AllJoyn™ Troubleshooting Guide

HT80-BA058-1 Rev A

October 3, 2012

Submit technical questions at:

http://www.alljoyn.org/forums

Qualcomm Innovation Center, Inc.
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Contents

Tables

1 Introduction

1.1 Scope

This document is for developers who want to troubleshoot issues with AllJoyn-enabled applications.

1.2 Purpose

This document includes common frequently asked questions and troubleshooting information for the AllJoyn application framework. These issues include:

I wrote an AllJoyn app in Android, and the discovery does not seem to work. Am I missing something?

How do I know when a peer is no longer available, or has moved out-of-range when I was talking to, or in a session with, that peer?

I have two or more devices/machines on which AllJoyn apps are running. I ran everything as instructed in the documentation. Why are my devices not seeing each other?

I installed the Java Android samples provided in the AllJoyn SDK, but they don't work on my device or emulator.

Where can I find Android Java samples using the AllJoyn SDK?

I am trying to advertise a name over AllJoyn but it gives me ER_BUS_REPLY_IS_ERROR_MESSAGE and an error "Invalid busname."

I am trying to send across a large amount of data, e.g., a picture or a file to a peer in session. What is the most efficient way to do this?

I wrote an app that uses AllJoyn, but my app or the AllJoyn piece hangs and I don't get any response when I am playing with different UI components in my app.

Can I use multiple bus attachments in my applications? Is it allowed? What are the drawbacks?

I installed one instance of an AllJoyn app on an Android emulator and another instance on a physical device/different machine/host machine/any other supported platform. Will these two be able to communicate?

I want to allow/disallow access to my session. How do I do this?

How do I create a single auto-joining session while avoiding a UI element to host/join a session?

1.3 Revision history

Table 1 provides the revision history for this document.

Table 1: Revision history

Version	Date	Description
A	October 2012	Initial release

2 FAQs

2.1 I wrote an AllJoyn app in Android, and the discovery does not seem to work. Am I missing something?

Check the setup.	Are the devices connected to the same access point?	The devices should be connected to the same access point for AllJoyn to work.
	Is the AllJoyn daemon running?	The AllJoyn daemon must be running for every app that uses AllJoyn. The daemon can be in standalone or bundled form.
	Is the access point conducive to a peer-to-peer network?	For AllJoyn to work on a WiFi network, it should have multi-cast packet routing enabled and wireless isolation turned off. AllJoyn handles the case of wireless isolation if you do not care which transport is used, but if you want strictly WiFi access, wireless isolation should be turned off.
Check for `AndroidManifest.xml`	The `AndroidManifest.xml`file must be in the app package.	Developers often look up AllJoyn samples to learn how to use AllJoyn. While getting the source code is usually correct, make sure the `AndroidManifest.xml` file is in the app package.
Check the app for four essential permissions.	`<uses-permission android:name= "android.permission.INTERNET"> </uses-permission>`	■ You need the first two permissions for AllJoyn discovery to work. ❐ The AllJoyn discovery mechanism sends out multicast packets. ❐ To send out multicast packets on Android through an app, add these two permissions to the `AndroidManifest.xml`file of the app.
	`<uses-permission android:name= "android.permission. CHANGE_WIFI_MULTICAST_STATE"> </uses-permission>`	
	`<uses-permission android:name= "android.permission.ACCESS_WIFI_STATE"> </uses-permission>`	■ You need the second two permissions when using AllJoyn 2.5 or above and want to use ICE, which is an alternate transport in AllJoyn. ❐ AllJoyn 2.5 has a feature called *proximity service* that determines when you are near someone, i.e., you are proximal to another AllJoyn service/client. ❐ One method for determining proximity is matching the access points that the devices see. If the devices see a common set of
	`<uses-permission android:name= "android.permission.CHANGE_WIFI_STATE"> </uses-permission>`	

		access points, it is safe to say that they are near each other. ❐ These two permissions allow an app to request and use the access point information provided by Android.

2.2 How do I know when a peer is no longer available, or has moved out-of-range when I was talking to, or in a session with, that peer?

