AllJoyn™ Software Framework: enabling the Internet of Everything

Service Frameworks Overview

Mitch Williams

Qualcomm Connected Experiences, Inc.

What is AllJoyn?

An Open Source API Framework For the Internet of Everything

A way devices and applications publish APIs over a network in a standard way

Why APIs?

- Because this is what software developers understand and work with every day

These APIs are the functionality that the “things” on the network expose to other “things”

- E.g. temperature, time of day, etc.…
- Services and/or devices can compose these APIs to provide whatever set of functionality they require
- The APIs are critical to interoperability between devices and services

How do applications know what APIs are available?

- AllJoyn provides application discovery and fine-grained discovery of the APIs supported by applications
- This is accomplished in a platform and radio-link agnostic way

Overview

AllJoyn implements a “distributed software bus”

- The bus provides the “medium” that enables AllJoyn applications to communicate via published APIs
  - Applications may be firmware on microcontrollers, mobile device “apps” or traditional applications on PCs/servers
- Applications publishing APIs are services, while those consuming the APIs are clients
  - An application can be both a service and a client: this is makes AllJoyn a peer-to-peer system
- Communication is via messages that map directly to APIs in high-level programming languages

Bus formation is ad hoc

- Based on discovery of applications/services
- Abstracts link-specific discovery mechanisms

Protocol is network-independent

- Wire protocol is based on the D-Bus wire-protocol with extensions
- Can run over Wi-Fi, Wi-Fi Direct, Ethernet, PLC and Bluetooth
  - Could likely run over others

Base Services

AllJoyn Base Services for fundamental use cases

- Standard AllJoyn building blocks for generically useful services
  - Notification
  - Onboarding
  - Control Panel
  - Config
- Accelerate application and device development
- All services utilize the About feature for advertising & discovery

Using About

Using the About Feature with Core Services

- Depending on which ‘piece’ of the service is being used, need either About ‘Client’ or About ‘Server’
- Each service framework code example shows how to perform this setup and initialization
- For more details, refer to the About Feature Usage Guides on AllSeenAlliance.org
- Generic boilerplate setup of AllJoyn and About required for each service framework

General AllJoyn & About Setup

- Create Bus Attachment
- Create Listeners (Bus, Session, SessionPort)
- Create a Session
- Join a Session

- Create About ‘Client’ or About ‘Server’
- Register service frameworks with About and Announce
- Setup Announcement Handler

Notification Service Framework

AllJoyn Notification Service Framework

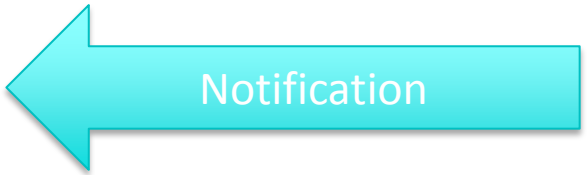
Simple, standardized interface for sending and receiving human-readable messages

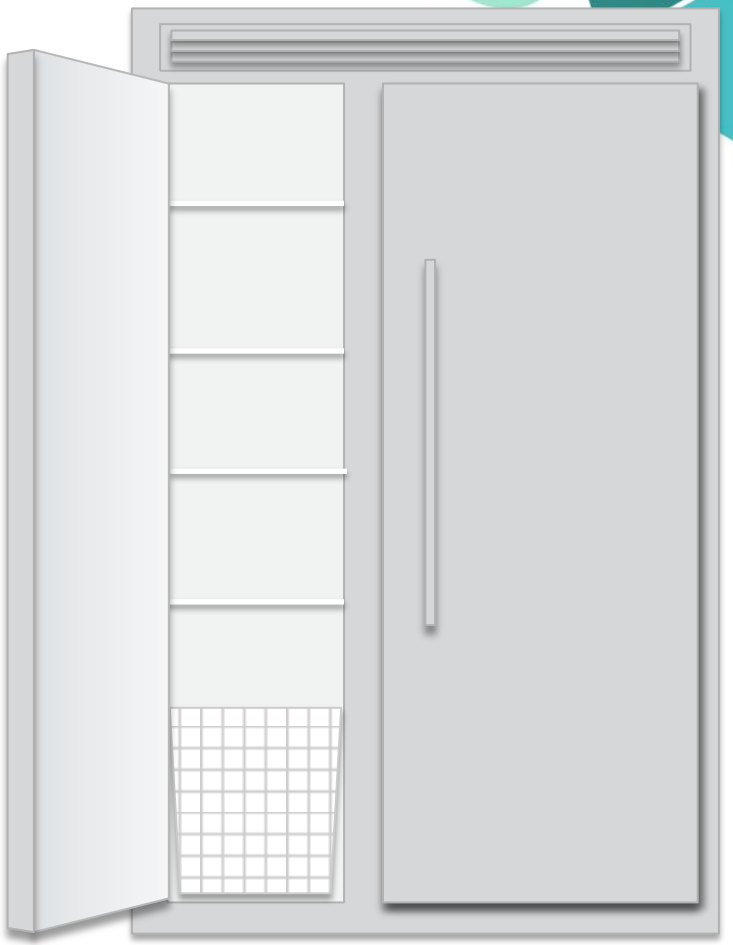
- Contents are text
  - Possible to reference image, audio, video: application can fetch media using reference
- Works across devices, operating systems and connection types
  - WiFi, Ethernet, PLC, etc…
- Producer (sender) can assign priority to notification
- Consumer (receiver) can configure preferences on types of notifications it receives

Examples

- Refrigerator could send a notification that freezer door has been left open for more than 5 minutes
  - This could be rendered on any consumer: mobile device, TV, set top box, etc…
- Washing machine can send a notification when wash cycle is complete

