ALLSEEN ALLIANCE

AllJoyn Framework System Overview

Brian Spencer
Engineer, Staff/Manager

Qualcomm Connected Experiences, Inc. (QCE)

August, 2014

What is the AllJoyn Framework?

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

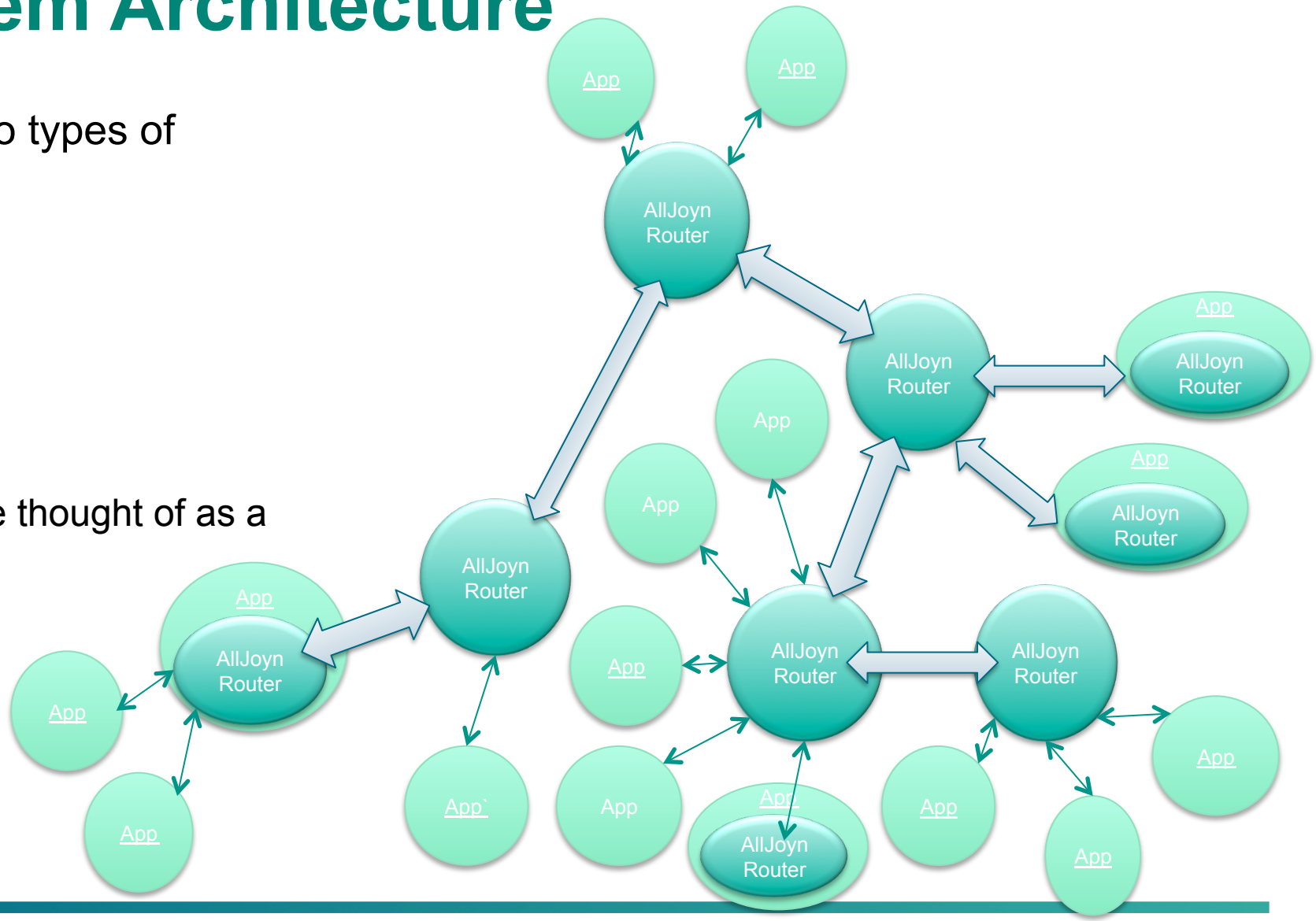
Overview

- The AllJoyn framework 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
  - Can run over Wi-Fi, Wi-Fi Direct, Ethernet, PLC and Bluetooth
    - Could likely run over others

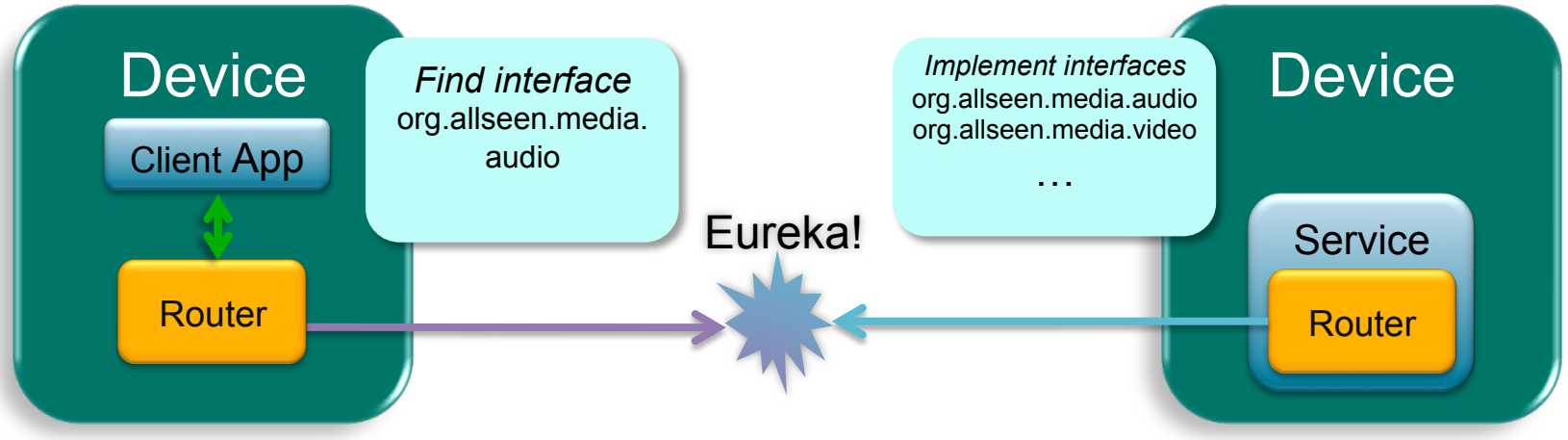
High-Level System Architecture

AllJoyn Bus is composed of two types of nodes:

- AllJoyn Router (R)
- Applications (App)
  - App can only connect to R
  - App can contain R
  - R can connect to other R
- The AllJoyn framework can be thought of as a mesh of stars



Ad Hoc Bus Formation: Discover



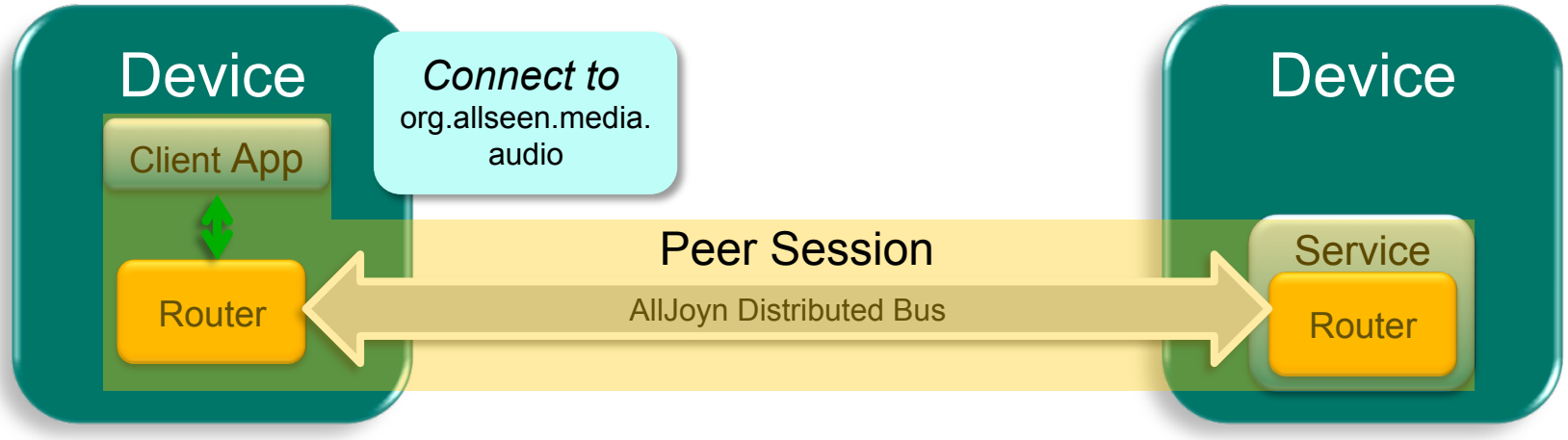
Interface descriptions contained in About message

