Tizen/Artik IoT Lecture Chapter 8. IoTivity P2P Connection

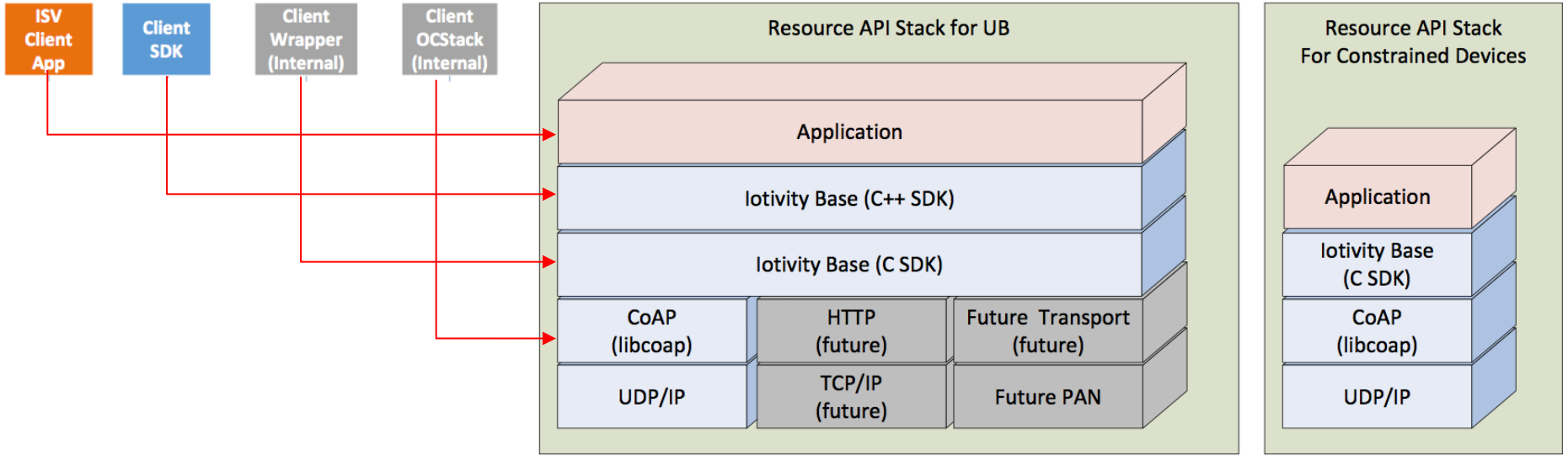
Sungkyunkwan University

Contents

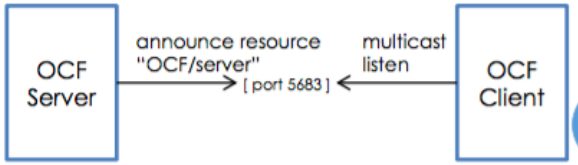
- **IoTivity API Set**
- **Discovery Subsystem**
  - Registering & Discovering a Resource
- **Querying Resource State**
  - GET
  - PUT
  - POST
- **Resource Observe**
- **SimpleClient & SimpleServer**

IoTivity API Set

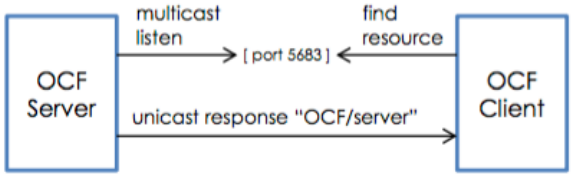
- **IoTivity supports various API set for platforms and languages**
  - Tizen, iOS, Android, Ubuntu, OS X, Windows (C/C++, Java, JS etc..)
  - There are two kinds of API stack : Rich-device and Constrained Devices



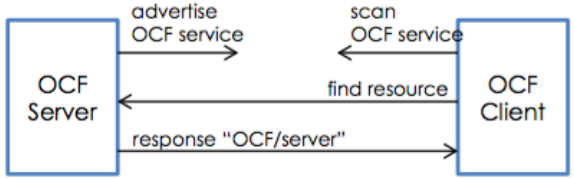
Discovery Subsystem



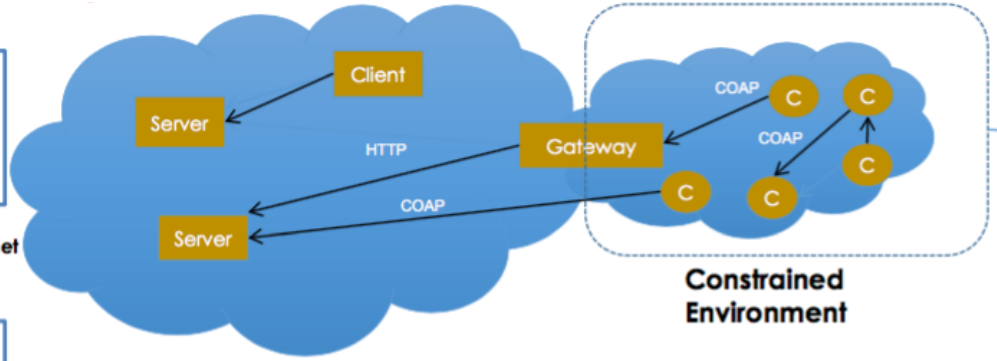
[Figure 1] Multicast announcement over Wi-Fi / Ethernet



[Figure 2] Multicast/Unicast over WiFi / Ethernet



[Figure 3] Advertise/Scan over BLE/BT



FOUNDATION™

Connectivity	Discovery Mechanism	Description
WiFi & Ethernet (over IP)	IP Multicast	CoAP Multicast Port: 5683 (Assigned by IANA) CoAP Secure Port: 5684
	IP Unicast over UDP	Precondition: OIC Server Address & Port are known
Bluetooth (EDR & BLE)	Using Scan & Advertise	OCF Specific Service UUID