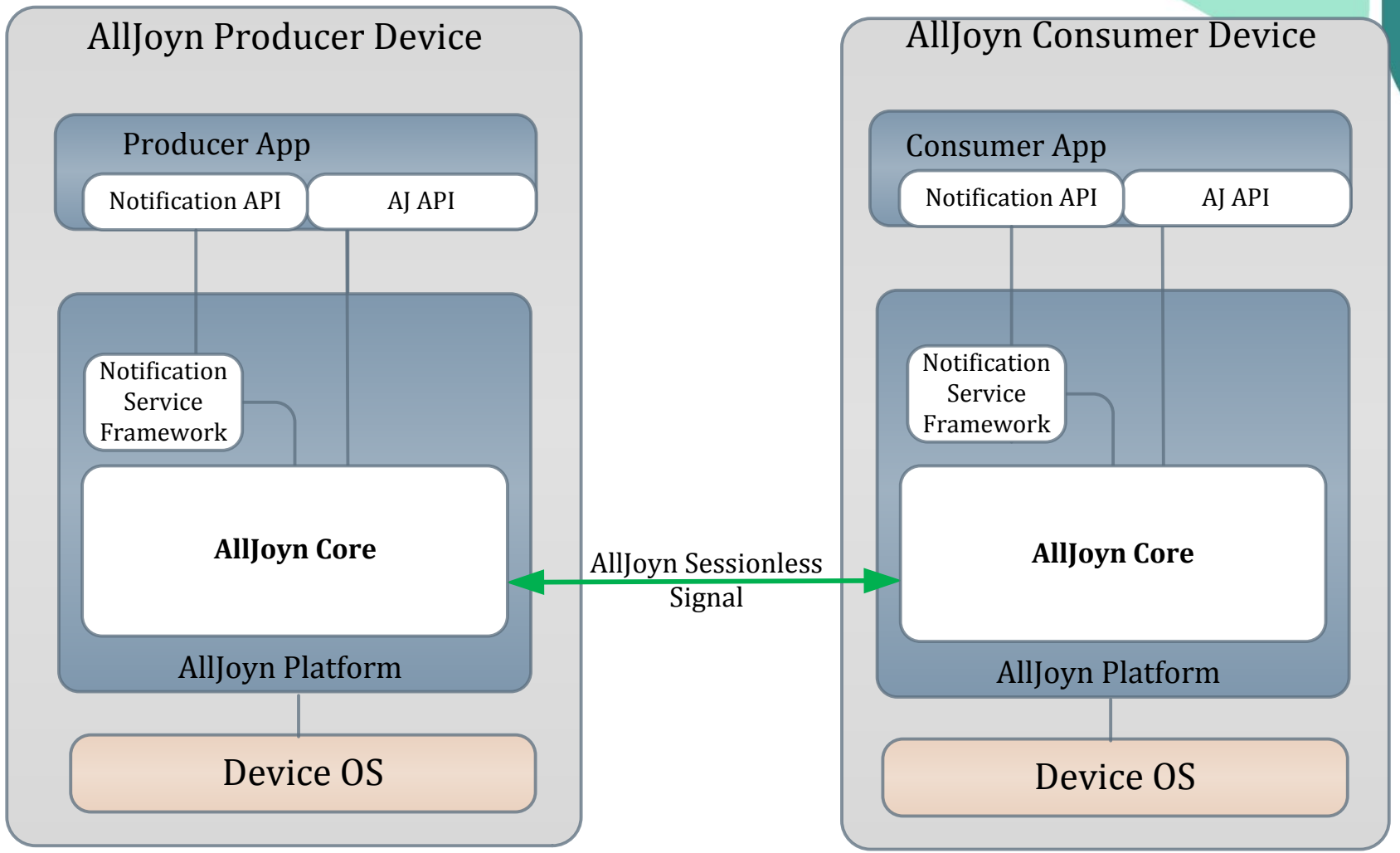
Notification Service Framework Example





- Freezer door left open for > 5 minutes
- Refrigerator emits notification
- TV renders it
  - Could also be rendered on a mobile device

Notification Service High Level Architecture



- Taken from Notification Service Interface Specification: https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-notification-service-framework-10-interface-specification

Notification Service Interface

```
<node
     xsi:noNamespaceSchemaLocation="https://www.allseenalliance.org/schemas/introspect.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <interface name="org.alljoyn.Notification">
          <property name="Version" type="q" access="read"/>
          <signal name="Notify">
                 <arg name="version" type="q"/>
                 <arg name="msgId" type="i"/>
                 <arg name="msgType" type="q"/>
                 <arg name="deviceId" type="s"/>
                 <arg name="deviceName" type="s"/>
                 <arg name="appId" type="ay"/>
                 <arg name="appName" type="s"/>
                 <arg name="langText" type="a{ss}"/>
                 <arg name="attributes" type="a{iv}"/>
                 <arg name="customAttributes" type="a{ss}"/>
          </signal>
     </interface>
</node>
```

- Taken from Notification Service Interface Specification: https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-notification-service-framework-10-interface-specification

Notification Service Sample Send & Receive

```
./ProducerBasic
Notification sent!
Exiting the application deletes the bus connection.
Waiting 10 seconds before exiting to allow the client to receive the message.
Goodbye!
```

```
./NotificationConsumerService
Begin Consumer Application. (Press CTRL+C to end application)
Enter in a list of app names (separated by ';') you would like to receive notifications from.
Empty list means all app names.

Waiting for notifications.
******************** Begin New Message Received ********************
Message Id: 1203620680
Device Id: ProducerBasic
Device Name: ProducerBasicDeviceName
App Id: 2826752AE35C416A82BCEF272C55EACE
App Name: testappName
Sender BusName: :ML0oo6yH.2
Message Type 2 info
Language: en  Message: Hello World
Language: es_SP  Message: Hola Mundo
Other parameters included:
Custom Attribute Key: Off  Custom Attribute Value: Goodbye
Custom Attribute Key: On  Custom Attribute Value: Hello
Rich Content Icon Url: http://iconurl.com
******************** Begin Rich Audio Content ********************
Language: en  Audio Url: http://url1.com
Language: es_SP  Audio Url: http://url2.com
******************** End Rich Audio Content ********************
Rich Content Icon Object Path: /icon/objectPath
Rich Content Audio Object Path: /Audio/objectPath
ControlPanelService object path: /ControlPanel/MyDevice/areYouSure
******************** End New Message Received ********************
```