- Services advertise, and Clients find, About messages
- Connect to advertisers supporting desired interfaces

Actual discovery mechanism is transport-dependent:

- On Wi-Fi, PLC, Ethernet: lightweight IP multicast protocol
- On Wi-Fi Direct: would use pre-association discovery

Ad Hoc Bus Formation: Session Creation



Session creation will cause AllJoyn Routers to connect and extend the bus

- Once connected, buses merge and have a single shared namespace
- Peers can discover when other peers join or leave the bus
- Peers can interact via their APIs
- Session reference counting keeps device-to-device connections alive
- Multicast events can be sent to all peers in the session

Software Components

AllJoyn Software Framework: High-level architecture

A comprehensive software framework lets devices and applications communicate

AllJoyn App Layer

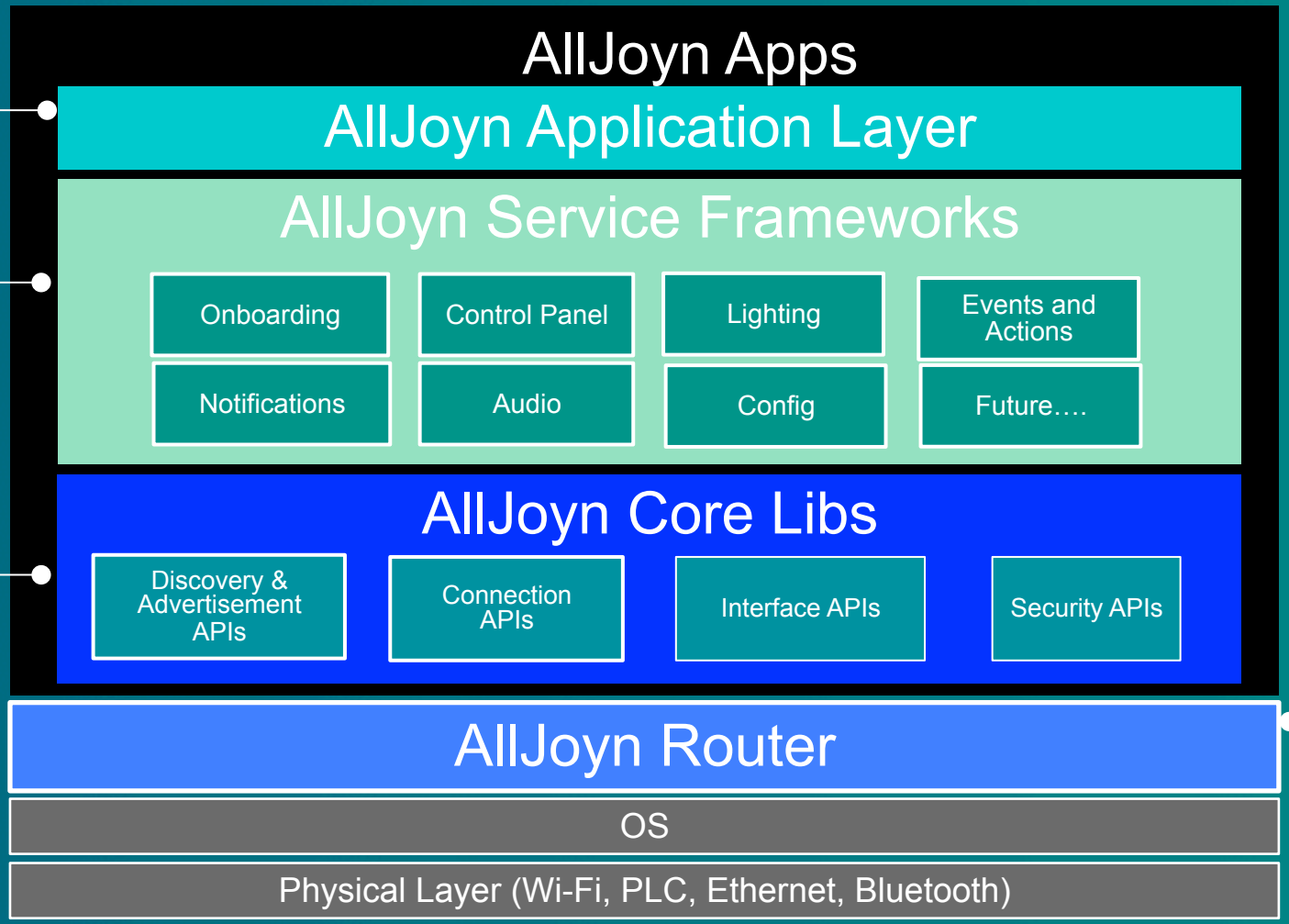
- Defines the User experience

AllJoyn Service Frameworks

- Interoperable, cross-platform modules for common IoE functionality
- Defines common interfaces between devices

AllJoyn Core Libs

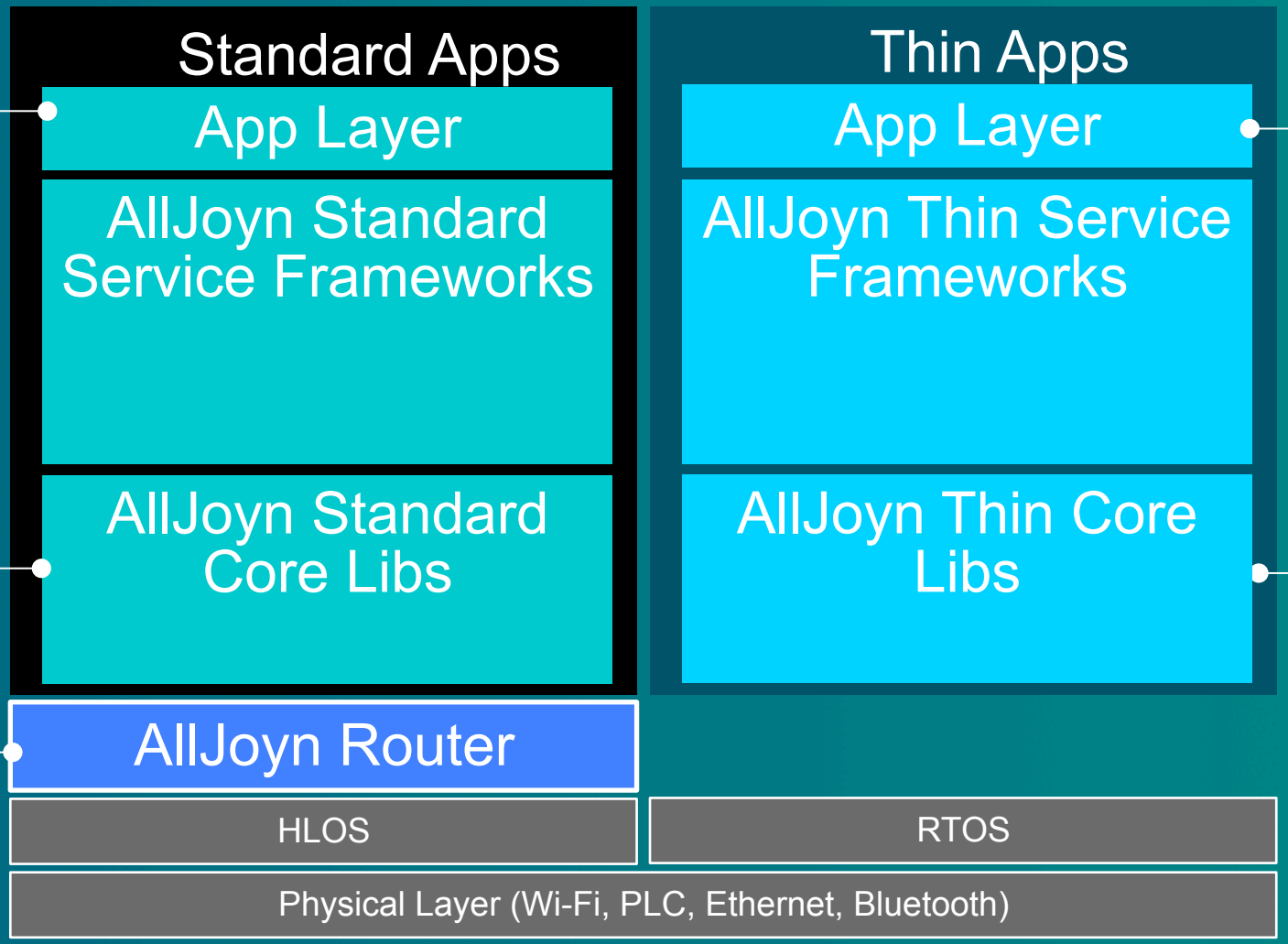
- Provides ability to find and connect to devices to do interesting things.
- Core libraries interact with the AllJoyn Router
- Provides access control and encryption



