send factory reset command to resource server to erase all the configuration parameter which got from bootstrap server

e) Resource client can send Reboot command to let the resource server to do system reboot

7.2 WORKING FLOW

This section introduces an experimental example of the scenarios run on Ubuntu platform & Android App.

7.2.1 Run all relevant applications: a bootstrapserver (Linux), con-server(Android) and *TM sample*(Android) *app* .

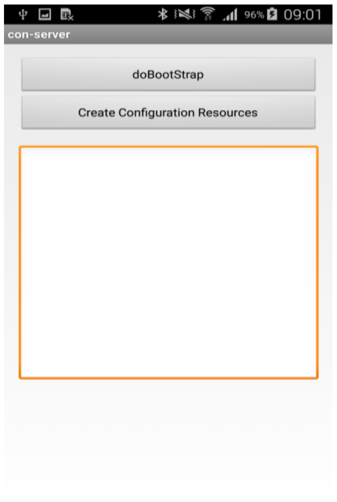
First of all, users execute all relevant applications. A bootstrapserver application represents a bootstrap server, a con-server application does a Thing, and a TM sample application does an administrator.

To execute the applications, enter as follows:

```
~/deviceA/service/things-manager/build/linux/release$ ./bootstrapserver
…
```

Now, run the con-server Android App

MAIN MENU SCREEN :

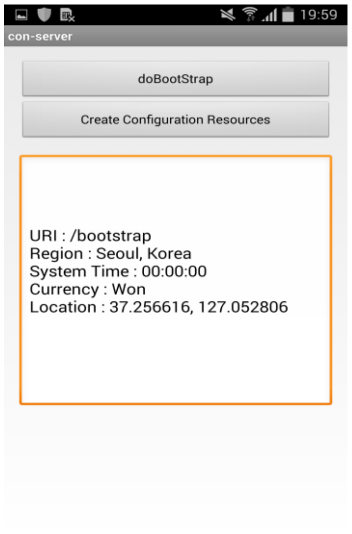


Two options are provided on the main menu :

- doBootStrap
- Create Configuration resources

7.2.2 doBootStrap :

This API is used by resource server, to receive configuration information from an available bootstrap server to configure it to access other IoT services. We receive configuration information in onBootStrapCallback, we are updating same information to UI as shown in the below figure.



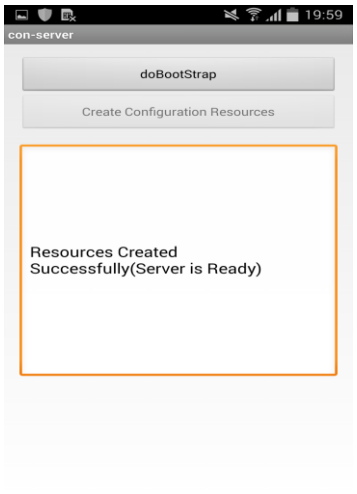
URI : /bootstrap
Region : Seoul, Korea
System Time : 00:00:00
Currency : Won
Location : 37.256616, 127.052806

7.2.2.1 Create Configuration resources:

Create configuration resources with the retrieved configuration parameters from a bootstrap server.

- Configuration resource
- Diagnostics resource
- FactorySetresource

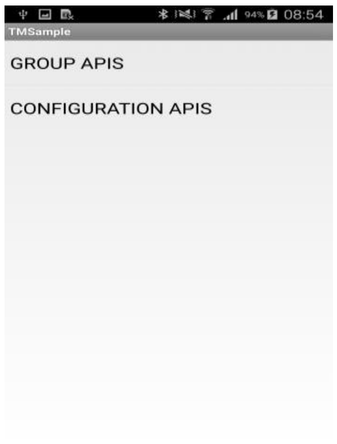
After successful creation of all the resources the UI will be updated as shown below



Resources Created Successfully(Server is Ready) we can test the Configuration APIs of Things Manager by running our sample app that has con-client in it i.e. CONFIGURATION APIS.