Notification Service - Producer

Send Notification (C++)

```
NotificationService* prodService = 0;
NotificationSender* Sender = 0;

...

Sender = prodService->initSend(bus, propertyStoreImpl);

Notification notification(messageType, vecMessages);
notification.setCustomAttributes(customAttributes);

...

if (Sender->send(notification, ttl) != ER_OK) {
    std::cout << "Could not send the message successfully" << std::endl;
} else {
    std::cout << "Notification sent with ttl of " << ttl << std::endl;
}
```

- Code snippet from Notification Service: https://git.allseenalliance.org/cgit/services/notification.git/tree/cpp/samples/ProducerService/ProducerService.cc

Notification Service - Consumer

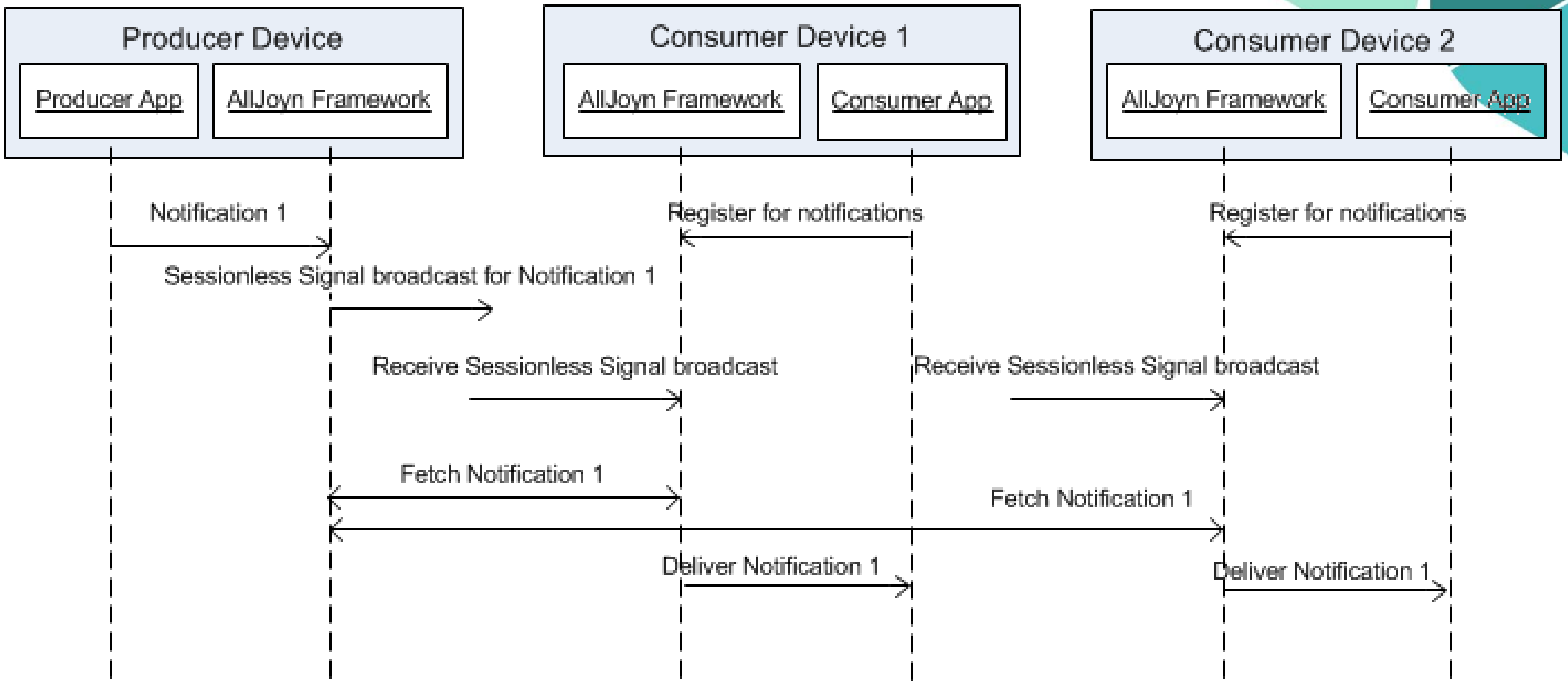
Receive Notification (Android)

```
public void startReceiver() {
        aboutService.startAboutClient(bus);
        …
        notificationService.initReceive(bus, this);
        bus.addMatch("sessionless='t',type='error'");
        …
}

public void receive(Notification notification) {
        String notifAppName = notification.getAppName();
        …
        renderNotification(notification);
}
```

- Code snippet from Notification Service: https://git.allseenalliance.org/cgit/services/notification.git/tree/java/sample_applications/android/NotificationServiceUISample/src/org/alljoyn/ns/sampleapp/IoeNotificationApplication.java

Notification Service Call Flow



- Taken from Notification Service Interface Specification: https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-notification-service-framework-10-interface-specification

Notification Best Practices & Debugging

TTL (Time to Live) should be set to correspond to the type of information included in the notification

- For example, if contents of message are no longer valid after 5 minutes, TTL should be set accordingly

Message Type should be set to correspond to the type of information included in the notification

- Use ‘Information’, ‘Warning’, ‘Emergency’

Debugging

- Can use command line apps shown previously to send & receive notifications across platforms
  - can send from Linux command line app and receive on Android app (or vice versa)