AllJoyn Router

- Manages communications between devices and apps
- Dynamic network management

Two Versions of the AllJoyn Framework To Choose



Standard App Layer

- App dev or OEM writes this

Standard Core Libraries

- Multiple bindings, runs on HLOS

AllJoyn Router

- AllJoyn Router can be bundled with a Standard App or run standalone

Thin App Layer

- OEM writes this

Thin Core Libraries

- C bindings, runs on RTOS
- Thin Apps using Thin Core requires an AllJoyn Router in the network

Software Components

Two main architectural components: the Core Library and the Router

Core Library

- This is referred to as a Core Library because all AllJoyn framework applications are “clients” of the router
  - This is true regardless of if, in their application context, they are exposing services, or are clients of other services
  - Applications are peers if they implement both client and service functionality
- The Core Library is what software developers interact with: the API set of the AllJoyn SDK
- There are two implementations of the Core Library: the **Standard Library** (SL) and the **Thin Library** (TL)
- The SL is targeted at applications running in HLOS environments
  - The SDK APIs for the SC provide a high level of abstraction, and allow complex multi-threaded applications
  - Native implementation is in C++ and there are a number of language bindings for various platforms available
- The TL is targeted at applications that would reside on deeply embedded devices (i.e., device firmware)
  - Targets a very minimal memory (RAM and ROM) footprint,
  - Implemented in C, with no other language bindings
  - TL depends on an AllJoyn Router running elsewhere, likely off device

Software Components: Router

Router

- The router may be bundled as part of the Standard Core Library, and so must run on an HLOS
- Can be deployed as a standalone daemon/service
  - Currently supported on Linux-based systems, such as OpenWRT
- The Router functionality consists of bus management and routing AllJoyn messages
  - Bus management includes
    - Managing the namespace: application addressing over the bus
    - Cross-device communication: discovering services and connecting to the AllJoyn Router supporting that service on behalf Apps
  - Message routing consists of delivering messages between applications, or to the router itself
    - AllJoyn control signaling also uses AllJoyn messages
- AllJoyn Router does the “heavy lifting” in the AllJoyn system
- Apps depend on AllJoyn Routers to interact with other Apps

AllJoyn SDK Concepts

Exposing Functionality

- AllJoyn applications expose their functionality via APIs implemented in objects
  - OOP approach
  - Object hierarchies are supported if required by application model
    - The AllJoyn framework will create parent objects automatically if application object is not the root
- Objects implement one or more interfaces
  - These are the method calls other AllJoyn applications interact with
- Interfaces are composed of members, which fall into three categories:
  - Methods – classic RMI/RPC object interaction
  - Signals – asynchronous event notification
    - Can be broadcast, multicast, or point-to-point
    - Can also send a Sessionless Signal: a broadcast signal that doesn't require an application session to be delivered
  - Properties – data members
    - These are accessed by built-in get/set methods
- All of this information can be introspected remotely

Advertise / Discover

“Find or Describe Capabilities”

- The About feature allows for publication of the interfaces used in an application
  - Interface is standardized: “org.alljoyn.About”
    - Contains a set of Signals and BusMethods to express what the application/device is and supports
    - Device information: make, model, manufacturer, etc.
    - Interface list: “org.alljoyn.controlpanel”, “org.allseen.media”, etc.
    - New 14.06 feature, Sessionless signal advertised as “org.alljoyn.About.sl” for optimization of network connections
  - 14.06 Next Generation Name Service
    - Used to detect presence
    - BusAttachment::Ping method allows single point query of availability (unicast)
    - Developer has control to check on demand
    - Core software now uses mDNS packets (unicast)
    - LostAdvertisedName wraps up the Ping at a 120-second interval
    - More robust detection when network changes

AllJoyn Interface

What is it?

- Defines the interactions that can occur from remote applications
  - Think of this as a header file of the distributed system
  - Can be introspected by remote Applications for dynamic integration

Collections of the following:

- Bus Method
  - Method call on a remote application
- Signal
  - Sent to all Applications in a session
- Property
  - Variable accessible via get/set
  - Developer chooses the access control

AllJoyn Interface

Data Types

All Bus Methods, Signals, and BusProperties can contain basic and complex data types

- Basic types
  - String (s), int (i), unsigned int (u), short (n), byte (y), double/float (d), long (x), and ObjectPath (o)
- Arrays
  - All data types can be set as an array
  - Defined by adding the character (a) before the data type, e.g., array of strings: (as)
- Structs
  - Struct is a collection of any combination of data types
  - Defined by using the characters ((and) or r), e.g., a contact could be (name, email, number): (ssi) or rssi
- Dictionary
  - Key-value pair in which the key can be any of the primitive types and the value any data type
  - Defined by using the characters ({ and } or e), e.g., a player table for a game could be: {sai} or eai

AllJoyn Interface

Data Types

- Variant
  - The AllJoyn interface provides for the ability of runtime data type detection
  - This in tandem with Introspection - it can enable new innovative experiences
  - Variants can be any type
  - Defined by the character (v), e.g., a customer record may contain various fields defined in an array (av)

  - Upon receipt of a Variant argument, the data can be understood by using the typeId field
    - Once the type is understood it can be pulled out of the object to be used

AllJoyn Interface

XML markup

- An interface can be described via an XML markup using the following format:
  - Interface tag: <interface name="[name]"> … </interface>
  - BusMethod tag: <method name="[name]"> … </interface>
  - Signal tag: <signal name="[name]"> … </signal>
  - Arguments to be used with BusMethods and Signal:

    <arg name="[name]" type="[data type]" direction="[in, out]"/>
  - Property tag: <property name="[name]" type="[data type]" access="[read, write, readwrite]"/>
  - BusObjects defined with collection of interfaces: <node name="[name]">

*text inside [ ] is strictly for show and would be an actual name, data type, or value-defined

AllJoyn Interface

About Feature as XML

```
<node name="/About"

     xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"


xsi:noNamespaceSchemaLocation="http://
www.allseenalliance.org/schemas/

      introspec t.xsd">

     <interface name="org.alljoyn.About">

          <property name="Version" type="q"
access="read"/>

          <method name="GetAboutData">

               <arg name="languageTag" type="s"
direction="in"/>

              <arg name="aboutData" type="a{sv}"
direction="out"/>

     </method>

          <method name="GetObjectDescription">

               <arg name="objectDescription" type="a(sas)"

                direction="out"/>

          </method>

          <signal name="Announce">

               <arg name="version" type="q"/>

               <arg name="port" type="q"/>

               <arg name="objectDescription" type="a(sas)"/>

               <arg name="metaData" type="a{sv}"/>

          </signal>

          </interface>

</node>
```

Code snippet from About Interface Specification: http://allseenalliance.org/docs-and-downloads/documentation/alljoyn-about-feature-10-interface-specification