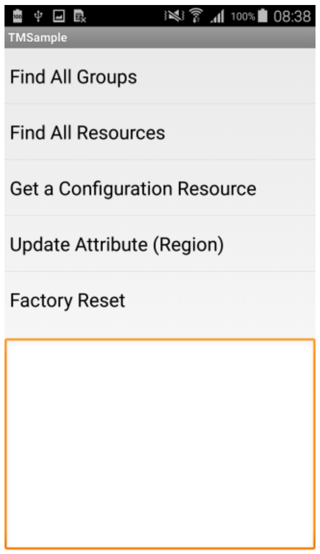
Now, Run the TM sample app

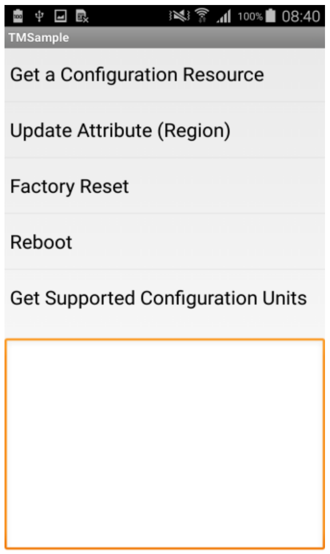
MAIN MENU SCREEN:



- ***CONFIGURATION APIS****:* Click on this menu to test Things configuration and Diagnostic APIs*.*

CONFIGURATION APIS SCREEN:

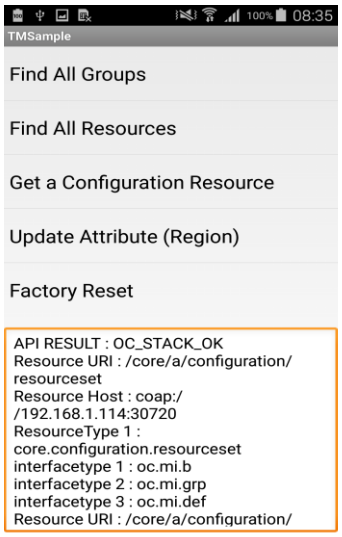



08:40
TMSample
Get a Configuration Resource
Update Attribute (Region)
Factory Reset
Reboot
Get Supported Configuration Units

Note that in the above snapshots, since in a single snapshot all options cannot be shown, two snapshots are captured to show all the options.

7.2.3 Find All Groups

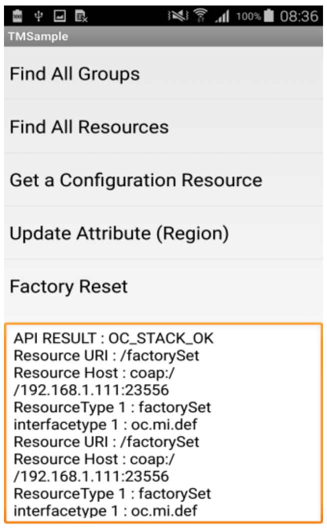
Search for the groups of the type core.configuration.resourceset, core.diagnostics.resourceset and core.factoryset.resourceset using API findCandidateResources.



```
API RESULT : OC_STACK_OK
Resource URI : /core/a/configuration/
resourceset
Resource Host : coap:/
/192.168.1.114:30720
ResourceType 1 :
core.configuration.resourceset
interfacetype 1 : oc.mi.b
interfacetype 2 : oc.mi.grp
interfacetype 3 : oc.mi.def
Resource URI : /core/a/configuration/
```

7.2.4 Find All Resources

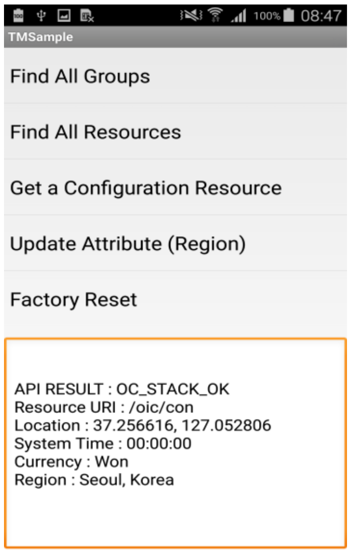
Search for the resources of the type oic.con, oic.diag and factorySet using API findCandidateResources.



```
API RESULT : OC_STACK_OK
Resource URI : /factorySet
Resource Host : coap:/
/192.168.1.111:23556
ResourceType 1 : factorySet
interfacetype 1 : oc.mi.def
Resource URI : /factorySet
Resource Host : coap:/
/192.168.1.111:23556
ResourceType 1 : factorySet
interfacetype 1 : oc.mi.def
```