Onboarding Service Framework

AllJoyn Onboarding Service Framework

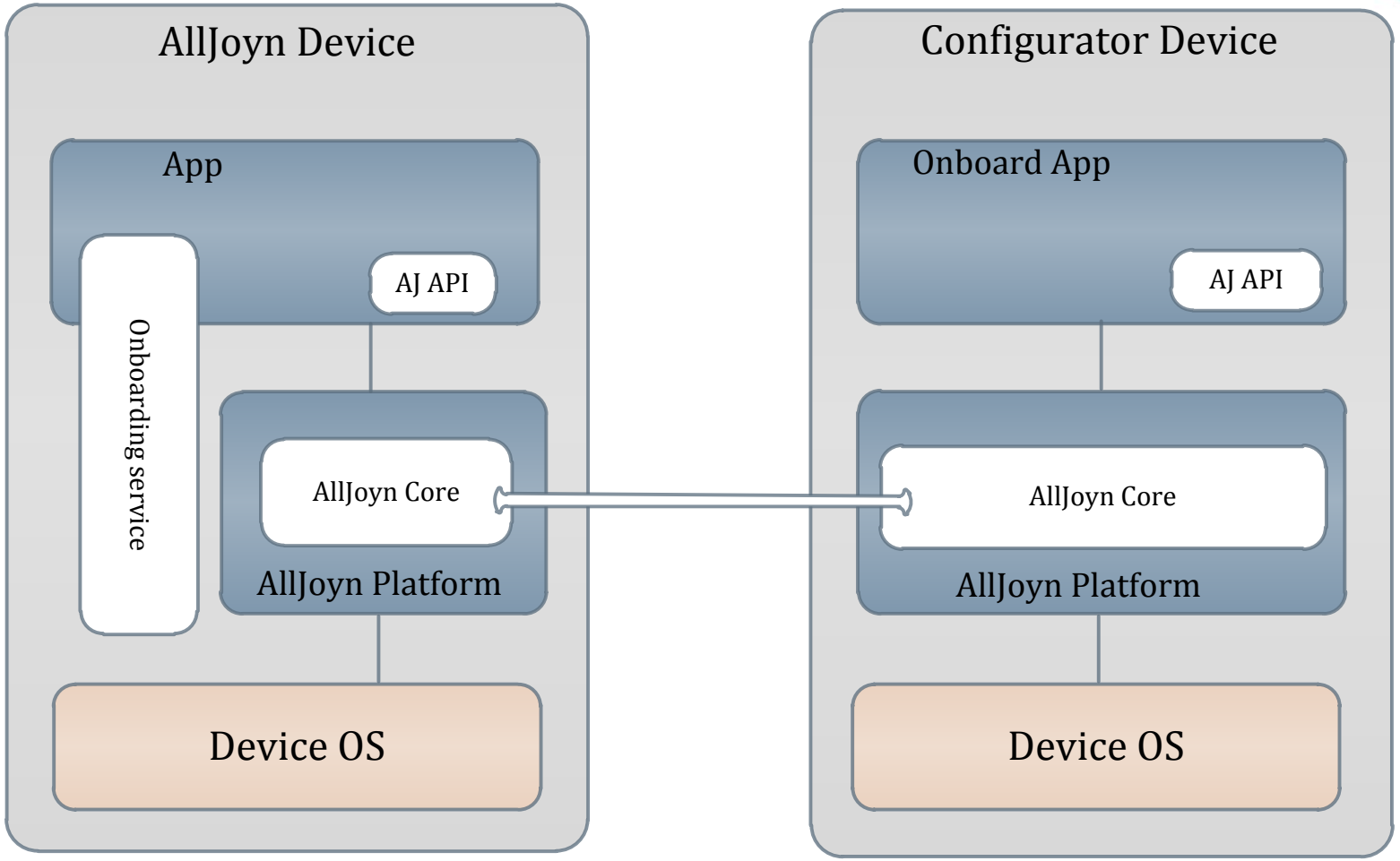
Provides a standard way to get devices onto a WiFi Network

- Onboarder is an application running on a smart device
- Onboardee is the device to be added to the WiFi network

Basic flow

- Onboarder discovers device that needs to be onboarded
- Connects to it, and provides configuration information
- Onboardee verifies it can connect
  - Informs onboarder that it has been successful or not

Onboarding Service High Level Architecture



- Taken from Onboarding Service Interface Specification: https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-onboarding-service-framework-10-interface-specification

Onboarding Service Interface

```
<node
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="http://www.allseenalliance.org/schemas/introspect.xsd">
     <interface name="org.alljoyn.Onboarding">
        <property name="Version" type="q" access="read"/>
        <property name="State" type="n" access="read"/>
        <property name="LastError" type="(ns)" access="read"/>
        <method name="ConfigureWifi">
            <arg name="SSID" type="s" direction="in"/>
            <arg name="passphrase" type="s" direction="in"/>
            <arg name="authType" type="n" direction="in"/>
            <arg name="status" type="n" direction="out"/>
        </method>
        <method name="Connect">
          <annotation name="org.freedesktop.DBus.Method.NoReply" value="true" />
         </method>
        <method name="Offboard">
            <annotation name="org.freedesktop.DBus.Method.NoReply" value="true" />
        </method>
        <method name="GetScanInfo">
            <arg name="age" type="q" direction="out"/>
            <arg name="scanList" type="a(sn)" direction="out"/>
        </method>
        <signal name="ConnectionResult">
            <arg type="(ns)" />
        </signal>
    </interface>
</node>
```

- Taken from Onboarding Service Interface Specification: https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-onboarding-service-framework-10-interface-specification