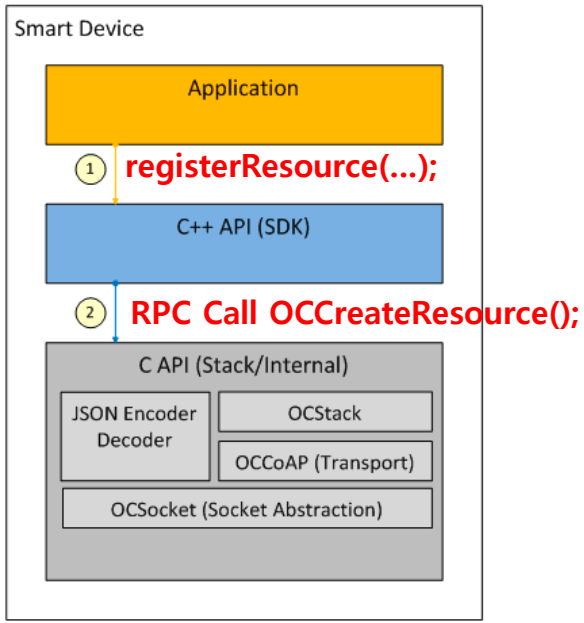
CoAP

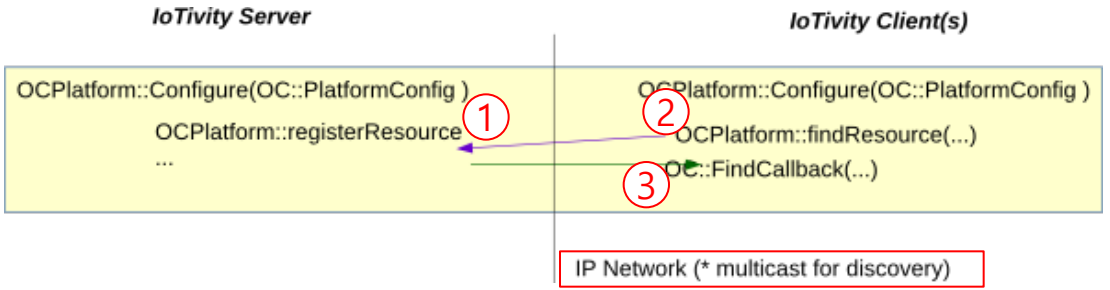
- Open IETF Standard (RFC 7252)
- Compact 4 Byte Header
- UDP (Default), SMS, TCP Support
- Strong DTLS Security
- Asynchronous Subscription
- Built-In Discovery

CoAP: Constrained Application Protocol
IANA: Internet Assigned Numbers Authority

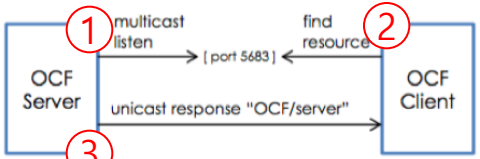
Registering and Discovering a Resource (1/3)

- **Resource Registration and Resource Finding are main functionalities in IoTivity Base Layer**





(1). Server: Registering Resource
(2). Client : Find Resource in IP Network (multicast)
(3). Client : When the client find resource, the callback function is invoked



[Figure 2] Multicast/Unicast over WiFi / Ethernet

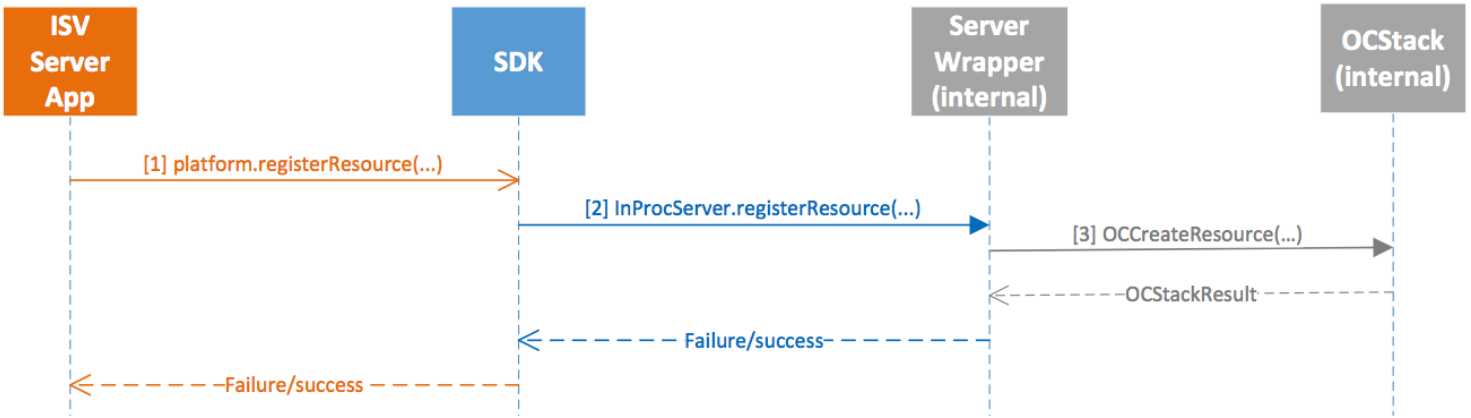
Registering and Discovering a Resource (2/3)

- **Registering a resource API**

```
OCStackResult OC::OCPlatform::registerResource(
    OCResourceHandle & resourceHandle,
    std::string & resourceURI,
    const std::string & resourceTypeName,
    const std::string & resourceInterface,
    EntityHandler entityHandler,
    uint8_t resourceProperty)
```

```
OCPlatform::registerResource(
    resourceHandle,
    "/light/1", "core.light", "oic.if.baseline",
    entityHandlerCb,
    OC_DISCOVERABLE | OC_OBSERVABLE);
```

<Example>



Registering and Discovering a Resource (3/3)

- **OCPlatform::findResource (const std::string& host**
  const std::string& resourceName
  OCConnectivityType connectivityType
  FindCallback resourceHandler)

CT_DEFAULT
CT_ADAPTER_IP
CT_ADAPTER_GATT_BTLE
CT_ADAPTER_RFCOMM_BTEDR
CT_ADAPTER_TCP
CT_ADAPTER_NFC
CT_FLAG_SECURE
CT_IP_USE_V6
CT_IP_USE_V4
CT_SCOPE_INTERFACE
CT_SCOPE_LINK
CT_SCOPE_REALM
CT_SCOPE_ADMIN
CT_SCOPE_SITE
CT_SCOPE_ORG
CT_SCOPE_GLOBAL

```
Example: simpleClient.cpp
...
                                  "/oic/res" = all OIC devices that are discoverable
requestURI << OC_RSRVD_WELL_KNOWN_URI;
                          host    Host IP Address of a service to direct resource discovery query. If null or empty, performs multicast resource discovery query
OCPlatform::findResource("", requestURI.str(),
          CT_DEFAULT, &foundResource);

...
```

