OPEN
CONNECTIVITY
FOUNDATION™

IoTivity 101
A Hands-On Class!

Presented by:
Joseph Morrow
Vijay Kesavan
Habib Virji

IoTivity

Pre-requisites For Hands-On

- Have a laptop with >= Ubuntu 12.04.
- Have WiFi capabilities. (If VM or Firewall is set, be sure to open up ports 5683 & 5684)
- Have issued the following command to ensure the environment is ready (You will need an internet connection!):

```
sudo apt-get update && sudo apt-get install scons build-essential g++ libboost-dev libboost-program-options-dev libboost-thread-dev uuid-dev libssl-dev libtool libglib2.0-dev
```

- **You may also share with another person.**

- WiFi SSID

“iotivity101”

- WiFi Password:

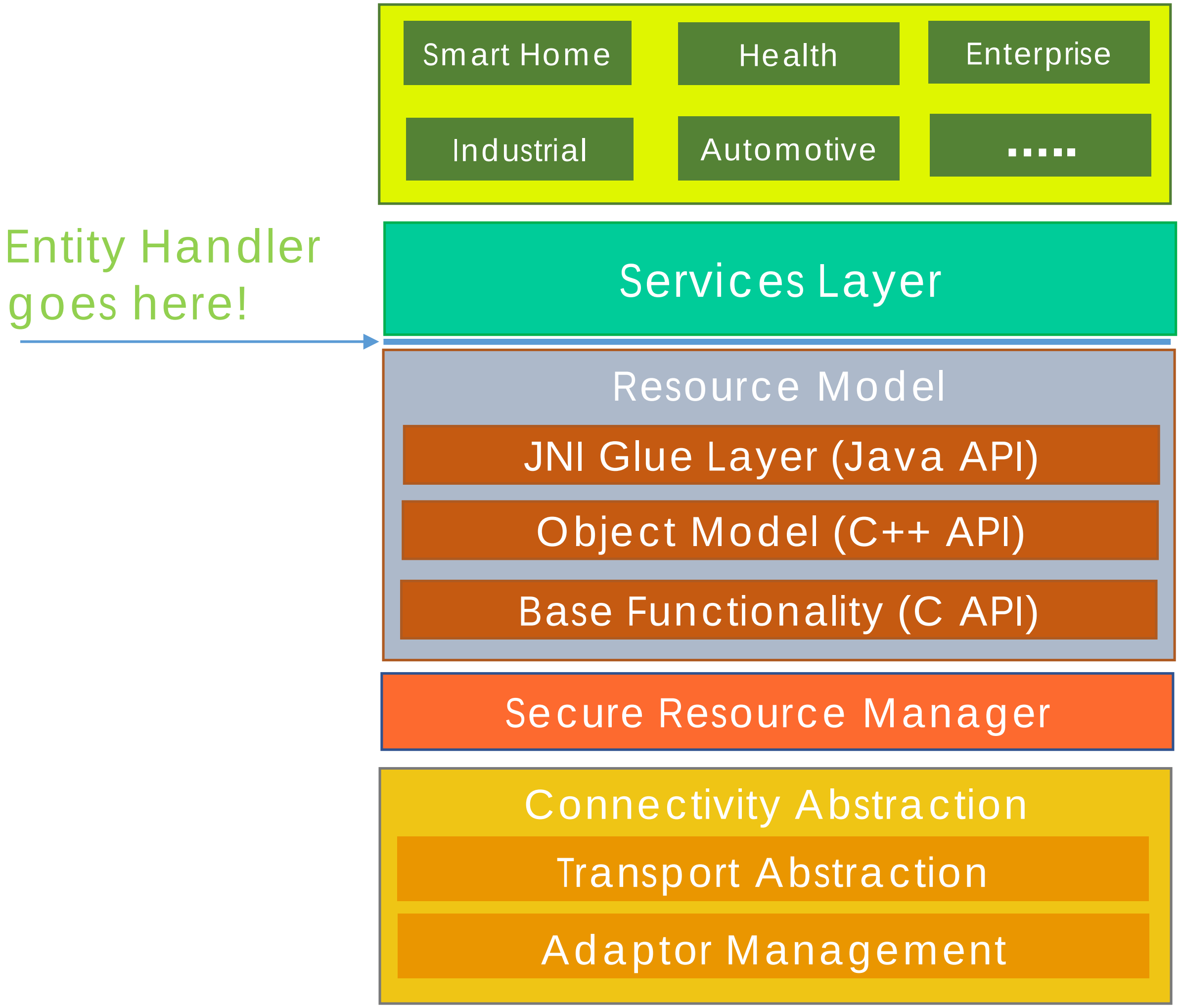
“password101”