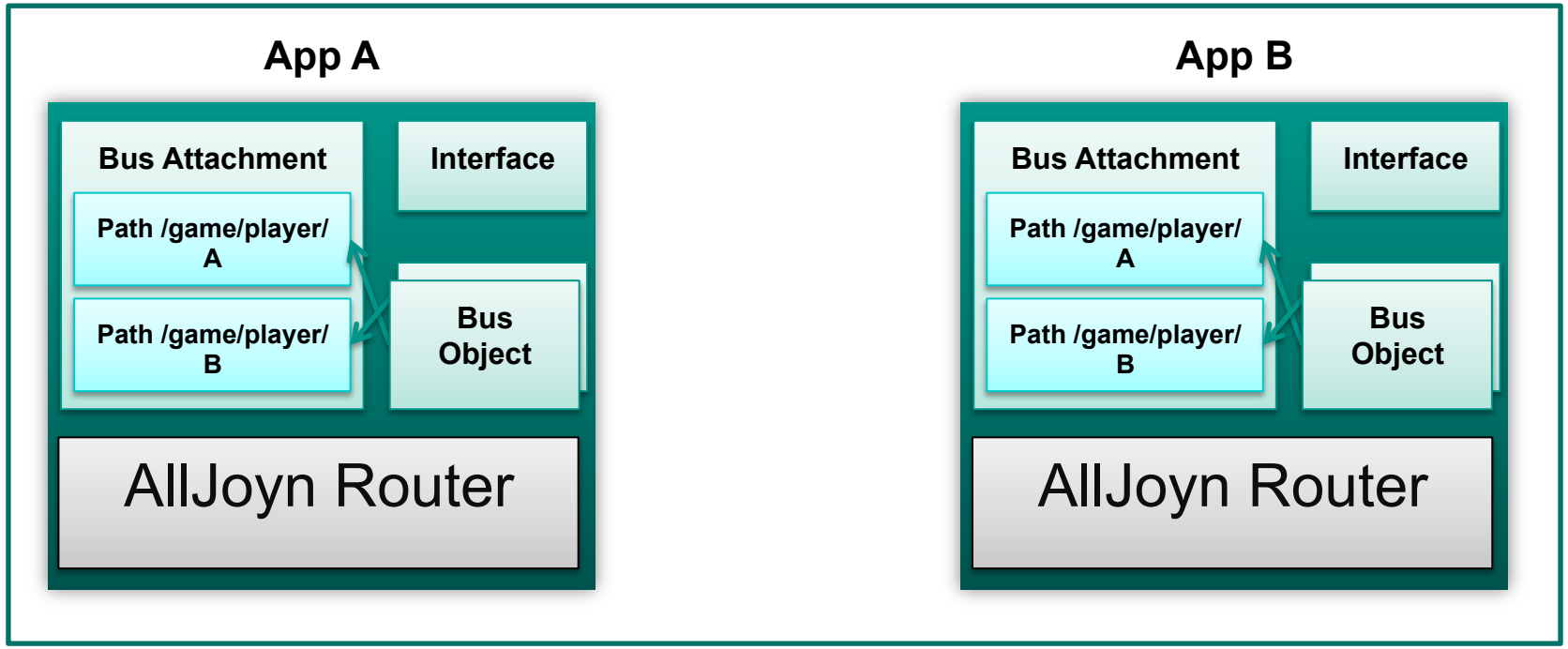
Bus Object -- Introduction

Base class that all applications use to expose functionality

- Bus Objects implement AllJoyn Interfaces
  - AllJoyn Interfaces are the definition of the service
  - Make up the API of interaction
- Method Calls are made by using Proxy Objects
  - This results in a method call message being sent to the object the proxy object represents
  - Method handler is invoked to execute method and generate the reply
  - The method reply is sent by the object back to the proxy object
- Signals are emitted by Objects and consumed by Signal Handlers
  - Sessionless signals work the same way, but do not require the application to create a session
    - They are broadcast and are delivered to any app interested in receiving sessionless signals
    - Only a single instance of a signal will be sent, i.e. if the same signal is emitted multiple times, only the last will be delivered
      - Useful for sending isochronous data, such as state updates

Bus Object -- Introduction

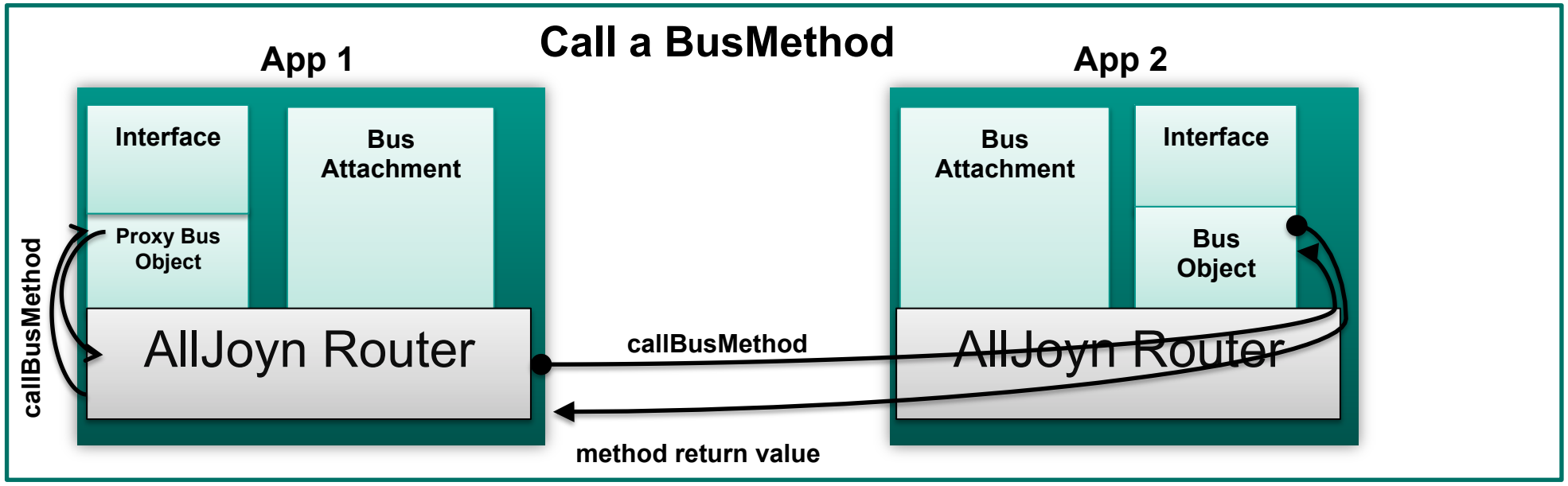
- Bus Objects Implement 1 or more of the Interfaces
- The Bus Object box represents 2 instances of the same Bus Object
- OOP style of programming allowing for Objects to be of the same type but semantically different



Each Bus Object is identical except that they exist on a different path.

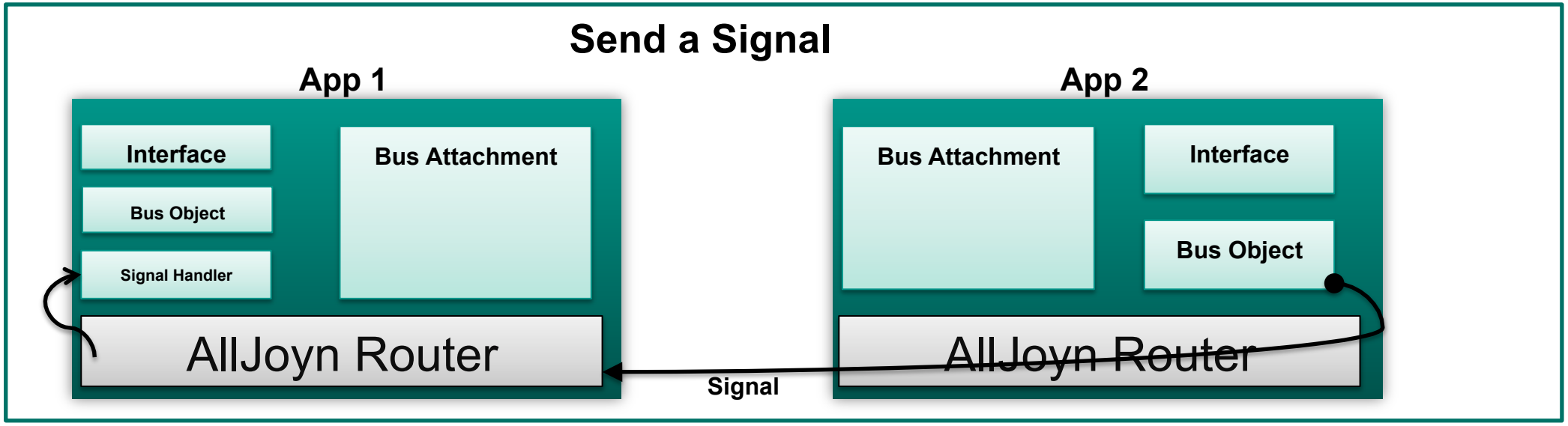
Bus Method

- Send data synchronously from one connected Application to another and receive a response
  - BusMethods are remote method calls that are blocking when executed (synchronous)
  - Function as a 1:1 interaction
  - A session must be established in order to call a Bus Method on another Application.
  - A Proxy Bus Object is used as a local representation of the object on remote Application