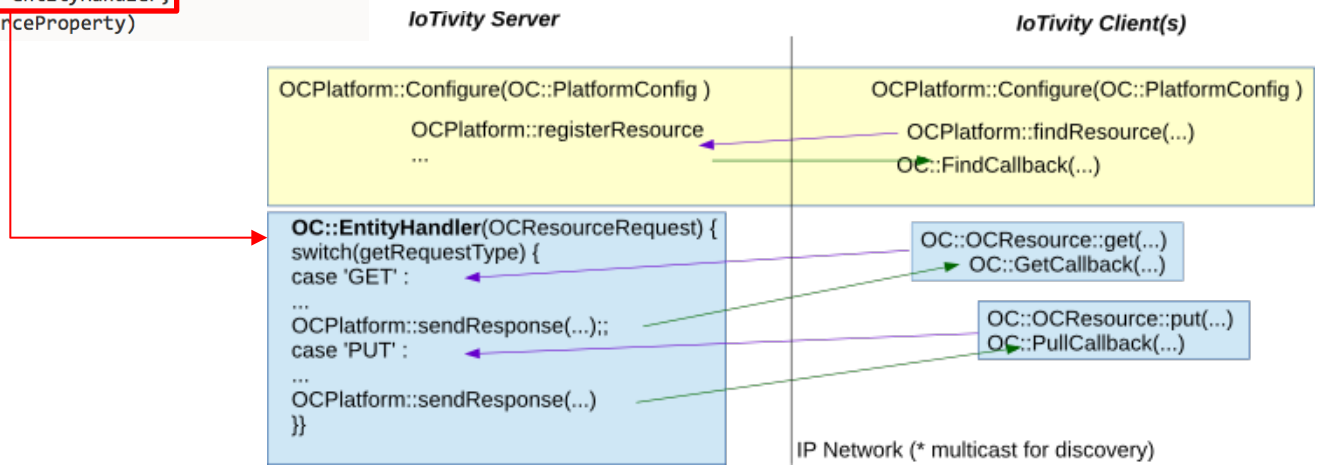
→ **API for Service and Resource discovery (client side only)**

host == null: multicast resource discovery query

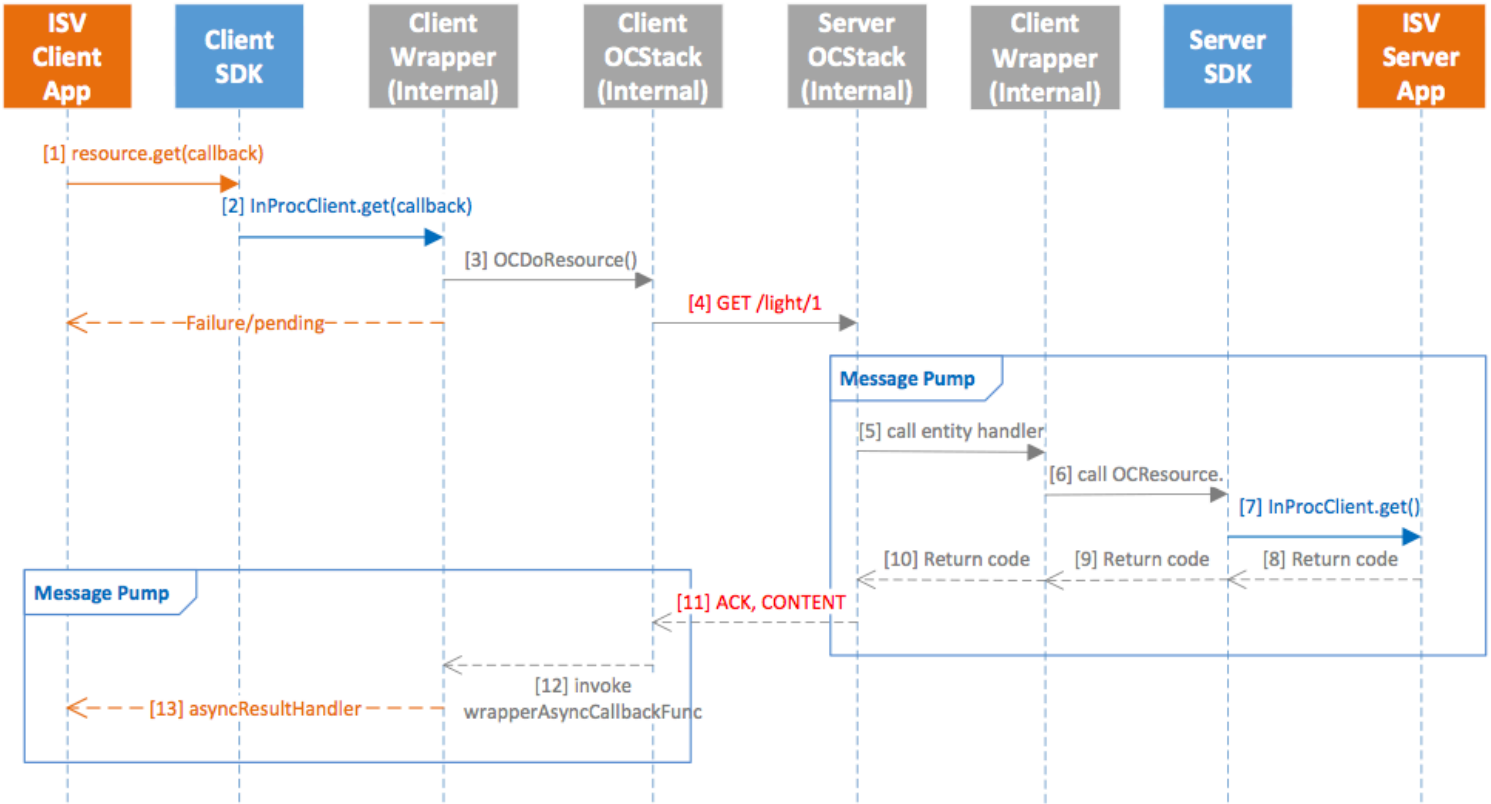
Querying Resource State

- **After registering and discovering states, the clients can invoke GET, PUT query**

```
OCStackResult OC::OCPlatform::registerResource(
    OCResourceHandle & resourceHandle,
    std::string & resourceURI,
    const std::string & resourceTypeName,
    const std::string & resourceInterface,
    EntityHandler entityHandler,
    uint8_t resourceProperty)
```



Querying Resource State: GET (1/2)



Querying Resource State: GET (2/2)


```
void getLightRepresentation(std::shared_ptr<OCResource> resource)
{
  if(resource)
  {
    std::cout << "Getting Light Representation..."<<std::endl;
    // Invoke resource's get API with the callback parameter
    QueryParamsMap test;
    resource->get(test, &onGet);
  }
}
```

①

```
// callback handler on GET request
void onGet(const OCRepresentation& rep, const int eCode)
{
  if(eCode == SUCCESS_RESPONSE)
  {
    std::cout << "GET request was successful" << std::endl;
    AttributeMap attributeMap = rep.getAttributeMap();
    std::cout << "Resource URI: " << rep.getUri() << std::endl;
    for(auto it = attributeMap.begin(); it != attributeMap.end(); ++it)
    {
      std::cout << "\tAttribute name: "<< it->first << " value: ";
      for(auto valueItr = it->second.begin(); valueItr != it->second.end(); ++valueItr)
      {
        std::cout <<"\t"<< *valueItr << " ";
      }
      std::cout << std::endl;
    }
```