Onboarding Flow

1. Onboardee device broadcasts in SoftAP mode

– AllJoyn boilerplate setup, Onboarding service setup

– Advertises via About that it supports Onboarding

– NOTE: Setup & use of SoftAP is device specific code

```
obController = new OnboardingControllerImpl(opts.GetScanFile(),
                                            opts.GetStateFile(),
                                            opts.GetErrorFile(),
                                            opts.GetConfigureCmd(),
                                            opts.GetConnectCmd(),
                                            opts.GetOffboardCmd(),
                                            (OBConcurrency)opts.GetConcurrency(),
                                            *msgBus);
OnboardingService onboardingService(*msgBus, *obController);

interfaces.clear();
interfaces.push_back("org.alljoyn.Onboarding");
aboutService->AddObjectDescription("/Onboarding", interfaces);

status = onboardingService.Register();
```

- Code snippet from Onboarding Service: https://git.allseenalliance.org/cgit/services/onboarding.git/tree/cpp/samples/OnboardingServiceSample/OnboardingServiceMain.cc

Onboarding Flow

2. Onboarder scans for SoftAPs

- SoftAP used by Onboardee follows a specific naming convention
  - Look at the interface specification for example
- NOTE: In example apps, SoftAP that Onboarder is expecting is often hard coded
- Scanning for SoftAPs is often device/OS specific code

Onboarding Flow

3. Onboarder connects to Onboardee SoftAP

– Onboarder sees About announcement from Onboardee

– Onboarder uses info in announcement to initialize connection via Onboarding Service to Onboardee

```
AnnouncementHandler receiver = new AnnouncementHandler(){

        @Override
        public void onAnnouncement(String busName, short port, BusObjectDescription[]
interfaces, Map<String, Variant> aboutMap) {

                Map<String, Object> newMap = new HashMap<String, Object>();
                try {
                        newMap = TransportUtil.fromVariantMap(aboutMap);
                        String deviceId = (String) (newMap.get(AboutKeys.ABOUT_APP_ID).toString());
                        String deviceFriendlyName = (String) newMap.get(AboutKeys.ABOUT_DEVICE_NAME);
                        m_logger.debug(TAG, "onAnnouncement received: with parameters:
busName:"+busName+", port:"+port+", deviceid"+deviceId+", deviceName:"+deviceFriendlyName);
                        addDevice(deviceId, busName, port, deviceFriendlyName, interfaces,
newMap);
                } catch (BusException e) {
                        e.printStackTrace();
                }
        }
```

- Code snippet from Onboarding Service: https://git.allseenalliance.org/cgit/services/onboarding.git/tree/java/sample_applications/android/OnboardingSampleClient/src/org/alljoyn/onboarding/test/OnboardingApplication.java

Onboarding Flow - Continued

4. Onboarder provides credentials for AP to onboard to

– SSID, passphrase, authentication type

```
@Override
public short configureWiFi(String ssid, String passphrase, short authType) throws BusException
{
        ProxyBusObject proxyObj = getProxyObject();
        // We make calls to the methods of the AllJoyn object through one of its interfaces.
        OnboardingTransport onboardingTransport =  proxyObj.getInterface(OnboardingTransport.class);
        return onboardingTransport.ConfigureWiFi(ssid, passphrase, authType);
}
```

- Code snippet from Onboarding Service: https://git.allseenalliance.org/cgit/services/onboarding.git/tree/java/OnboardingService/src/org/alljoyn/onboarding/client/OnboardingClientImpl.java

```
void OnboardingService::ConfigureWiFiHandler(const ajn::InterfaceDescription::Member* member, ajn::Message&
msg) {
…
 if (WEP != authType) {
   size_t rawLength = strPass.length() / 2 + 1;
   char raw[rawLength];
   CHECK_BREAK(HexToRaw(strPass.c_str(), strPass.length(), raw, rawLength));
   raw[strPass.length() / 2] = '\0';
   m_OnboardingController.ConfigureWiFi(SSID, raw, authType, configureWifiStatus, error, errorMessage);
 } else {
   m_OnboardingController.ConfigureWiFi(SSID, strPass, authType, configureWifiStatus, error, errorMessage);
 }
```

- Code snippet from Onboarding Service: https://git.allseenalliance.org/cgit/services/onboarding.git/tree/cpp/src/OnboardingService.cc

Onboarding Flow - Continued

5. Onboardee & Onboarder connect to the provided AP

– If Onboardee is unsuccessful in connecting, will fall back to SoftAP, can resume at Step 4

```
@Override
public void connectWiFi() throws BusException
{
        ProxyBusObject proxyObj = getProxyObject();
        // We make calls to the methods of the AllJoyn object through one of its interfaces.
        OnboardingTransport onboardingTransport =  proxyObj.getInterface(OnboardingTransport.class);
        onboardingTransport.Connect();
}
```

- Code snippet from Onboarding Service: https://git.allseenalliance.org/cgit/services/onboarding.git/tree/java/OnboardingService/src/org/alljoyn/onboarding/client/OnboardingClientImpl.java

```
/*-------------------------------------------------------------------------
 * METHOD: Connect()
 * This method is called by the ConnectHandler with the corresponding input and
 * output arguments. This method is empty, the developer should fill it with the
 * developer's implementation of the Connect method handler.
 *-------------------------------------------------------------------------*/
void OnboardingControllerImpl::Connect() {
/* Fill in method handler implementation here. */
    std::cout << "entered Connect" << std::endl;
    CancelAdvertise();
    execute_system(m_connectCmd.c_str());
    AdvertiseAndAnnounce();
} /* Connect() */
```

- Code snippet from Onboarding Service: https://git.allseenalliance.org/cgit/services/onboarding.git/tree/cpp/samples/OnboardingServiceSample/OnboardingControllerImpl.cc

Onboarding Flow - Continued

6. Both Onboarder & Onboardee on new AP, Onboarding is complete

- Onboardee tells Onboarder if it was successful
- Can use the state managed by the Onboarding Service

```
/**
 * @return the state:
 * 0 - Personal AP Not Configured
 * 1 - Personal AP Configured/Not Validated
 * 2 - Personal AP Configured/Validating
 * 3 - Personal AP Configured/Validated
 * 4 - Personal AP Configured/Error
 * 5 - Personal AP Configured/Retry
 * @throws BusException
 */
@BusProperty(signature="n")
public short getState() throws BusException;
```

- Code snippet from Onboarding Service: https://git.allseenalliance.org/cgit/services/onboarding.git/tree/java/OnboardingService/src/org/alljoyn/onboarding/transport/OnboardingTransport.java

Development & Debugging

What if I don’t have a device to onboard?