The Agenda (90 Minutes)

- Architecture & Directory Structure
- Scons (Build System)
- Resource Representation
- Generic IoTivity Sever-Client flow
- Server
  - Resource registration/creation
  - Resource “Entity Handler” (Process & Respond to incoming CRUDN requests)
  - Chat Server
- Client
  - Device & Resource Discovery
  - Resource Requests (Send outgoing CRUDN requests)
  - Chat Client

IoTivity Stack Architecture

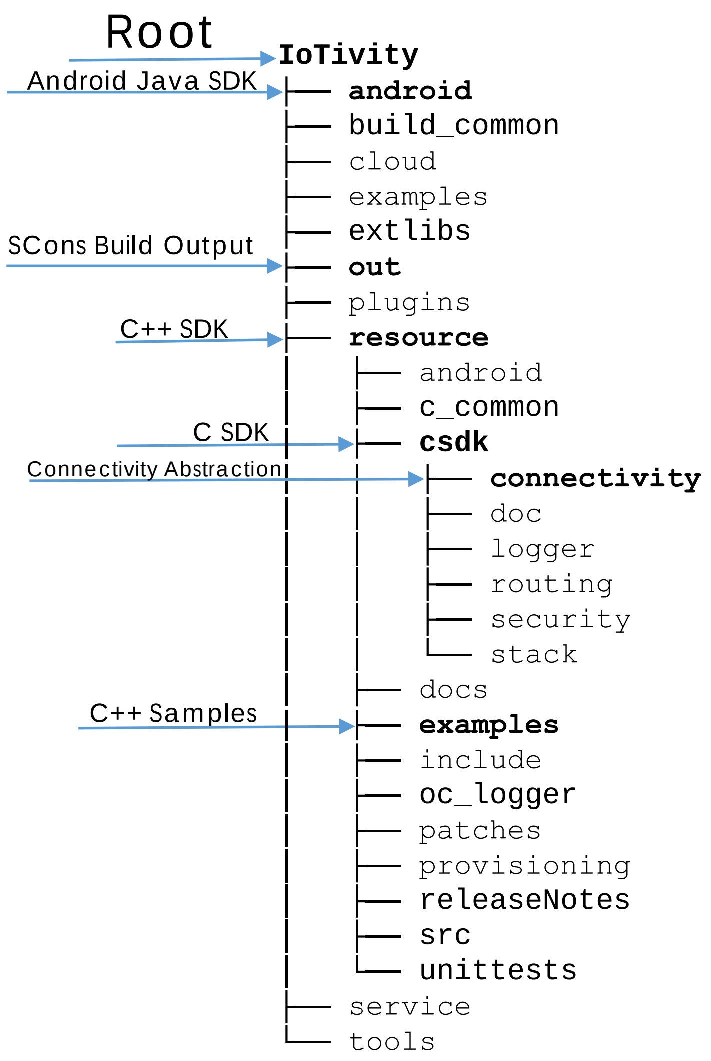
OPEN
CONNECTIVITY
FOUNDATION™



- APIs:
  - C: Resource model, RESTful
  - C++: Object model
  - Android: Built on C++
  - Windows enablement
  - Javascript binding
- Platforms:
  - Ubuntu (12.04)
  - Arduino: Due, ATMega 2560
  - Android
  - Tizen
  - Yocto

IoTivity

Directory Structure



Root

Android Java SDK

SCons Build Output

C++ SDK

C SDK

Connectivity Abstraction

C++ Samples

```
IoTivity
├── android
├── build_common
├── cloud
├── examples
├── extlibs
├── out
├── plugins
├── resource
│   ├── android
│   ├── c_common
│   ├── csdk
│   │   ├── connectivity
│   │   ├── doc
│   │   ├── logger
│   │   ├── routing
│   │   ├── security
│   │   └── stack
│   ├── docs
│   ├── examples
│   ├── include
│   ├── oc_logger
│   ├── patches
│   ├── provisioning
│   ├── releaseNotes
│   ├── src
│   └── unittests
├── service
└── tools
```

You can get a similar view of IoTivity by issuing command line “tree” at the root of the IoTivity project.