2.2.1 Background

AllJoyn has three kinds of main listeners (excluding `AuthListener`)	`BusListener`	Has callbacks for `FoundAdvertisedName`, `NameOwnerChanged`, and `NameLost`
	`SessionPortListener`	Typically used by a peer that is hosting a session/service; you can have two callbacks inside your implementation of this listener. 1. `acceptSessionJoiner` - A service uses this callback to accept or reject peers that have sent it a request to join a session. 2. `sessionJoined` - The bus calls this callback when a client joins a service. This is called on the service side if it implements the `SessionPortListener`.
	`SessionListener`	This listener has callbacks that the service and client can implement to get notifications about who joined or left the session. 1. `sessionLost` - The bus calls this callback when the last member of the session has left. **Tip** Developers might look at the service as one of the members of the session and therefore think that as long as the service is up, the session is up. This seems true logically, but actually, a session must have two or more peers. Thus, having the service up does not necessarily mean there is a session. Further, on WiFi, if the session owner leaves, the session is still up, and communication can flow, but new users can now join. 2. `sessionMemberAdded` - The bus calls this callback when a member is added to a multipoint session. This more frequently helps the client keep track of who joined the session. 3. `sessionMemberRemoved` - Useful for tracking the member that left a session and can be used by the service and the client. The service typically has a way (using `sessionJoined` from the `SessionPortListener`) to find out who joined

		the session without using `sessionMemberAdded`. Use only `sessionMemberRemoved` to track who left the session.

2.2.2 Approach

Now that we covered the listeners, you might assume that this is easy, and that you just need to track the `sessionMemberRemoved` callbacks to know if a peer to whom you were talking has left the session. This is correct, but these steps can help considerably.

Is the link still up, or has it gone down since the peer moved out-of-range?	■ Every time two peers connect, i.e., are in a session, there is a link between them. A link between them implies a link between their respective daemons. ■ We are looking for a way to find out if the link is still up, or has gone down since the peer moved out-of-range. ■ We can use a timeout for the link, so that if the link has been inactive for a specified period of time, we can conclude that it has gone down.
Is there a way to set this timeout value manually, so that I don't have to wait too long?	Yes! You can manually set the link timeout using `SetLinkTimeout()`. **Tip** If set to a value < 40 seconds, the default is 40 seconds.
Why do we do this?	■ In a TCP connection, resources are consumed when we have to send out a probe checking the status of the link. ■ If we set a value of < 40 seconds, the battery is consumed at an unsatisfactory rate. ■ A developer might wonder why the battery is draining when no message is being sent. This is not AllJoyn-specific, but the nature of the TCP transport.
Are you using Bluetooth®?	■ `SetLinkTimeout` has no apparent effect because for Bluetooth, sending probing signals does not consume measurably more resources than maintaining the link. If the link is up, probing to establish whether the link is still up does not consume as many resources as it does in TCP.
When to set a link timeout?	■ Setting a link timeout when you establish a session would be ideal. ■ If you are waiting for a `sessionMemberRemoved` to be called for the peer that is no longer in the session because it has moved out-of-range, you have to wait at least 40 seconds (or even longer) if you have specified a longer time. If the peer had not gone out of the coverage area and had closed down cleanly, you would not have to wait 40 seconds and could have received the `sessionMemberRemoved` callback instantly.

2.3 I have two or more devices/machines on which AllJoyn apps are running. I ran everything as instructed in the documentation. Why are my devices not seeing each other?

1. Is the AllJoyn daemon running on both devices?

2. Are all the devices connected to the same Wi-Fi network? If using Bluetooth, make sure the devices are within range of each other.
3. Does your Wi-Fi network block multicast packets? (This is false in most cases, but especially true in office environments.)
4. Are you advertising a correctly formed, well-known name? Well-known names can contain letters, numbers, underscores (_), and a dot (.)
5. Are you discovering a prefix or the correct name on the client side?
6. If steps *1* through *6* check out, and you are still unable to discover the service on a device, ping the other device.

2.4 I installed the Java Android samples provided in the AllJoyn SDK, but they don't work on my device or emulator.

- The AllJoyn samples included in the SDK are built to run on devices or an emulator running Android version Froyo or above.
- An AllJoyn app running in an emulator cannot communicate with any other app outside the emulator. This is the way the emulator in Android is designed. It is not a restriction of AllJoyn.

2.5 Where can I find Android Java samples using the AllJoyn SDK?

- If you look inside the SDK you will find folders named **samples** and **java** at the topmost level of the SDK. **samples** contains AllJoyn samples written using JNI, whereas the AllJoyn part is written in native code.
- java/samples contains samples that are written using Java bindings for AllJoyn and are the ones you should be referring to.
- The reason apps written using JNI exist in the SDK is to demonstrate the capability to use native bindings for AllJoyn on Android.

2.6 I am trying to advertise a name over AllJoyn but it gives me ER_BUS_REPLY_IS_ERROR_MESSAGE and an error "Invalid busname."

AllJoyn uses the DBus wire protocol and thus has predefined rules on how to form names. Follow these rules when choosing a name to be advertised:

- Bus names that start with a colon (':') character are unique connection names. Other bus names are called well-known bus names.
- Bus names comprise one or more element(s) separated by a period ('.') character. All elements must contain at least one character.