Signal

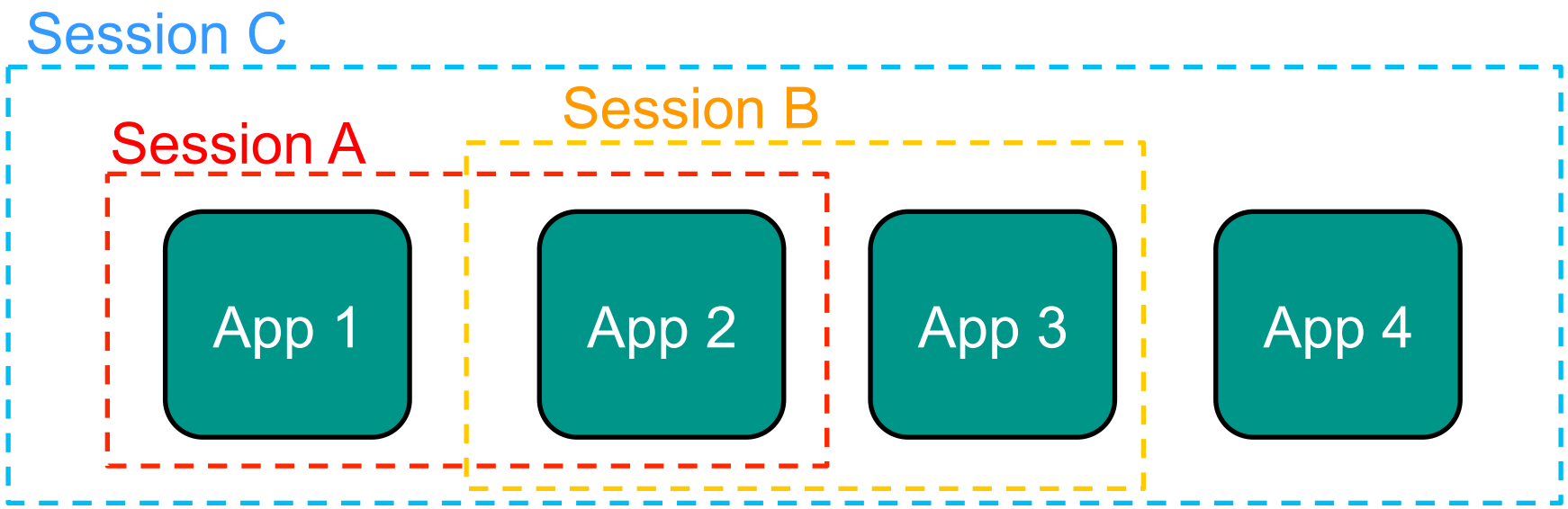
- Send data asynchronously from one connected Application to all Applications in a session
  - Signals are asynchronous and do not have a reply
  - Functions as a broadcast to all Applications
  - A session or AllJoyn Routers must be connected in order to transmit signals
  - Signals are sent from Bus Objects and received by the registered Signal Handler
  - Reliable delivery as long as other Applications have a connection



Session

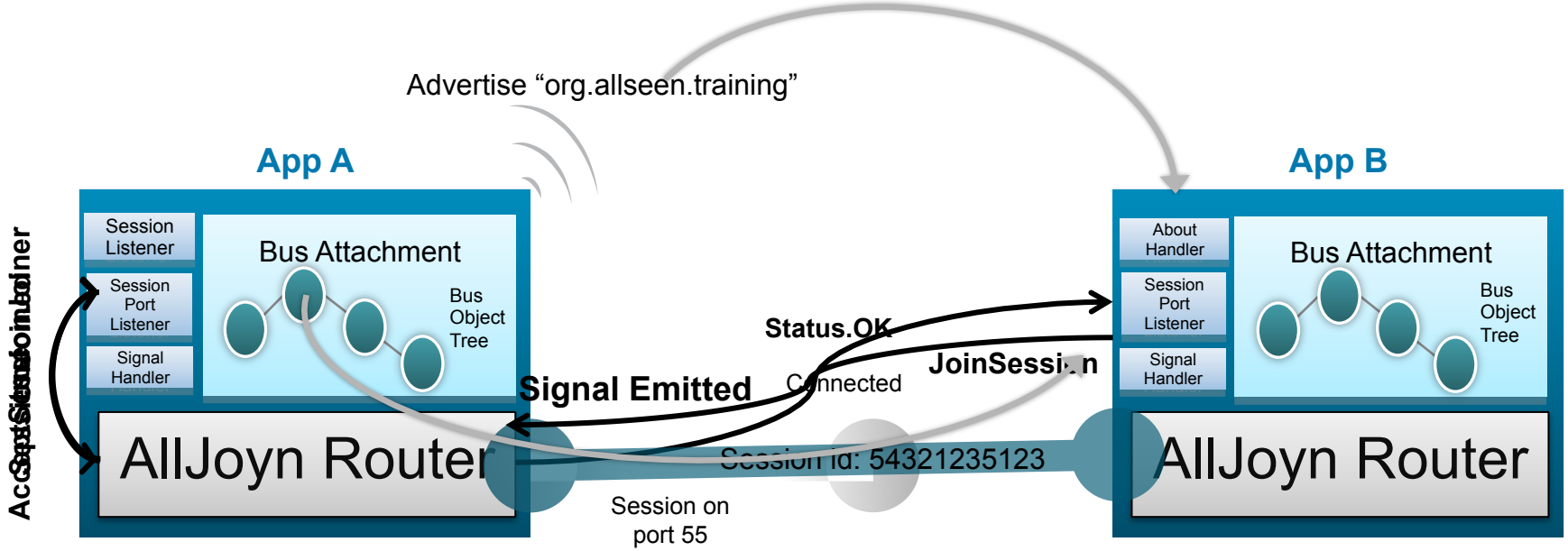
What is this conceptually?

- A group of applications that are connected, allowing them to exchange data
  - Any application can create a session (BusAttachment::BindSession)
  - Advertisement is the process of letting others know you can be joined
  - Can be 1:1 or 1:many
  - Allows for signals to travel to only those with the same session ID.



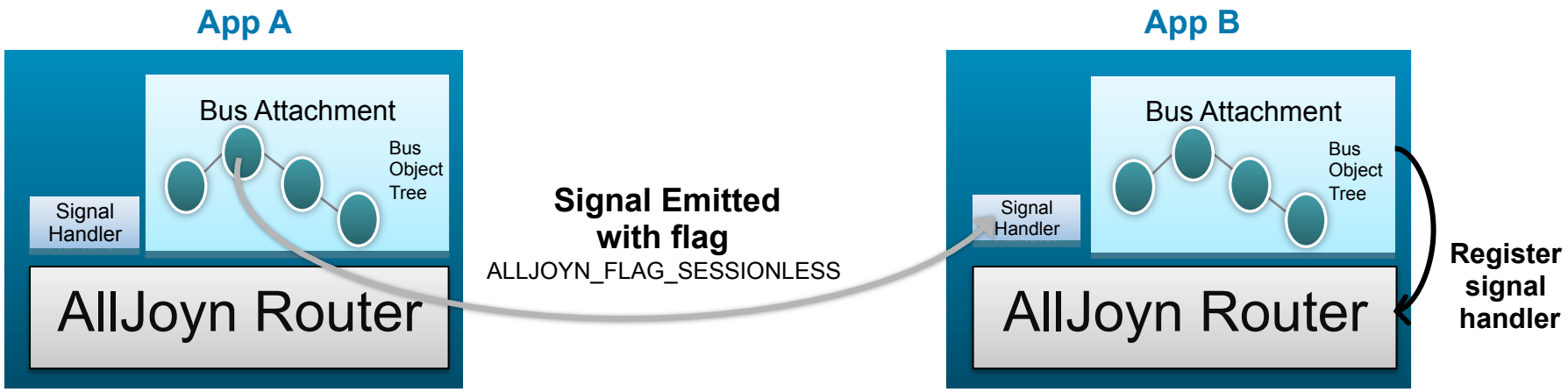
Session

Sends a Signal without needing to be in a Session



Sessionless Signal

Sends a Signal without needing to be in a Session



Sessionless Signal

Sends a Signal without needing to be in a Session

- Signal API call + ALLJOYN_FLAG_SESSIONLESS flag
  - About Feature and Notification Service Framework uses sessionless signals
  - Avoid complexity of which application advertises/joins and discovers/binds
  - Only to be used for small data transmissions that fit into a single signal
    - Not for file transfer or image transfer
  - Not intended for high-level traffic
    - Each signal overwrites the previous signal if not yet delivered