Response

Unmarshaling

<Client>

②

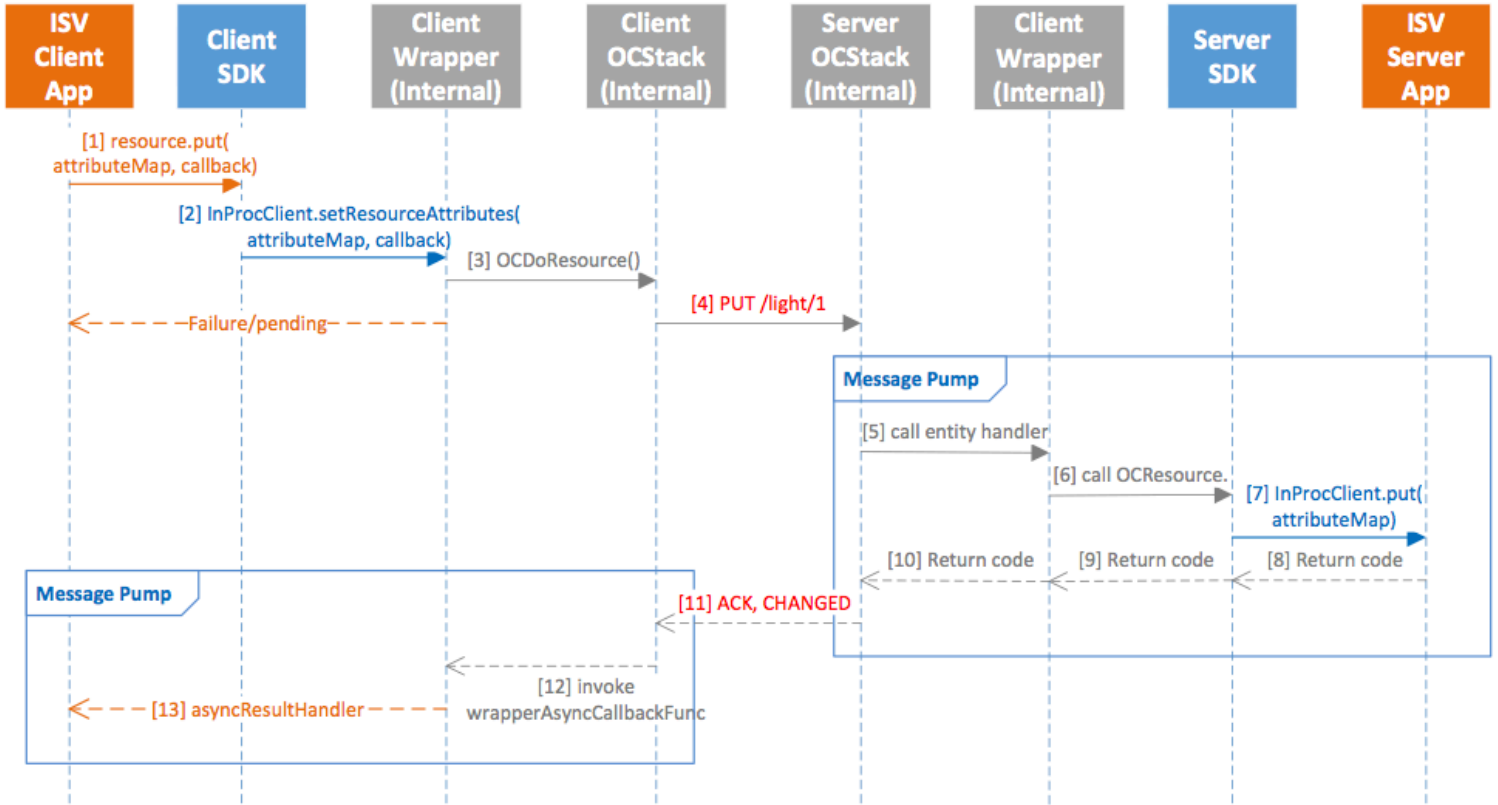
```
// Handling GET request in Entity handler
if(requestType == "GET")
  {
    cout << "\t\t\trequestType : GET\n";
    // Check for query params (if any)
    QueryParamsMap queryParamsMap = request->getQueryParameters();
    cout << "\t\t\tquery params: \n";
    for(QueryParamsMap::iterator it = queryParamsMap.begin(); it != queryParamsMap.end(); it++)
    {
      cout << "\t\t\t\t" << it->first << ":" << it->second << endl;
    }
    // Process query params and do required operations ..
    // Get the representation of this resource at this point and send it as response
    // AttributeMap attributeMap;
```

Query

Unmarshaling

<Server>

Querying Resource State: PUT (1/2)



Querying Resource State: PUT (2/2)


```
void putLightRepresentation(std::shared_ptr<OCResource> resource)
{
  if(resource)
  {
    OCRepresentation rep;

    std::cout << "Putting light representation..."<<std::endl;
    // Create AttributeMap
    AttributeMap attributeMap;
    // Add the attribute name and values in the attribute map
    AttributeValues stateVal;
    stateVal.push_back("true");

    AttributeValues powerVal;
    powerVal.push_back("10");

    attributeMap["state"] = stateVal;
    attributeMap["power"] = powerVal;

    // Create QueryParameters Map and add query params (if any)
    QueryParamsMap queryParamsMap;

    rep.setAttributeMap(attributeMap);

    // Invoke resource's put API with attribute map, query map and the callback parameter
    resource->put(rep, queryParamsMap, &onPut);
  }
}
```

Key:Value Packing

Put

<Client>

```
//Entity handle sample for PUT
  if(requestType == "PUT")
  {
    cout << "\t\t\trequestType : PUT\n";

    // Check for query params (if any)
    QueryParamsMap queryParamsMap = request->getQueryParameters();

    cout << "\t\t\tquery params: \n";
    for(auto it = queryParamsMap.begin(); it != queryParamsMap.end(); it++)
    {
      cout << "\t\t\t\t\t" << it->first << ":" << it->second << endl;
    }

    // Get the representation from the request
    OCRepresentation rep = request->getResourceRepresentation();

    myLightResource.setRepresentation(rep); // See code snippet below

    // Do related operations related to PUT request // See code snippet below
    rep = myLightResource.getRepresentation();
```