Scons – "A software construction tool"

- A Python-based build system that has a command line interface.
- http://scons.org/
  - Root File: SConstruct
  - Build Files: SConscript
- IoTivity Usage:
  - Entry Point:
    - <IOTIVITY>/SConstruct
  - SConscript:
    - Every directory with source files gets a SConscript.
  - Output Binaries:
    - <IOTIVITY>/out/<OS>/<ARCH>/<BUILD>/*
  - Further Information:
    - See <IOTIVITY>/Readme.scons.txt

All you have to do to start a build is issue following command where SConstruct resides: "scons"

To see available options, issue command: "scons -h"

Server-Client Flow

Server

1. PlatformCfg cfg;

{

2. RegisterResource()
3. EntityHandlers for Put, Post, Observe, Get, and Delete (Be sure to check for all even if you don't support them!)

}

4. Exit scope to de-initialize Iotivity.

Client

1. PlatformCfg cfg;

{

2. findResource()
3. onDiscoveryResponse()
4. Request() for Put, Post, Observe, Get or Delete.

}

4. onRequestResponse for Put, Post, Observe, Get, and Delete (eg. 'onPutResponse()')
5. Exit scope to de-initialize Iotivity.

Resource Representation

- C++ Interface:
  - Abstracts CBOR library "TinyCBOR".
  - <IOTIVITY>/resource/include/
    - OCRepresentation.h
    - AttributeValue.h
- Supported Types in C++:
  - Int
  - Double
  - Bool
  - String
  - Vector

What is CBOR?

- Data format that is based off of JSON data modeling, but the resulting encoded representation is compressed.
- Example Human-Readable CBOR Representation (Similar to JSON!):

```
[ // Begin Array
        { // Begin Map (aka "Object“ in JSON)
                "name": "John Doe",
                "color": "green",
                "message": "Hello world!"
        } // End Map (aka "Object“ in JSON)
]  // End Array
```

- See Backup Slide "CBOR" to see more about CBOR.

Server – Resource Registration/Creation

OPEN
CONNECTIVITY
FOUNDATION™

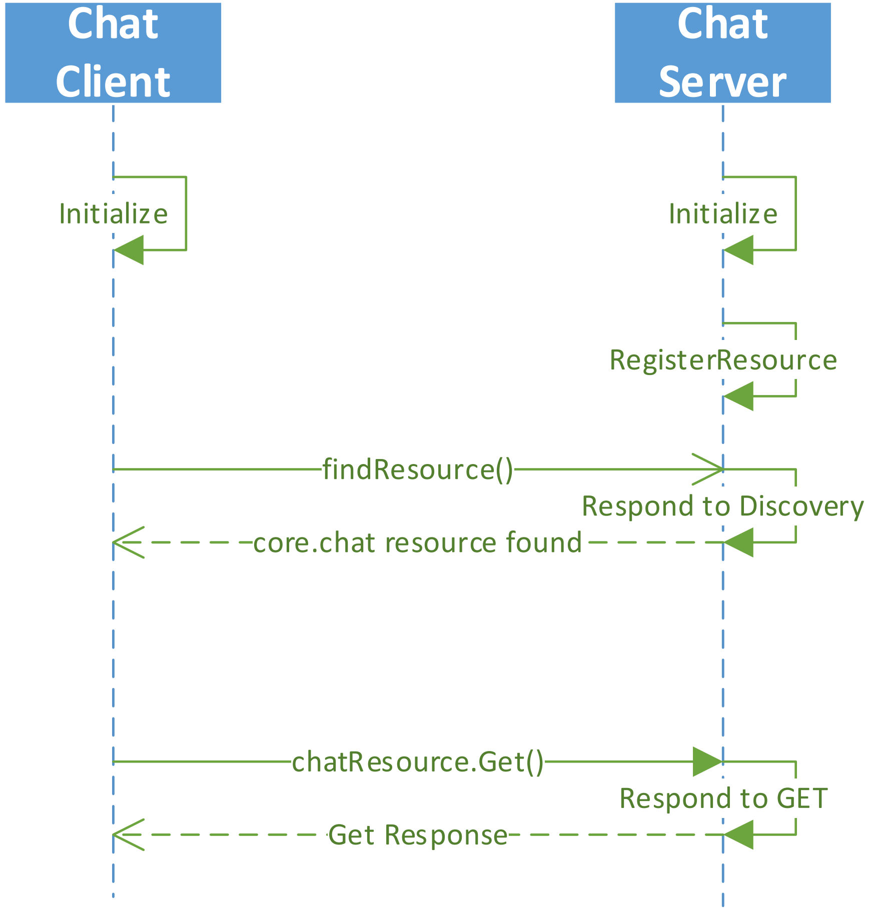
- C++ API – registerResource() Important Parameters