Base Services

AllJoyn Base Services for fundamental use cases

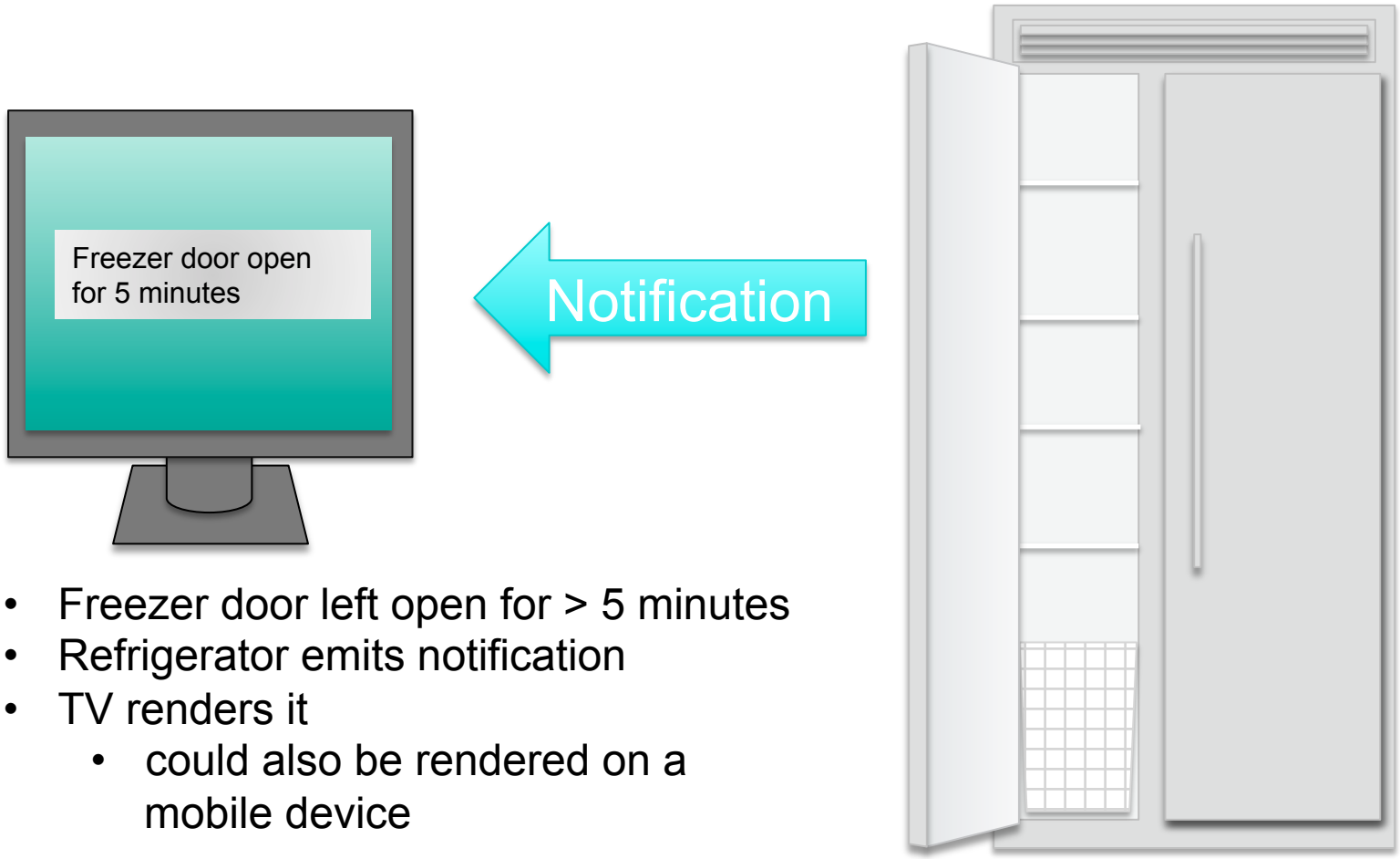
- Standard AllJoyn building blocks for generically useful services
  - Notification
  - Onboarding
  - Control Panel
  - Configuration
  - Audio
- Accelerate application and device development
- All services utilize the About feature for advertising and discovery

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
    - Wi-Fi, 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 could send a notification when wash cycle is complete

Notification Service Framework example



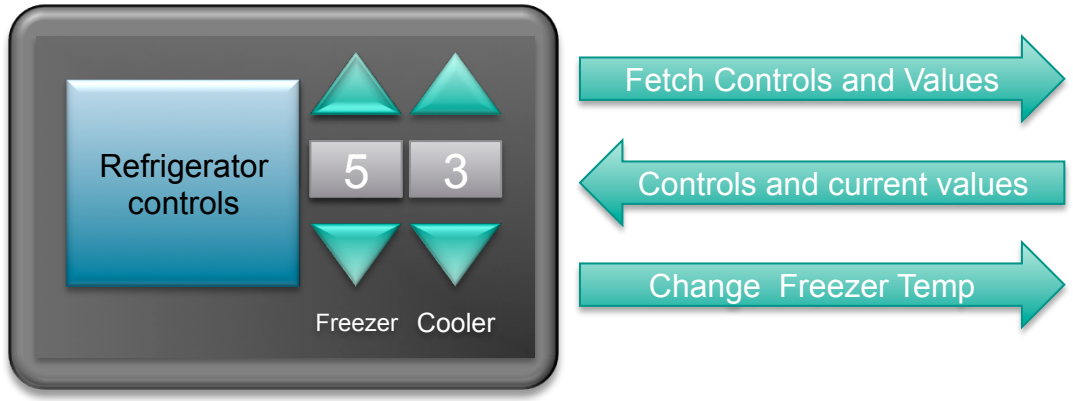
- Freezer door left open for > 5 minutes
- Refrigerator emits notification
- TV renders it
  - could also be rendered on a mobile device

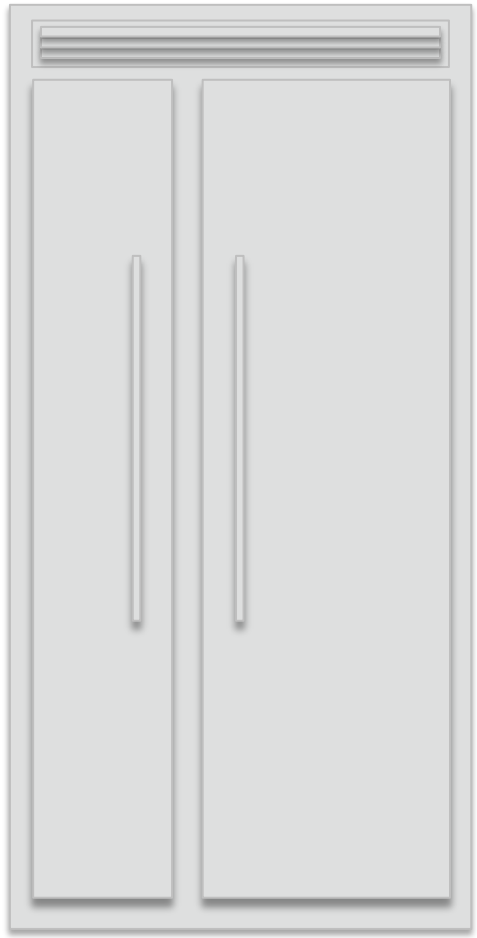
AllJoyn Control Panel Service Framework

- Infrastructure for exposing user interfaces for devices remotely
  - Model is there is a controller and a controllee
  - Controlee exposes its 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

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++(Standard Core) and C(Thin Core)
- 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 service framework combines Generated and Provided code

Events and Actions

Overview

- Events and Actions
  - Allows a consumer to understand what a device “sends” or “can do”
  - Events can be connected to 1 or more Actions
  - Connects devices in new ways from different manufacturers

- Event
  - A Signal with a human-readable description
  - The first part of a sentence

- Action
  - A BusMethod with a human-readable description
  - Forms the end of a sentence

New Introspection Interface

- org.allseen.Introspectable Interface
  - Added a new common Interface to every BusObject
  - Returns the Introspection XML but contains the addition of <description> tags
    - Only elements with a description set by the developer will be present
  - Allows ability to mark a Signal as being sessionless

```
<interface name="org.allseen.Introspectable">
   <method name="GetDescriptionLanguages">
          <arg name="languageTags" type=”as" direction=”out"/>
    </method>
    <method name="IntrospectWithDescription ">
       <arg name=”languageTag" type=”s” direction=”in"/>
       <arg name=”data" type=”s” direction="out"/>
    </method>
</interface>
```

API Changes

Each of the language bindings can expose Events and Actions

Standard Core Library (C++ and Objective C)

- After an Interface is created, use the following calls to add a description:
  - SetDescriptionLanguage
  - SetDescription
  - SetMemberDescription
  - SetArgDescription
  - SetPropertyDescription
  - SetDescriptionTranslator

API Changes

Each of the language bindings can expose Events and Actions

Standard Core Library (Java Binding)

- When defining the interface in the annotation add in ‘description=‘ fields
  - Ex:

```
…
…
@BusInterface (name = "org.samples.described",
               descriptionLanguage="en",
               description="This interface is described")
public interface DescribedInterface {

   @BusMethod (description="This is a method")
   public void myMethod();

   @BusSignal (description="This is a signal")
   public void mySignal();
…
…
```

Code snippet from:
https://git.allseenalliance.org/cgit/core/alljoyn.git/tree/alljoyn_java/samples/java/JavaSDKDoc/JavaSDKDocIntrospectWithDescriptionService/src/org/alljoyn/bus/samples/DescribedInterface.java

API Changes

Each of the language bindings can expose Events and Actions

Thin Core Library (C)

- Due to the nature of a Thin Core Library application being memory constrained and fixed in a device, the API is intended to be table-driven to read the values.
- There is a new encoding that has been created to map a developer entered description to an AllJoyn Interface Member.
  - AJ_DESCRIPTION_ID(BusObject base ID, Interface index, Member index, Arg index)
  - Register the supported description language:
    - AJ_RegisterDescriptionLanguages
  - Register the BusObject with the translator callback
    - AJ_RegisterObjectListWithDescriptions
  - Implement the Translator
    - typedef const char* (*AJ_DescriptionLookupFunc)(uint32_t descId, const char* lang);
  - For Debugging new Print API for an object list
    - AJ_PrintXMLWithDescriptions

Sample Application

Android Sample Application

Standard Library (C++)