7.2.5 Get a Configuration resource

Get configuration table from “oic.con” resource. We will get four attributes of configuration resource : Location, System Time, Currency and Region as shown in below screenshot.

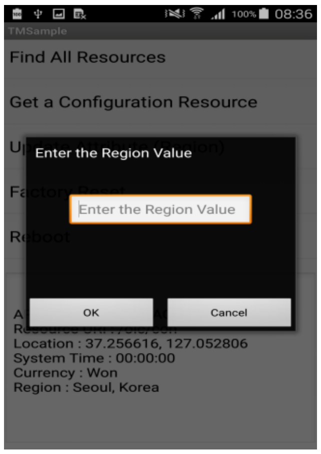


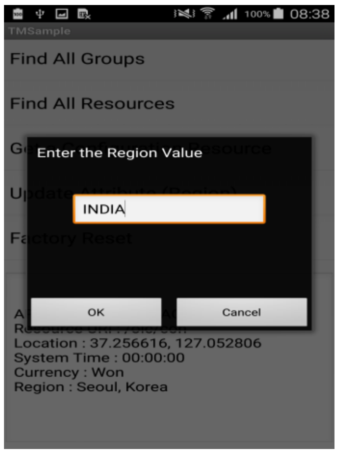
```
API RESULT : OC_STACK_OK
Resource URI : /oic/con
Location : 37.256616, 127.052806
System Time : 00:00:00
Currency : Won
Region : Seoul, Korea
```

7.2.6 Update Attribute (region)

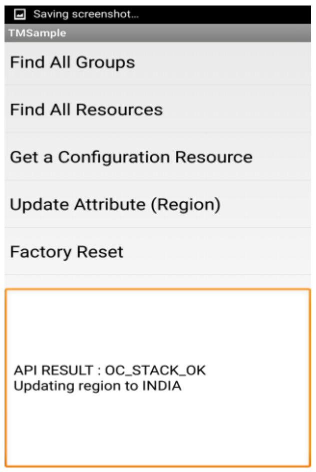
Set attribute **"region"** of oic.con resource with new value(India).

Once user select Update attribute(region) option in the app the following dialog will be shown



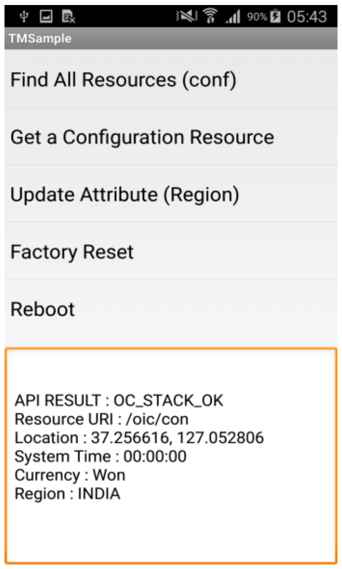


If user clicks ok then, the following screen will be shown



```
API RESULT : OC_STACK_OK
Updating region to INDIA
```

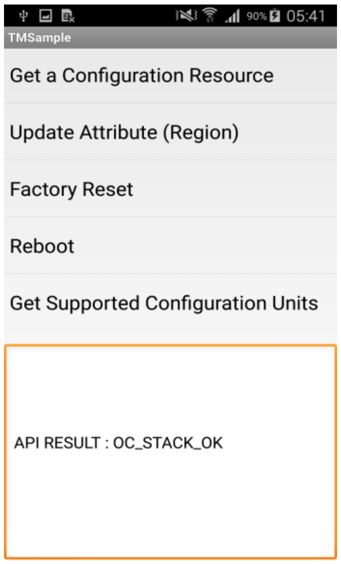
Now, if user clicks on the 3rd option “Get a Configuration resource”, user will see the region attributes has been set to the new value given by him.