Query unpacking

```
void setRepresentation(OCRepresentation& light)
{
  AttributeMap attributeMap = light.getAttributeMap();

  if(attributeMap.find("state") != attributeMap.end() && attributeMap.find("power") != attributeMap.end())
  {
    cout << "\t\t\t" << "Received representation: " << endl;
    cout << "\t\t\t\t" << "power: " << attributeMap["power"][0] << endl;
    cout << "\t\t\t\t" << "state: " << attributeMap["state"][0] << endl;

    m_state = attributeMap["state"][0].compare("true") == 0;
    m_power= std::stoi(attributeMap["power"][0]);
  }
}
```

Resource Set

<Server>

Querying Resource State: POST (1/2)

- **POST can create new resource like PUT**

```
// Local function to put a different state for this resource
void postLightRepresentation(std::shared_ptr<OCResource> resource)
{
    if(resource)
    {
        OCRepresentation rep;

        std::cout << "Posting light representation..."<<std::endl;

        mylight.m_state = false;
        mylight.m_power = 105;

        rep.setValue("state", mylight.m_state);
        rep.setValue("power", mylight.m_power);

        // Invoke resource's post API with rep, query map and the callback parameter
        resource->post(rep, QueryParamsMap(), &onPost);
    }
}
```

1 Post Invocation

<Client>

```
else if(requestType == "POST")
{
    cout << "\t\t\trequestType : POST\n";

    OCRepresentation rep = request->getResourceRepresentation();

    // Do related operations related to POST request
    OCRepresentation rep_post = post(rep);
    pResponse->setResourceRepresentation(rep_post);
    pResponse->setErrorCode(200);
    if(rep_post.hasAttribute("createduri"))
    {
        pResponse->setResponseResult(OC_EH_RESOURCE_CREATED);
        pResponse->setNewResourceUri(rep_post.getValue<std::string>("createduri"));
    }
    else
    {
        pResponse->setResponseResult(OC_EH_OK);
    }
    if(OC_STACK_OK == OCPlatform::sendResponse(pResponse))
    {
        ehResult = OC_EH_OK;
    }
}
```

Attribute setting

2

Create New Resource

7

3

Send Response

<Server>

Querying Resource State: POST (2/2)


```
void onPost(const HeaderOptions& /*headerOptions*/,
        const OCRepresentation& rep, const int eCode)
{
    Check Resource Created
    4  if(eCode == OC_STACK_OK || eCode == OC_STACK_RESOURCE_CREATED)
       {
           std::cout << "POST request was successful" << std::endl;
    5      if(rep.hasAttribute("createduri"))   Attribute Check
           {
               std::cout << "\tUri of the created resource: "
                   << rep.getValue<std::string>("createduri") << std::endl;
           }
           else
           {
               rep.getValue("state", mylight.m_state);
               rep.getValue("power", mylight.m_power);
               rep.getValue("name", mylight.m_name);

               std::cout << "\tstate: " << mylight.m_state << std::endl;
               std::cout << "\tpower: " << mylight.m_power << std::endl;
               std::cout << "\tname: " << mylight.m_name << std::endl;
           }
    6      OCRepresentation rep2;

           std::cout << "Posting light representation..."<<std::endl;

           mylight.m_state = true;
           mylight.m_power = 55;

           rep2.setValue("state", mylight.m_state);
           rep2.setValue("power", mylight.m_power);

           curResource->post(rep2, QueryParamsMap(), &onPost2);
       }
```

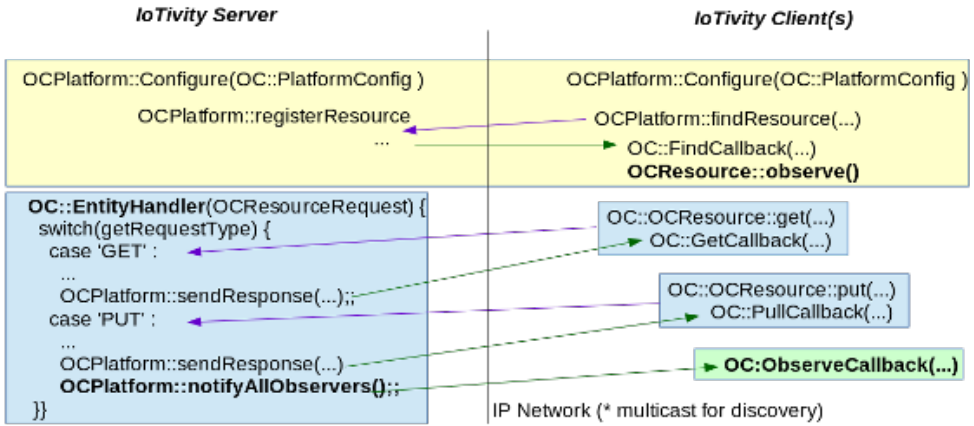
```
void onPost2(const HeaderOptions& /*headerOptions*/,
        const OCRepresentation& rep, const int eCode)
{
    8  if(eCode == OC_STACK_OK || eCode == OC_STACK_RESOURCE_CREATED)
       {
           std::cout << "POST request was successful" << std::endl;

           if(rep.hasAttribute("createduri"))
           {
               std::cout << "\tUri of the created resource: "
                   << rep.getValue<std::string>("createduri") << std::endl;
           }
           else
           {
    9          rep.getValue("state", mylight.m_state);
               rep.getValue("power", mylight.m_power);
               rep.getValue("name", mylight.m_name);

               std::cout << "\tstate: " << mylight.m_state << std::endl;
               std::cout << "\tpower: " << mylight.m_power << std::endl;
               std::cout << "\tname: " << mylight.m_name << std::endl;
           }

           if (OBSERVE_TYPE_TO_USE == ObserveType::Observe)
               std::cout << std::endl << "Observe is used." << std::endl << std::endl;
           else if (OBSERVE_TYPE_TO_USE == ObserveType::ObserveAll)
               std::cout << std::endl << "ObserveAll is used." << std::endl << std::endl;

           curResource->observe(OBSERVE_TYPE_TO_USE, QueryParamsMap(), &onObserve);

       }
```