- Exists precompiled in the AllJoyn SDK for Android
- AllJoyn SDK for Android contains the sample application to be used as a reference design
- Rule Engine software is capable of running by itself or inside the Android application.
  - Remote engine uses the AllJoyn Framework to communicate via an example interface
  - Remote engine does not persist rules
  - Remote engine offers tracking of devices and updates rules in memory
  - Intended as a sample, not the end solution

Sample Application

Android Sample Application

Standard Library (C++)

- To help showcase power of Events and Actions, the Android AllJoyn SDK contains a sample application with the following:
  - Browse the nearby Events and Actions
  - Track when devices come and go
  - Long press on an Event/Action to view Interface/BusObject details
  - Can save a “Rule” locally
    - “Rule” is a dynamic registration for the Event member Signal that when received will call the Action
  - Can connect to remote rule engine and set rules
  - Delete all rules

Sample Application

Android Sample Application

Standard Library (C++)

- Browse ability at a high level
  - Application starts up and looks for devices that advertise capabilities via the About Feature
  - Each AllJoyn entity found will cause an attempt to “IntrospectWithDescriptions”
    - Then parse the entire BusObject Tree and repeat with the next <node> element
  - If xml is returned, parse and store descriptions if found
- Rules at a high level
  - A rule is a registration for a Signal Handler and an invocation of a BusMethod
  - The devices are tracked via the About Feature for the uniqueName endpoints
  - Session created on-demand to execute BusMethod on “Action” AllJoyn entity

Why use Events and/or Actions?

Benefits

- Allows end consumers the ability to join products from different manufacturers
- End consumer can build new experiences greater than the original device intent
  - Lights can blink when someone sets off a motion sensor
  - Heater turns on when the temperature outside drops
  - Lights brighten when a camera is about to capture a photo
- No impact to single product experience
  - Already exposes APIs for a developer to interact with the device
  - Now exposes strings that can be understood easily
- Keeps focus on the product and not how it will interact with other products
  - Removes burden to explicitly focus on new innovative use cases
  - Lets end consumers use their creativity

Why use Events and/or Actions?

Drawbacks

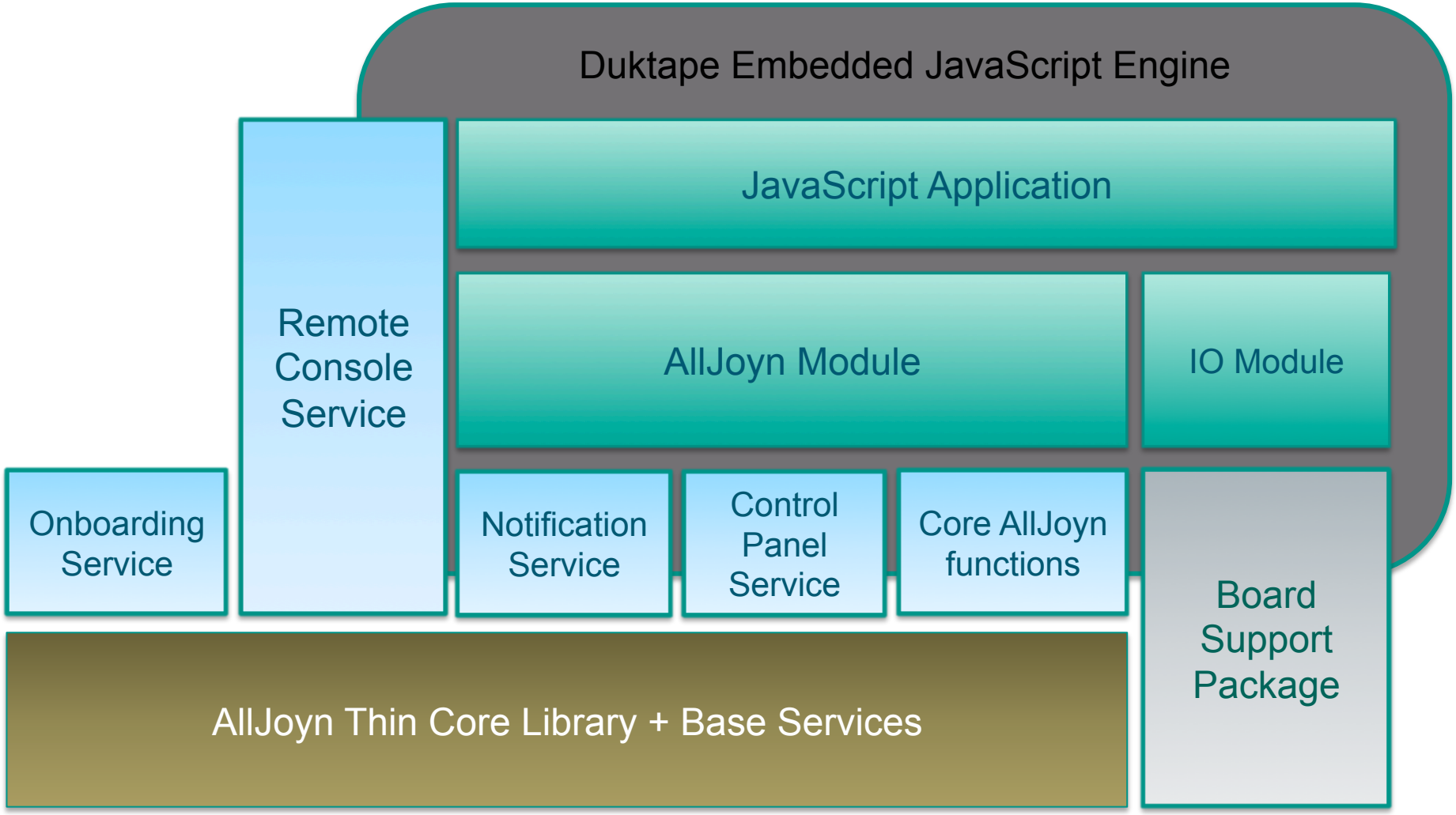
- Think through good descriptions to set for Events and Actions
- Add more specific signals and methods that guess at what an end user would want
- Descriptions must convey a part of a sentence enough to let an end user understand
- Product may interact with a competitor's product

AllJoyn.js

What is AllJoyn.js?

- AllJoyn.js allows the AllJoyn Thin Core Library (AJTCL) and base services with “duktape”, an open source small-footprint ECMAScript 5.0 compliant runtime engine.
  - For more information on “duktape” see www.duktape.org
- A set of JavaScript APIs provide an easy to use abstraction layer over the AllJoyn core, base services, and the device IO peripherals.
- The combined implementation is targeted at microcontrollers having a minimum of 128K RAM (preferably 256K for “real applications”) and 500K Flash.
- Also designed to run on Linux, Windows, and OSX
- Includes a “console” service for installing scripts and debugging application code.

AllJoyn.js Architecture



Console Service

- An AllJoyn service that runs alongside the JavaScript app
  - Functionality is exposed as an AllJoyn interface
  - AllJoyn.js source tree includes a command line console client

- Provides remote access to running JavaScript application
  - OTA flashing of new JavaScript applications
  - Execute JavaScript code on target in real-time
  - Logging of output from JavaScript print() and alert() functions
  - Displays notifications from running JavaScript program

Console Application

- The console application is a standalone AllJoyn application that communicates with an AllJoyn service running alongside the JavaScript application
- If called with a JavaScript file, the console application connects to installs a new application into a running AllJoyn.js instance
  - The previous application is overwritten
  - If there are errors running the script, they are output to the console
- If called without a JavaScript file, the console application connects to a running AllJoyn.js
- In either case, after connecting to the AllJoyn.js instance any input is sent to the JavaScript interpreter
  - This allows real-time interaction with the running JavaScript program

Example Console Interaction

```
Found script console service: :Zp5SKg6r.4
Joined session: 841438313
JSON.stringify(AJ)
Eval: JSON.stringify(AJ);
Eval result=0: {"interfaceDefinition":{"org.allseen.DoorBell":{"ding_dong":{"type":
1}}},"objectDefinition":{"/DoorBell":{"interfaces":["org.allseen.DoorBell"]}},"config":
{"linkTimeout":10000,"callTimeout":10000},"METHOD":0,"SIGNAL":1,"PROPERTY":
2,"defaultLanguage":"en"}
JSON.stringify(IO);
Eval: JSON.stringify(IO);
Eval result=0: {"pin12":{"id":12},"pin11":{"id":11},"pin10":{"id":10},"pin9":{"id":
9},"pin8":{"id":8},"pin7":{"id":7},"pin6":{"id":6},"pin5":{"id":5},"pin4":{"id":4},"pin3":
{"id":3},"pin2":{"id":2},"pin1":{"id":1},"openDrain":2,"pullUp":4,"pullDown":8,"risingEdge":
1,"fallingEdge":2}
IO.pin1.info.description
Eval: IO.pin1.info.description;
Eval result=0: Red LED
IO.pin1.functions
Eval: IO.pin1.functions;
Eval result=0: digitalOut
2+3
Eval: 2+3;
Eval result=0: 5
alert("Hello world")
Eval: alert("Hello world");
Hello world
Eval result=0: undefined
```

Programming model

- AllJoyn.js is 100% event driven.
  - No blocking calls
  - Write operations that cannot be buffered may introduce delays
- Functions registered with the AllJoyn object (AJ) are called when various AllJoyn events happen:
  - AllJoyn bus attachment events:

    `onAttach onDetach`
  - AllJoyn messages:

    `onSignal onMethodCall onPropSet onPropGet onPropGetAll`
- Functions can be registered with one-shot and interval timers

  `setTimeout clearTimeout setInterval resetInterval clearInterval`
- Functions can be registered to be called on input and output triggers
  - `setTrigger`

Debug output

- Duktape has two built-in functions for logging output to a debug console:

```
print()
alert()
```

- In AllJoyn.js, these do basically the same thing except the output string is prefixed with “PRINT” or “ALERT”
- However, if the console client is connected to the running JavaScript application, output is redirected and displayed by the console application.