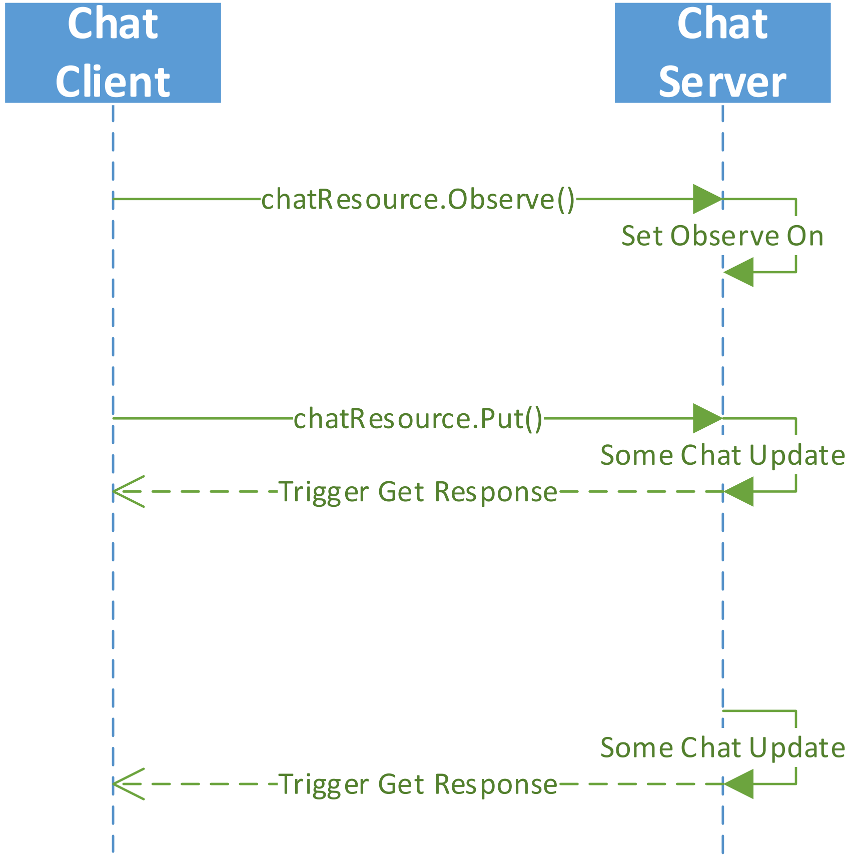
- **Resource URI** (I.E. OCF's addressing scheme.)
  - The Relative URI that you specify for your resource. (eg. "my/chatresource/")
- **Resource Type Name**
  - The Type that you specify for your resource. Think of this as the 'name' you've chosen for your resource. (eg. "chatresourcetype")
- **Resource Interface Name**
  - The Interface name you specify for your resource. Think of this as the "profile" you've chosen to implement against. (eg. "my.chat.interface")
- **Entity Handler**
  - The callback function you need registering to handle CRUDN requests.
- **Resource Properties**
  - Bit mask to specify the options for a resource:
    - Discoverable – If specified, the IoTivity stack will respond to Discovery requests on behalf of this resource.
    - Observable – If specified, this Header options for this resource will state that this resource is capable of supporting 'Notify'.
    - Secure – If specified, …

IoTivity

Chat Server & Client

OPEN
CONNECTIVITY
FOUNDATION™





Client sent a message.