<Client> New Attribute Setting

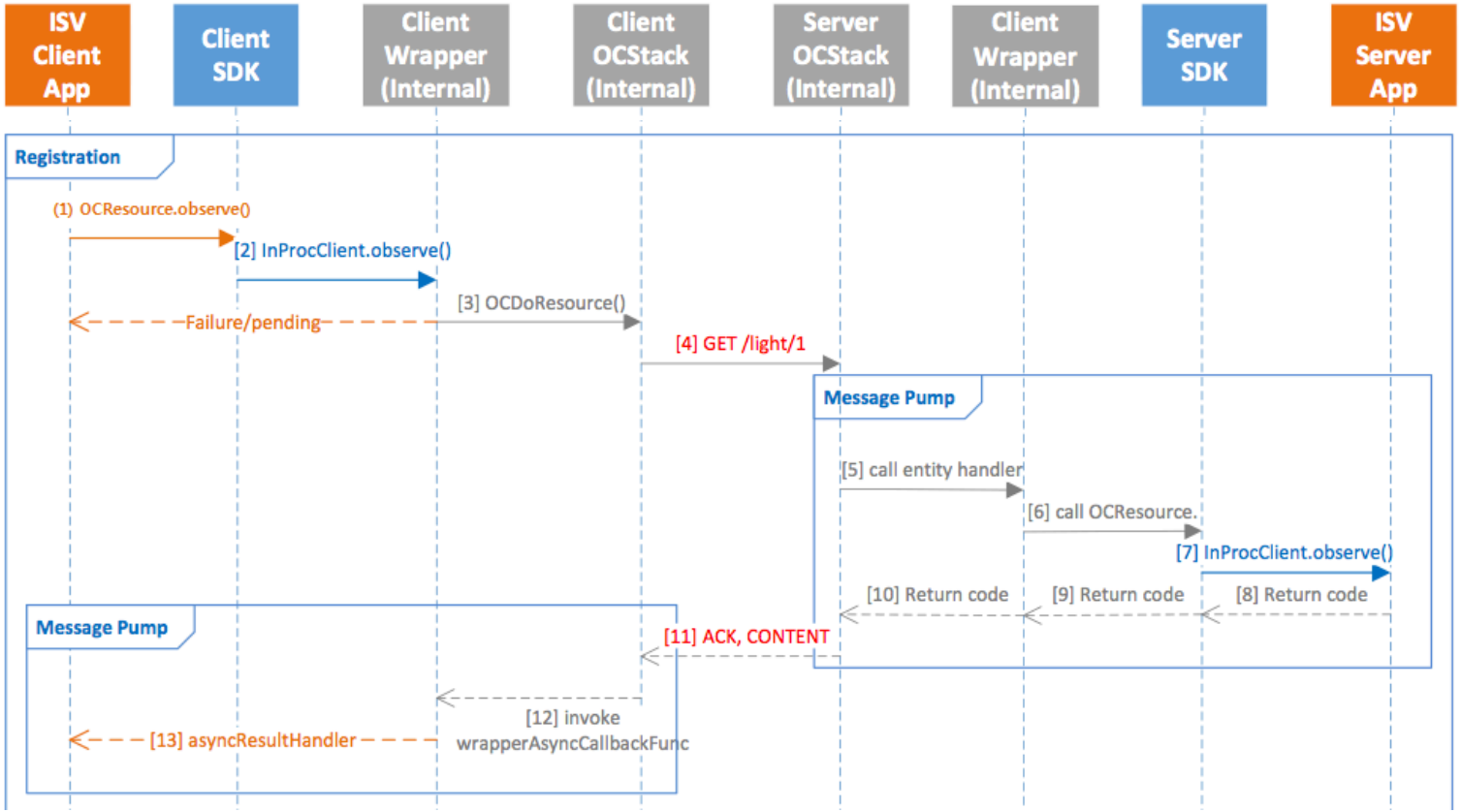
Resource Observe (1/5)

- **Observe happens at resource level and operates in server/client model**

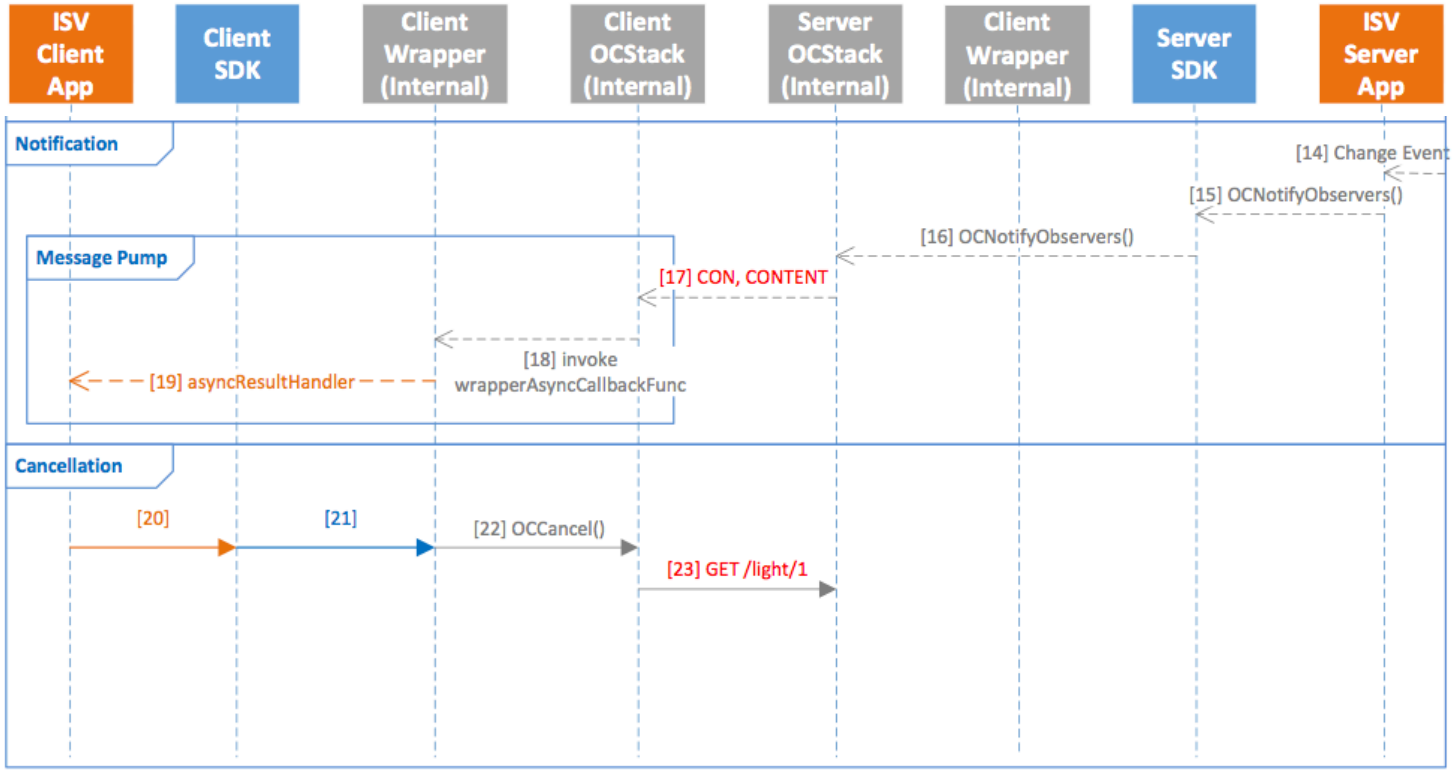
```
OCPlatform::registerResource(
    resourceHandle,
    "/light/1", "core.light", "oic.if.baseline",
    entityHandlerCb,
    OC_DISCOVERABLE | OC_OBSERVABLE);
```