– Can use the “Onboarding, About, Config Simulator Application”
  - Android application that runs as a background service
  - Simulates a device that can be onboarded
  - Source: https://git.allseenalliance.org/cgit/services/simulators.git/tree/android/about_conf_onb_server

Onboarding Service – After Onboarding

Once app/device has been onboarded:

- Can use the Config Service to set a “Friendly Name” for the device

Config Service

- Enables ability to set configurable persistent values
- Flexible for developers to customize to add their own fields
- By default allows for a “Friendly Name” to be set. This name provides an end user the ability to specify a string that they can associate with the product, i.e. “Living Room TV”, “Patio Speaker”, etc.

Offboarding

- Onboarding Service also has ability to offboard a device

Control Panel Service Framework

AllJoyn Control Panel Service Framework

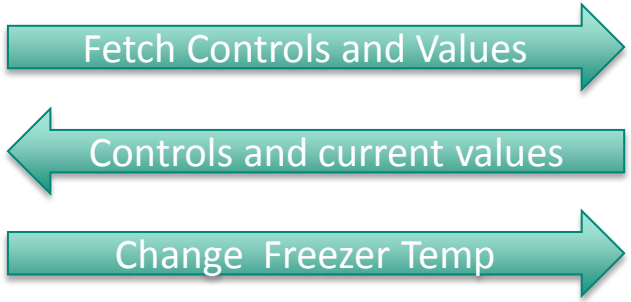
Infrastructure for exposing user interfaces for devices remotely

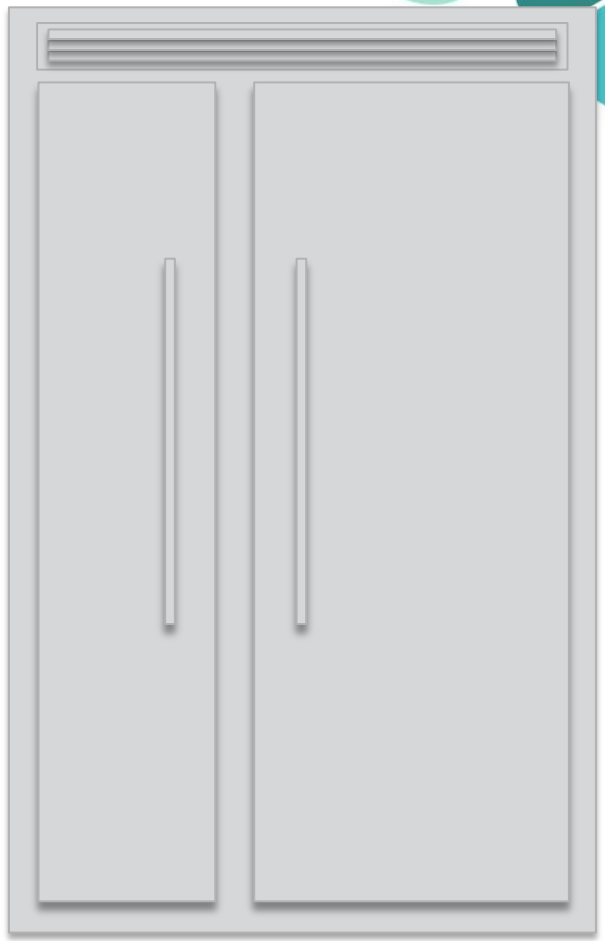
- Model is there is a controller and a controllee
- Controlee exposes it's UI using the framework
- Controller renders the UI and sends control commands back to controllee base on UI input
- Defined such that any control application using the framework can render a controllee exposed UI
  - That is a generic app can control any type of devices that exposes controls using this framework

Examples

- After receiving a notification that the oven has been on Broil for 5 minutes a user could bring up the oven's control panel and change the setting to "bake at 250" to keep the food warm
- A user could check the current values of a refrigerator (including current temperature) and modify the settings to make things hotter or colder as needed.