Some other client sent a message.

IoTivity

Chat Client

- Please see README in “iotivity_developers_day” folder.

CRUDN Mappings in IoTivity

- OCF's CRUDN Maps to HTTP Verbs in IoTivity:

OCF Term	IoTivity Term
Create	Post
Retrieve	Get
Update	Put
Delete	Delete
Notify	Observe

OPEN
CONNECTIVITY
FOUNDATION™

Backup

IoTivity

Server – Incoming Request 1/2

- C++ API – EntityHandler()

```
OCEntityHandlerResult entityHandler(std::shared_ptr<OC::OCResourceRequest> request);
```

- request – The incoming request from client has the following public interface:

```
OCResourceRequest(); // Constructor
virtual ~OCResourceRequest(void); // Destructor
std::string getRequestType(); // Returns request types “GET”, “PUT”, “POST”, or “DELETE” in string form.
const QueryParamsMap& getQueryParameters(); // Returns map of queries on InterfaceType and ResourceType.
int getRequestHandlerFlag(); // Returns options whether this is an incoming normal request or observe request.
const OCRepresentation& getResourceRepresentation(); //  Returns resource representation from PUT request.
const ObservationInfo& getObservationInfo(); // Returns information for observe options.
void setResourceUri(const std::string resourceUri); // Sets the relative URI for this resource.
std::string getResourceUri(void); // Returns the relative URI for this resource.
const HeaderOptions& getHeaderOptions(); // Returns header options. These are options external to core payload.
const OCRequestHandle& getRequestHandle(); // Handle to this request.
const OCResourceHandle& getResourceHandle(); // Handle to this respective resource.
```

Server – Incoming Request 2/2

- C++ API – sendResponse()

```
OCStackResult sendResponse(const std::shared_ptr<OCResourceResponse> pResponse);
```

  - pResponse – The outgoing response to the client has the following public interface.

- Example use of sendResponse():

```
OCEntityHandlerResult entityHandler(std::shared_ptr<OC::OCResourceRequest> request)
{

/*  Hiding logic to ensure that this request is a properly formed GET request.  */
   OC::OCRepresentation chatRep = {};
   chatRep.setValue("name", m_name);
   chatRep.setValue("color", m_color);
   chatRep.setValue("message", m_message);

   auto pResponse = std::make_shared<OC::OCResourceResponse>();
   pResponse->setResourceRepresentation(chatRep);
   if(OC_STACK_OK == OC::OCPlatform::sendResponse(pResponse))
   {
      ehResult = OC_EH_OK;
   }
}
```

CBOR - “Concise Binary Object Representation”

OPEN CONNECTIVITY FOUNDATION™

- IoTivity uses TinyCBOR Library. See https://github.com/01org/tinycbor.
- Summary: Encode and Decode APIs are a lot like JSON’s, but the resulting payload is compressed OTA.
- C++ SDK:
  - Limited encode/decode abilities for data types because it abstracts the CBOR APIs.
  - <IOTIVITY>/resource/include/
    - OCRepresentation.h
    - AttributeValue.h
- C SDK
  - Has ability to encode/decode more data types.
  - <IOTIVITY>/resource/csdk/stack/include/
    - ocpayload.h

IoTivity

Resource Creation – “resourceProperty”

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

- Optional Resource Property Bit Masks:
  - OC_DISCOVERABLE
  - OC_OBSERVABLE
  - OC_SECURE
- Example:

```
uint8_t resourceProperty = (uint8_t) OC_DISCOVERABLE | OC_OBSERVABLE;
```

Slow Response

- When the Entity Handler cannot respond fast enough, the server may save the response handle (ie. reference to "OCResourceResponse" type) and make a subsequent call to OCPlatform::sendResponse() later when it can fulfill the request. The Entity Handler must return from the original request ASAP.