Resource Observe (2/5): Sequence Flow



Resource Observe (3/5): Sequence Flow



Resource Observe (4/5): SimpleClient

- **Observe API**

```
OCStackResult OC::OCResource::observe ( ObserveType                observeType,
                                        const QueryParamsMap &     queryParametersMap,
                                        ObserveCallback            observeHandler
                                      )
```

```
void onPost2(const HeaderOptions& /*headerOptions*/,
        const OCRepresentation& rep, const int eCode)
{
    try
    {
        if(eCode == OC_STACK_OK || eCode == OC_STACK_RESOURCE_CREATED)
        {
            std::cout << "POST request was successful" << std::endl;

            if(rep.hasAttribute("createduri"))
            {
                std::cout << "\tUri of the created resource: "
                    << rep.getValue<std::string>("createduri") << std::endl;
            }
            else
            {
                rep.getValue("state", mylight.m_state);
                rep.getValue("power", mylight.m_power);
                rep.getValue("name", mylight.m_name);

                std::cout << "\tstate: " << mylight.m_state << std::endl;
                std::cout << "\tpower: " << mylight.m_power << std::endl;
                std::cout << "\tname: " << mylight.m_name << std::endl;
            }

            if (OBSERVE_TYPE_TO_USE == ObserveType::Observe)
                std::cout << std::endl << "Observe is used." << std::endl << std::endl;
            else if (OBSERVE_TYPE_TO_USE == ObserveType::ObserveAll)
                std::cout << std::endl << "ObserveAll is used." << std::endl << std::endl;

            curResource->observe(OBSERVE_TYPE_TO_USE, QueryParamsMap(), &onObserve);
        }
```

```
void onObserve(const HeaderOptions /*headerOptions*/, const OCRepresentation& rep,
               const int& eCode, const int& sequenceNumber)
{
    try
    {
        if(eCode == OC_STACK_OK && sequenceNumber != maxSequenceNumber + 1)
        {
            if(sequenceNumber == OC_OBSERVE_REGISTER)
            {
                std::cout << "Observe registration action is successful" << std::endl;
            }

            std::cout << "OBSERVE RESULT:"<<std::endl;
            std::cout << "\tSequenceNumber: "<< sequenceNumber << std::endl;
            rep.getValue("state", mylight.m_state);
            rep.getValue("power", mylight.m_power);
            rep.getValue("name", mylight.m_name);

            std::cout << "\tstate: " << mylight.m_state << std::endl;
            std::cout << "\tpower: " << mylight.m_power << std::endl;
            std::cout << "\tname: " << mylight.m_name << std::endl;

            if(observe_count() == 11)
            {
                std::cout<<"Cancelling Observe..."<<std::endl;
                OCStackResult result = curResource->cancelObserve();

                std::cout << "Cancel result: "<< result <<std::endl;
                sleep(10);
                std::cout << "DONE"<<std::endl;
                std::exit(0);
            }
        }
```

<Client>

Resource Observe (5/5): SimpleServer

- **Observe Registration and notify**

① ② Entity Handler check observe message & push observer into observer list

```
if(requestFlag & RequestHandlerFlag::ObserverFlag)
{
    ObservationInfo observationInfo = request->getObservationInfo();
    if(ObserveAction::ObserveRegister == observationInfo.action)
    {
        m_interestedObservers.push_back(observationInfo.obsId);
    }
    else if(ObserveAction::ObserveUnregister == observationInfo.action)
    {
        m_interestedObservers.erase(std::remove(
                                        m_interestedObservers.begin(),
                                        m_interestedObservers.end(),
                                        observationInfo.obsId),
                                        m_interestedObservers.end());
    }
```

③ Create Light Value changed Watcher Thread

```
    startedThread)
    pthread_create (&threadId, NULL, ChangeLightRepresentation, (void *)this);
    startedThread = 1;
}
```

```
OCStackResult OC::OCPlatform::notifyListOfObservers ( OCResourceHandle          resourceHandle,
                                                      ObservationIds &          observationIds,
                                                      const std::shared_ptr< OCResourceResponse > responsePtr
                                                    )
```

```
OCStackResult OC::OCPlatform::notifyAllObservers ( OCResourceHandle resourceHandle )
```

```
void * ChangeLightRepresentation (void *param)
{
    LightResource* lightPtr = (LightResource*) param;

    // This function continuously monitors for the changes
    while (1)
    {
        sleep (3);

        if (gObservation)
        {
            // If under observation if there are any changes to the light resource
            // we call notifyObservors
            //
            // For demostration we are changing the power value and notifying.
            lightPtr->m_power += 10;

            cout << "\nPower updated to : " << lightPtr->m_power << endl;
            cout << "Notifying observers with resource handle: " << lightPtr->getHandle() << endl;

            OCStackResult result = OC_STACK_OK;

            if(isListOfObservers) CheckObserverList
            {
                std::shared_ptr<OCResourceResponse> resourceResponse =
                            {std::make_shared<OCResourceResponse>()};

                resourceResponse->setErrorCode(200);
                resourceResponse->setResourceRepresentation(lightPtr->get(), DEFAULT_INTERFACE);

④               result = OCPlatform::notifyListOfObservers(  lightPtr->getHandle(),
                                                             lightPtr->m_interestedObservers,
                                                             resourceResponse);
            }
            else
            {
                result = OCPlatform::notifyAllObservers(lightPtr->getHandle());
            }

            if(OC_STACK_NO_OBSERVERS == result)
            {
                cout << "No More observers, stopping notifications" << endl;
                gObservation = 0;
            }
        }
    }

    return NULL;
}
```

Send Notify

SimpleClient (Android)

- **IoTivity provides SimpleServer and SimpleClient examples for C/C++ and Andorid**
  - SimpleClient and SimpleServer includes restful APIs such as get, put, post and observe