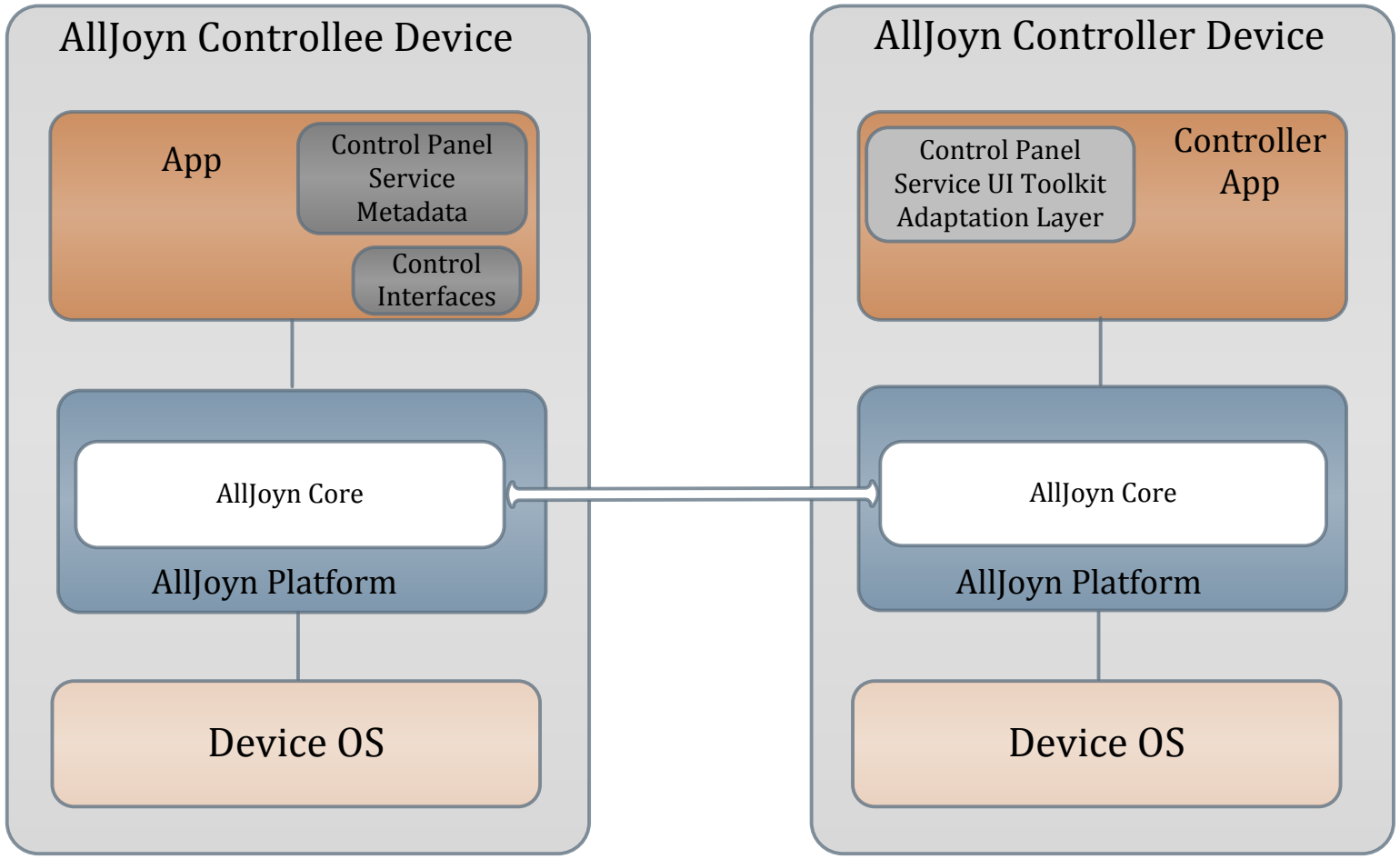
Control Panel Service Framework Example





- After detecting refrigerator
- Mobile device fetches the controls and values
- On receipt it renders them on the display
- Freezer temperature changed on mobile
- Change in the temperature is reported to refrigerator

Control Panel Service High Level Architecture



- Taken from Control Panel Service Interface Specification: https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-control-panel-service-framework-10-interface-specification

Creating a Control Panel

Design (not define) controls via XML

- Types of controls, layout, widget hints
- Define callbacks to be triggered

Use code generator tool to create ‘Generated Code’

- Takes XML definition and produces code for Control Panel
- Located in the ‘tools’ directory for C++ and C (Thin Client)

Create ‘Provided Code’ for your app/device/platform

- Callbacks used by generated code
- Example is the code that actually changes the temperature for a refrigerator/freezer

Control Panel framework combines Generated & Provided code

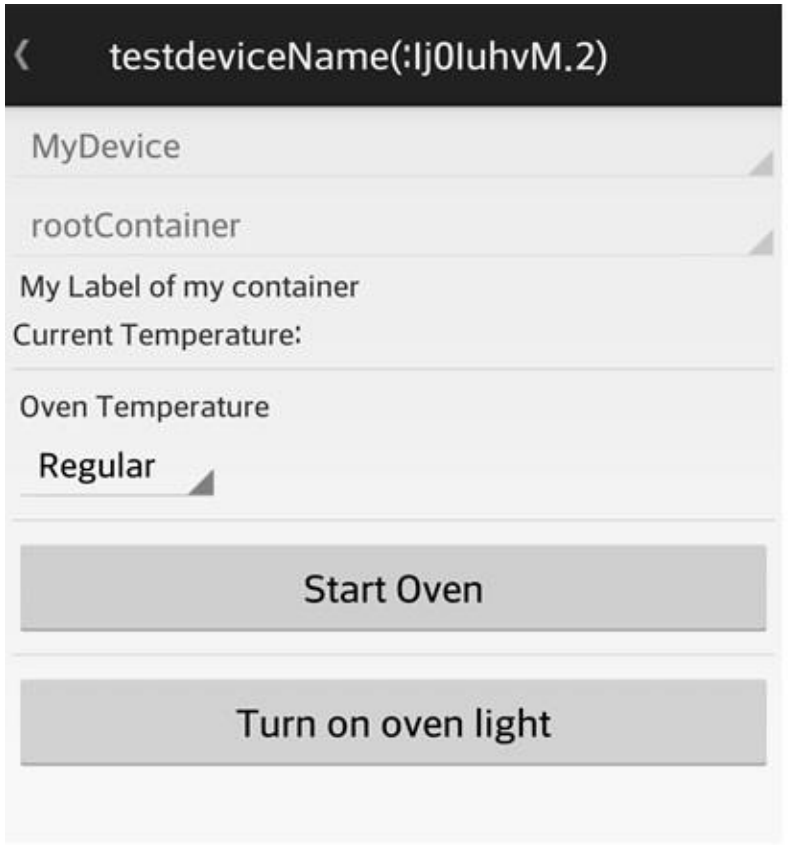
Sample Control Panel XML

```
...
<elements>
   <labelProperty>
       <name>CurrentTemp</name>
       <enabled>true</enabled>
       <label>
            <value type="literal" language="en">Current Temperature:</value>
...
       </label>
...
       <hints>
             <hint>textlabel</hint>
       </hints>
   </labelProperty>
   <scalarProperty dataType="UINT16">
       <name>heatProperty</name>
       <getCode>getuint16Var</getCode>
       <setCode>setuint16Var(%s)</setCode>
       <secured>false</secured>
       <enabled>true</enabled>
       <writable>true</writable>
...
```