- Each element must contain only the ASCII characters "[A-Z][a-z][0-9]_-". Only elements that are part of a unique connection name may begin with a digit; elements in other bus names must not begin with a digit.
- Bus names must contain at least one '.' (period) character (and thus at least two elements).
- Bus names must not begin with a '.' (period) character.
- Bus names must not exceed the maximum name length.

The same rules apply when requesting a well-known name on the bus.

2.7 I am trying to send across a large amount of data, e.g., a picture or a file to a peer in session. What is the most efficient way to do this?

AllJoyn has three ways to send data across to a peer

■ Method calls	Method calls suit short reply response interactions. However, for something like transferring a file, the overhead is greater for making a call and getting a response than it is for signals.
■ Signals	Signals are unidirectional. A sender just places data in the body of the signal and sends it. This is useful considering the maximum amount of data one can send in an AllJoyn message is 128 Kb. If the file size is smaller, any of the methods to send data are fine. But, if the file size is greater, a good practice is to break the data into chunks and send it using signals.
■ Raw sockets	Raw sockets is an evolving concept in AllJoyn. The idea is to obtain a raw socket to which you can write data, and the peer can read it as it would from a regular socket. When using Java bindings for AllJoyn, one thing to note is that the raw Java socket returned by the AllJoyn framework is a non-blocking socket, so one should avoid writing a large amount of data rapidly on this socket. Sending 255 bytes at a time is recommended.

2.8 I wrote an app that uses AllJoyn, but my app or the AllJoyn piece hangs and I don't get any response when I am playing with different UI components in my app.

- AllJoyn has both synchronous and asynchronous calls which means it can block on some calls. When we write an Android app, we typically have an activity class that handled the UI part. Doing an AllJoyn-related task in this main UI thread can lead to unpredictable wait periods where the app takes time to respond or does not update the UI components as expected.
- A highly recommended way while using AllJoyn in your Android app is to do all AllJoyn-related activities in a separate thread; in the context of Android, this is typically a `BusHandler`. This assures that all of the AllJoyn-related activities do not interfere with any other component of the Android app.

2.9 Can I use multiple bus attachments in my applications? Is it allowed? What are the drawbacks?

- A `BusAttachment` is a representation of the AllJoyn bus.
- Creating it is a heavy operation in terms of memory and other resources.
- Avoid creating multiple bus attachments unless there is a justification for doing so.
- A common guideline to follow is determining whether the types of functionalities provided by the AllJoyn piece in your app are distinct and unrelated. Suppose you have two completely different modules in your app that use AllJoyn for totally unrelated things. In this case, consider creating a separate `BusAttachment`. It should, however, be avoided as much as possible due to the overhead in creating and maintaining its lifecycle in an app.

2.10 I installed one instance of an AllJoyn app on an Android emulator and another instance on a physical device/different machine/host machine/any other supported platform. Will these two be able to communicate?

In terms of networking, the Android emulator acts like a closed black box. It does not let you form TCP connections outside the emulator. AllJoyn has few networking components like using multicast for discovering other devices that have AllJoyn running on them. This, coupled with other restrictions on the emulator, make it impossible for an AllJoyn app to talk to anything outside the emulator. You can always have multiple instances of AllJoyn apps talking to each other inside the emulator, but not across the emulator.

2.11 I want to allow/disallow access to my session. How do I do this?

AllJoyn provides the ability to allow/disallow access to a session in the `AcceptSessionJoiner` callback. The only information you have at this time is the `sessionPort` and `joinerId`, the user's `busId`. If you require more checks in order to allow access, gather information out-of-band.

Two suggestions follow:

- Build a table based on `nameOwnerChanged` to correlate `busId` with a `wellKnownName`.
- Have a connect and accept on a different session, communicate information to determine who the user is, then join the session to which you want to allow access.

2.12 How do I create a single auto-joining session while avoiding a UI element to host/join a session?

AllJoyn has a unique guid that is associated with a `busAttachment`. A simple algorithm is to append onto the advertisements "_"+ `<busAttachment>.getGlobalGUIDString()`. Then when you discover `wellKnownNames` use a simple algorithm of highest (or lowest) GUID value and join that session. Now all app instances have a way to use a single multi-point session.

Note If you build this type of system, keep in mind an edge case where devices are coming and going very frequently which means sessions are being joined/left very frequently. In this scenario, a single session that is auto-joined is not ideal. It's worth rethinking the application and providing a UI where users can select the sessions to join.