Platfrom Config

```
PlatformConfig platformConfig = new PlatformConfig(
        this,
        context,
        ServiceType.IN_PROC,
        ModeType.SERVER,
        "0.0.0.0", // By setting to "0.0.0.0", it binds to all available interfaces
        0,         // Uses randomly available port
        QualityOfService.LOW
);
```

```
public void createNewLightResource(String resourceUri, String resourceName){
    msg("Creating a light");
    Light light = new Light(
            resourceUri,      //URI
            resourceName,     //name
            false,            //state
            0                 //power
    );
    msg(light.toString());
    light.setContext(this);

    msg("Registering light as a resource");
    try {
        light.registerResource();
    } catch (OcException e) {
        Log.e(TAG, e.toString());
        msg("Failed to register a light resource");
    }
    lights.add(light);
}
```

Register Resource

1

<SimpleServer>

```
PlatformConfig platformConfig = new PlatformConfig(
        this,
        context,
        ServiceType.IN_PROC,
        ModeType.CLIENT,
        "0.0.0.0", // By setting to "0.0.0.0", it binds to all available interfaces
        0,         // Uses randomly available port
        QualityOfService.LOW
);
msg("Configuring platform.");
OcPlatform.Configure(platformConfig);
```

Platfrom Config

2

```
OcPlatform.findResource("",
        requestUri,
        EnumSet.of(OcConnectivityType.CT_DEFAULT),
        this
);
```

Find Resource

<SimpleClient>

CT_ADAPTER_GATT_BTLE	GATT over Bluetooth LE.
CT_ADAPTER_IP	IPv4 and IPv6, including 6LoWPAN.
CT_ADAPTER_NFC	NFC Transport.
CT_ADAPTER_REMOTE_ACCESS	Remote Access over XMPP.
CT_ADAPTER_RFCOMM_BTEDR	RFCOMM over Bluetooth EDR.
CT_ADAPTER_TCP	CoAP over TCP.
CT_DEFAULT	use when defaults are ok.
CT_FLAG_SECURE	secure the transport path.
CT_IP_USE_V4	IP adapter only.
CT_IP_USE_V6	IP adapter only.
CT_SCOPE_ADMIN	IPv6 Admin-Local scope.
CT_SCOPE_GLOBAL	IPv6 Global scope.
CT_SCOPE_INTERFACE	IPv6 Interface-Local scope(loopback).
CT_SCOPE_LINK	IPv6 Link-Local scope (default).
CT_SCOPE_ORG	IPv6 Organization-Local scope.
CT_SCOPE_REALM	IPv6 Realm-Local scope.
CT_SCOPE_SITE	IPv6 Site-Local scope.

SimpleServer - Android


```
@Override
public synchronized EntityHandlerResult handleEntity(final OcResourceRequest request) {
    EntityHandlerResult ehResult = EntityHandlerResult.ERROR;
    if (null == request) {
        msg("Server request is invalid");
        return ehResult;
    }
    // Get the request flags
    EnumSet<RequestHandlerFlag> requestFlags = request.getRequestHandlerFlagSet();
    if (requestFlags.contains(RequestHandlerFlag.INIT)) {
        msg("\t\tRequest Flag: Init");
        ehResult = EntityHandlerResult.OK;
    }
    if (requestFlags.contains(RequestHandlerFlag.REQUEST)) {
        msg("\t\tRequest Flag: Request");
        ehResult = handleRequest(request);
    }
    if (requestFlags.contains(RequestHandlerFlag.OBSERVER)) {
        msg("\t\tRequest Flag: Observer");
        ehResult = handleObserver(request);
    }
    return ehResult;
}
```

HandleEntity Implementation

```
private EntityHandlerResult handleRequest(OcResourceRequest request) {
    EntityHandlerResult ehResult = EntityHandlerResult.ERROR;
    // Check for query params (if any)
    Map<String, String> queries = request.getQueryParameters();
    if (!queries.isEmpty()) {
        msg("Query processing is up to entityHandler");
    } else {
        msg("No query parameters in this request");
    }

    for (Map.Entry<String, String> entry : queries.entrySet()) {
        msg("Query key: " + entry.getKey() + " value: " + entry.getValue());
    }

    //Get the request type
    RequestType requestType = request.getRequestType();
    switch (requestType) {
        case GET:
            msg("\t\t\tRequest Type is GET");
            ehResult = handleGetRequest(request);
            break;
        case PUT:
            msg("\t\t\tRequest Type is PUT");
            ehResult = handlePutRequest(request);
            break;
        case POST:
            msg("\t\t\tRequest Type is POST");
            ehResult = handlePostRequest(request);
            break;
        case DELETE:
            msg("\t\t\tRequest Type is DELETE");
            ehResult = handleDeleteRequest();
            break;
    }
    return ehResult;
}
```

Request Handler

```
private EntityHandlerResult handleObserver(final OcResourceRequest request) {
    ObservationInfo observationInfo = request.getObservationInfo();
    switch (observationInfo.getObserveAction()) {
        case REGISTER:
            if (null == mObservationIds) {
                mObservationIds = new LinkedList<>();
            }
            mObservationIds.add(observationInfo.getOcObservationId());
            break;
        case UNREGISTER:
            mObservationIds.remove((Byte)observationInfo.getOcObservationId());
            break;
    }
    // Observation happens on a different thread in notifyObservers method.
    // If we have not created the thread already, we do so now.
    if (null == mObserverNotifier) {
        mObserverNotifier = new Thread(new Runnable() {
            public void run() {
                notifyObservers(request);
            }
        });
        mObserverNotifier.start();
    }
    return EntityHandlerResult.OK;
}
```

Observer List Addition

Observing Thread Creation

```
private EntityHandlerResult handleGetRequest(final OcResourceRequest request) {
    EntityHandlerResult ehResult;
    OcResourceResponse response = new OcResourceResponse();
    response.setRequestHandle(request.getRequestHandle());
    response.setResourceHandle(request.getResourceHandle());

    if (mIsSlowResponse) { // Slow response case
        new Thread(new Runnable() {
            public void run() {
                handleSlowResponse(request);
            }
        }).start();
        ehResult = EntityHandlerResult.SLOW;
    } else { // normal response case.
        response.setErrorCode(SUCCESS);
        response.setResponseResult(EntityHandlerResult.OK);
        response.setResourceRepresentation(getOcRepresentation());
        ehResult = sendResponse(response);
    }
    return ehResult;
}
```

Get Resoponse

```
private EntityHandlerResult handlePutRequest(OcResourceRequest request) {
    OcResourceResponse response = new OcResourceResponse();
    response.setRequestHandle(request.getRequestHandle());
    response.setResourceHandle(request.getResourceHandle());

    setOcRepresentation(request.getResourceRepresentation());
    response.setResourceRepresentation(getOcRepresentation());
    response.setResponseResult(EntityHandlerResult.OK);
    response.setErrorCode(SUCCESS);
    return sendResponse(response);
}
```

<SimpleServer>

Android Simple Client & Server Test

- **Using Bluetooth (RFCOMM over Bluetooth)**