- Code snippet from Control Panel Service: https://git.allseenalliance.org/cgit/services/controlpanel.git/tree/cpp/tools/CPSAppGenerator/SampleXMLs/testOneOfEachWidget.xml

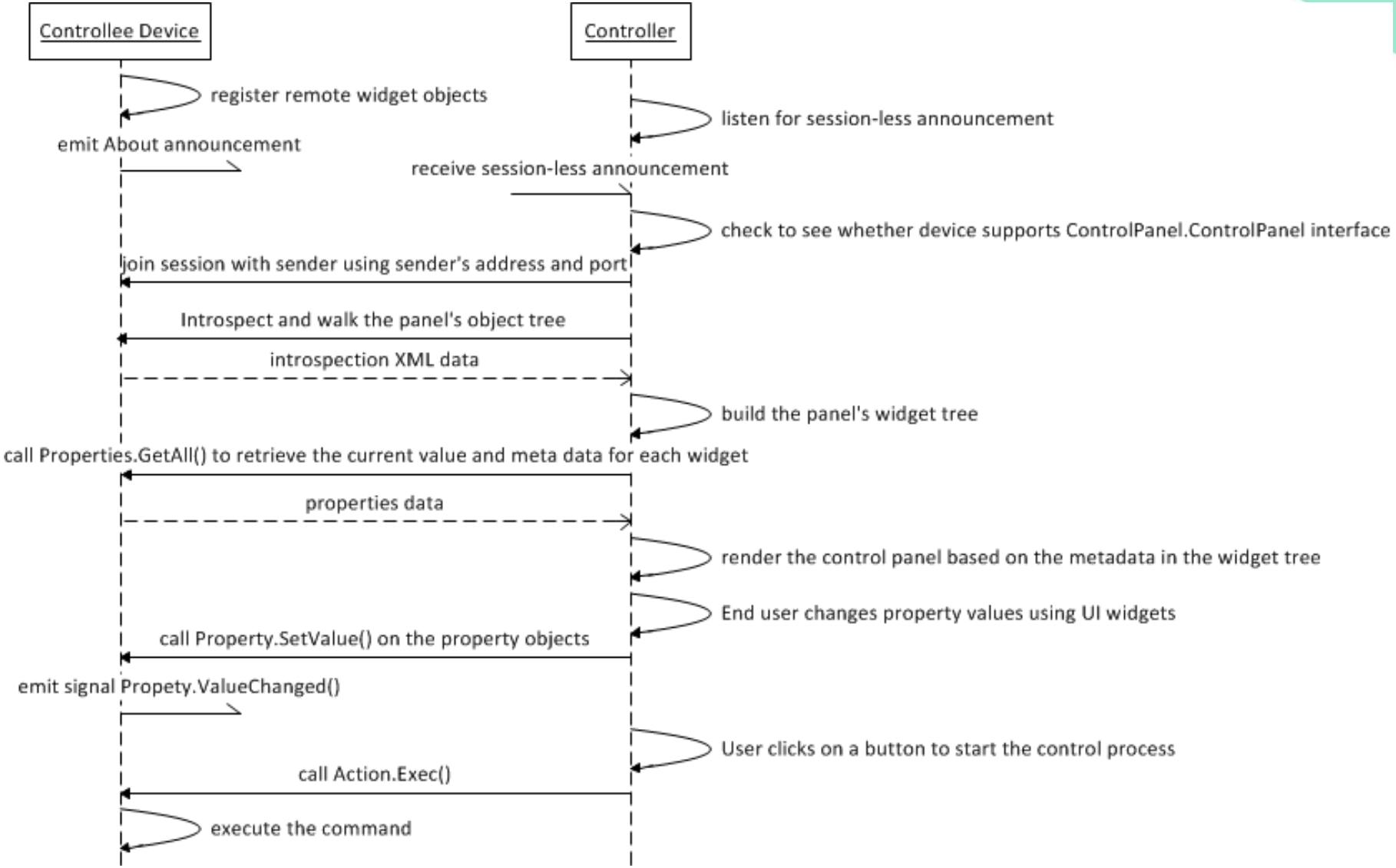
Sample Control Panel in Android

Control Panel as shown by ControlPanelBrowser sample app in Control Panel Android SDK



ControlPanelBrowser.apk located at alljoyn-android/services/alljoyn-controlpanel-1.0.1-rel/tools/

Control Panel Service Call Flow



- Taken from Control Panel Service Interface Specification: https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-control-panel-service-framework-10-interface-specification

Recap

Services are standard AllJoyn building blocks

- Notification, Onboarding, Config, Control Panel
- All services utilize the About feature for advertising & discovery
- Services are interoperable across platforms & language bindings

Notification

- “The text message for the Internet of Everything”

Onboarding

- Provides standard way to get devices on a WiFi network

Control Panel

- Framework for exposing user interfaces for devices remotely

Source Code license information

Source code contained in this presentation is licensed by the AllSeen Alliance under the ISC open source license:


質問は？
Questions?
问题？
Ερωτήσεις?
Questions?
Fragen?
Вопросы?
Spørsmål?
Questions?
Domande?
Questions?
Vragen?
질문이 있습니까?
Cwestiynau?
Spørgsmål?
Questions?
FRÅGOR?
¿Preguntas?
Questions?
Kysymyksiä?

Thank You!