Interface and Object definitions

- A definition is required for the interfaces and objects used by an AllJoyn.js application
- These definitions supply essential information required to send and receive signals, make and handle method calls, and access properties

```
AJ.interfaceDefinition[‘test.InterfaceA'] = {
    mySignal:{ type:AJ.SIGNAL, args:[’s’] },
    myProperty:{ type:AJ.PROPERTY, signature:’u’ },
    myMethod:{ type:AJ.METHOD, args:[‘i’, ‘i’], returns:[’i’] }
};

AJ.interfaceDefinition[‘org.example.Interface2’] = {
     /* signals, methods, and properties */
};

AJ.objectDefinition['/myApp'] = {
    interfaces:[‘test.InterfaceA’,‘org.examples.Interface2’]
};
```

Base Services

Base service integration

- AllJoyn.js currently integrates four base services:
  - Onboarding – gets a device onto a Wi-Fi network
  - Config – sets up authentication credentials, friendly name, etc.
  - Notifications – send text messages for human consumption
  - Control Panel – a generic UI toolkit

- Onboarding and Config are mostly transparent to JavaScript applications
  - Some config parameters can be read and set

- AllJoyn.js implements APIs to support Notifications and Control Panel

Notifications -- simple

- To create a notification with a message in the default language:

```
var notif = AJ.notification(<urgency>, "Hello World");
```

Urgency is one of the following constants:

```
AJ.notification.Emergency (=0)
AJ.notification.Warning   (=1)
AJ.notification.Info      (=2)
```

- To send the notification:

```
notif.send(<time-to-live>);
```

Time-to-live is the number of seconds that the notification will remain deliverable. *The notification service requires this to be at least 30 seconds and no more than 12 hours (4320 seconds).*

- The creation and send operations are separated out so that additional properties can be set on the notification before it is sent.

Notifications -- customized

- Additional properties can be set on the notification prior to calling send.
- Explicitly set the text to specify multiple languages for the notification

```
notif.text = {
    en:"Hello World",
    sp:"Hola Mundo“
};
```

- Associate an icon with the notification

```
notif.iconUrl = "http://url/to/icon";
notif.iconPath = “/notif/icon”;          // Object path on notif sender
```

- A notification can be canceled by the sender after it has been sent

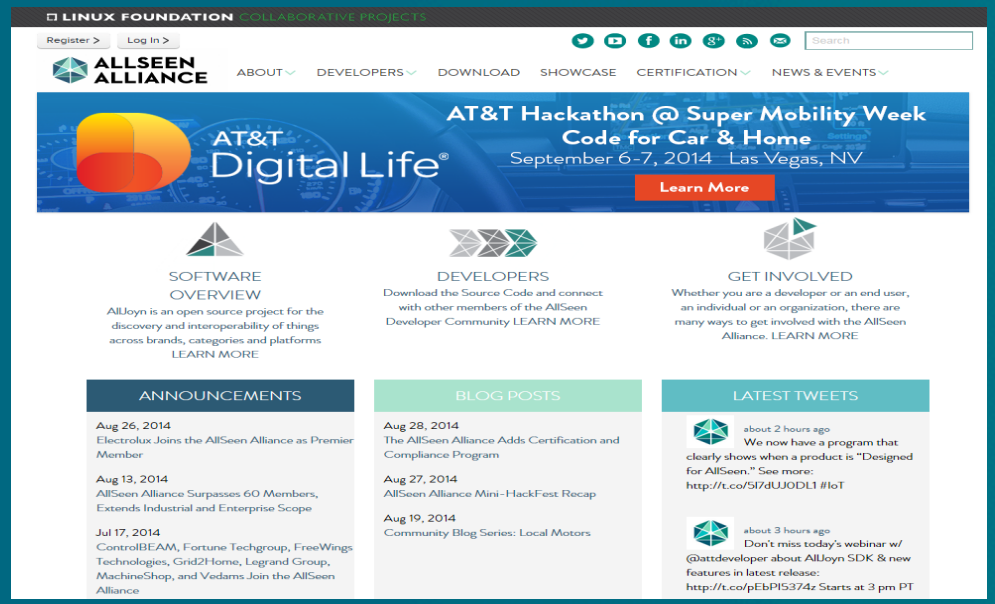
```
notif.cancel();
```

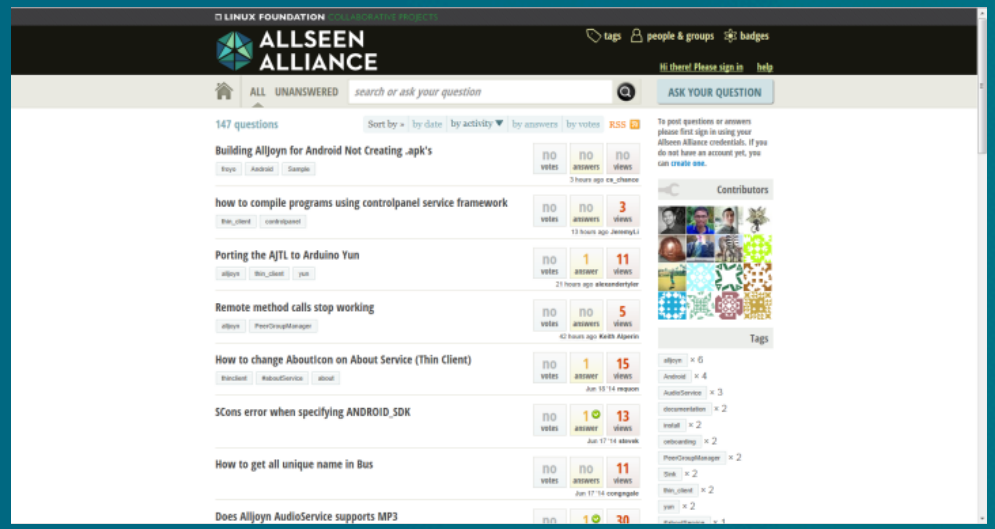
Control Panel Services

- The Control Panel service allows a headless application to expose a simple control panel built from a set of simple widgets.
  - An generic application running on a handset, tablet, or other device can render the UI without knowing anything about the device being controlled
- The JavaScript application creates a control panel, adds a top-level container widget and then adds various widgets that define the UI

```
var cp = AJ.controlPanel();
var root = cp.containerWidget();
var rate = root.propertyWidget(cp.SLIDER, 500, "Flash rate:");
rate.range = { min:20, max:1000, increment:50, units:"msec" };
var led = IO.digitalOut(IO.pin1);
var blinky = setInterval(function(){led.toggle()}, rate.value);
rate.onValueChanged = function(val){resetInterval(blinky, val)}
AJ.onAttach = function() { cp.load(); }
```

For More Information





Alliance Wiki -- https://wiki.allseenalliance.org

- Documents, downloads, and developer tools
- Source Code, release overviews, roadmaps
- Training & Service Framework details
- Working Groups, New Proposals & meeting minutes

Forums -- https://ask.allseenalliance.org/questions

Public Mail Lists -- https://lists.allseenalliance.org/mailman/listinfo

Showcase -- https://allseenalliance.org/showcase

Monthly Newsletter -- https://allseenalliance.org/news-and-events/newsletters

Next Steps

- Download the AllJoyn SDK for your development platform from http://www.allseenalliance.org.
- Start with the sample applications to ensure your environment is set up correctly.
- Pull down the Hackfest git project: https://git.allseenalliance.org/cgit/extras/hackfest.git/
  - Wiki page contains information on this project: https://wiki.allseenalliance.org/develop/hackfests
- Dive in and start hacking!
- If you have questions ASK us @ http://ask.allseenalliance.org
- Want to build from source? http://wiki.allseenalliance.org

Questions?

Thank You

Follow Us On

For more information on AllSeen Alliance, visit us at: allseenalliance.org & allseenalliance.org/news/blogs