90% 05:43
TMSample
Find All Resources (conf)
Get a Configuration Resource
Update Attribute (Region)
Factory Reset
Reboot

```
API RESULT : OC_STACK_OK
Resource URI : /oic/con
Location : 37.256616, 127.052806
System Time : 00:00:00
Currency : Won
Region : INDIA
```

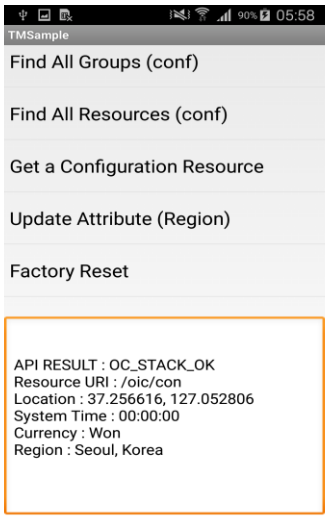
7.2.7 Factory Reset

Invoke factoryReset on “oic.diag” resource to restore all system configuration parameters to default one.


TMSample
Get a Configuration Resource
Update Attribute (Region)
Factory Reset
Reboot
Get Supported Configuration Units

```
API RESULT : OC_STACK_OK
```

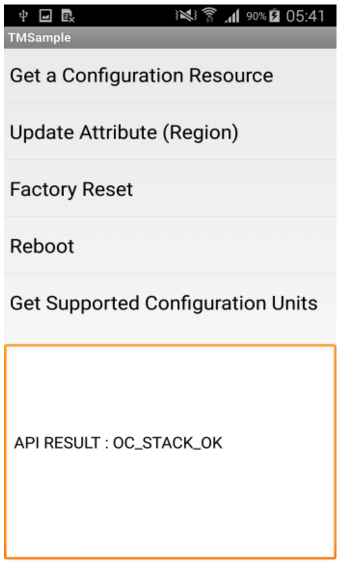
Note that now, If you press Get Configuration (region) option, you can see that it’s been reset to default value:


TMSample
Find All Groups (conf)
Find All Resources (conf)
Get a Configuration Resource
Update Attribute (Region)
Factory Reset

```
API RESULT : OC_STACK_OK
Resource URI : /oic/con
Location : 37.256616, 127.052806
System Time : 00:00:00
Currency : Won
Region : Seoul, Korea
```

7.2.8 Reboot

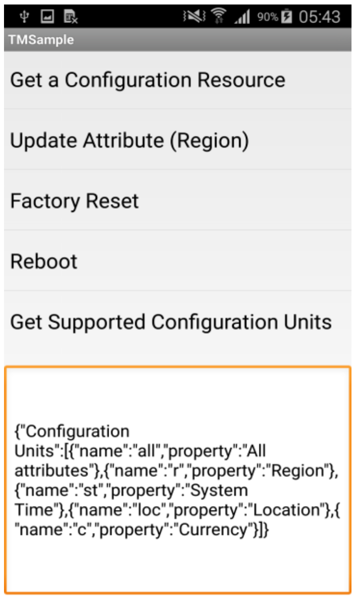
Invoke reboot on **"oic.diag"** resource to reboot the con-server.


05:41
TMSample
Get a Configuration Resource
Update Attribute (Region)
Factory Reset
Reboot
Get Supported Configuration Units

```
API RESULT : OC_STACK_OK
```

7.2.9 Get Supported Configuration Units

Get supported configuration unit in JSon format.


05:43
TMSample
Get a Configuration Resource
Update Attribute (Region)
Factory Reset
Reboot
Get Supported Configuration Units

```
{"Configuration
Units":[{"name":"all","property":"All
attributes"},{"name":"r","property":"Region"},
{"name":"st","property":"System
Time"},{"name":"loc","property":"Location"},{
"name":"c","property":"Currency"}]}
```