AllJoyn™ C# Chat Sample Walkthrough

HT80-BA048-1 Rev A

July 31, 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 shows how to use AllJoyn to build a Windows 8 UI-style Chat application in C# for the Windows 8/Windows RT platform. It illustrates how to use AllJoyn APIs to perform mobile peer-to-peer chat communication. The source code of the Chat sample can be found in Windows RT Sample Applications in *http://www.alljoyn.org/docs-and-downloads*.

The Chat application sample can either host a chat session or join a chat session created by a peer on another device. To host a chat session, users enter the channel name and click **Start Channel**. To join a chat session, users select a found channel and click **Join Channel**. After joining a chat session, users can click **Send Message** or press **Enter** to send the chat message via bus signal calls.

1.2 Audience

This guide is for developers who plan to create C#-based AllJoyn-enabled Windows 8 UI-style applications on Windows 8/WinRT.

1.3 Prerequisites

This guide assumes that you have set up the Windows 8 UI-style application development environment. Visual Studio Ultimate/Express 2012 is required.

1.4 Revision history

The table below provides the revision history for this document.

Table 1: Revision history

Version	Date	Description
A	July 2012	Initial release

2 Getting Started with the Sample Application

2.1 Create the chat project

Follow the *Microsoft Windows 8 UI application development guide*, to create a C#-based Windows 8 UI-style application with the project name 'chat.'

2.2 Declare application capabilities

To use AllJoyn, first declare the required capabilities in the `Package.appxmanifest` file. In addition, the Debug version of `AllJoyn.dll` directs the log output to a file named `alljoyn.log` in the `Libraries\Documents` directory, so add the file type association as follows:

1. In **Solution Explorer** of Visual Studio 2012, select the 'Chat' project.
2. Double click file `Package.appxmanifest`.
3. Select the **Capabilities** tab.
4. Check these boxes: **Documents Library Access**, **Home or Work Networking**, **Internet (Client & Server)**, and **Internet (Client)**.
5. Select the **Declarations** tab.
6. In the list of **Available Declarations:** select and add **File Type Associations**.
7. In **Properties**, set **Name** as '.log'.
8. Under **Supported file types**, set **Content type** as text/plain and **File type** as '.log'.
9. Press **Ctrl+S** to save the configuration changes.

2.3 Add reference to AllJoyn.dll

The AllJoyn core logic is built as a Win RT component `AllJoyn.dll` (which is part of the AllJoyn SDK [Beta] - Windows RT) thus the chat project has to add a reference to `AllJoyn.winmd`. Follow these steps:

1. In **Solution Explorer** of Visual Studio 2012, select the chat project.
2. Right-click **Reference**, and select **Add Reference**.
3. When prompted, select **Browse** and navigate to the directory where `AllJoyn.dll` and `AllJoyn.winmd` are located.
4. Select `AllJoyn.winmd` and click **OK**.
5. Declare the AllJoyn namespace in `MainPage.xml.cs:` using namespace AllJoyn.

2.4 AllJoyn initialization

1. Put the work of initializing AllJoyn in a separate task, so that it does not block the UI thread.

```
private void InitializeAllJoyn()
{
    Task t1 = new Task(() =>
   {
       …. ….
   }
   t1.Start();
}
```

2. (Optional) Enable debug info logging and set the debug level of each module if the Debug version of `AllJoyn.dll` is used.

```
Debug.UseOSLogging(true);
Debug.SetDebugLevel("ALL", 1);
Debug.SetDebugLevel("ALLJOYN", 7);
```

3. Create the `BusAttachment` object.

```
try
{
    bus = new BusAttachment(APPLICATION_NAME, true);
}
catch (Exception ex)
{
    QStatus stat = AllJoynException.FromCOMException(ex.HResult);
}
```

   When invoking AllJoyn API fails, AllJoyn will throw to report the reason of the failure. The application has to catch and handle the exception. Call the static method API `int QStatus AllJoynException.FromCOMException(int hresult)` to convert the exception's `Hresult` property to understandable `Qstatus`. In addition, in the Debug version of `AllJoyn.dll`, it provides the API `AllJoynException.GetExceptionMessage(int hresult)` to get a detailed message explaining the failure.

4. Start the `BusAttachment` object. This only begins the process of starting the bus. Sending and receiving messages cannot begin until the bus is connected.

```
bus.Start();
```

5. Connect `BusAttachment` to AllJoyn daemon Bus. The `connectspec` as `null:` denotes connecting to the daemon bundled in the lib `Alljoyn.dll`.

```
string connectSpec = "null:";
bus.ConnectAsync(connectSpec).AsTask().Wait();
```

6. Create a `Listeners` class object. The `Listeners` class is not an AllJoyn class. It is a composite class of `BusListener`, `SessionListener`, and `SessionPortListener`.

`Listeners` class defines the implicit operator, so the `Listeners` object can be passed to the AllJoyn API since it is implicitly cast to `BusListener`, `SessionListener`, or `SessionPortListener`.

```
busListeners = new Listeners(bus, this);
```

7. Register the `BusListener` to the BusAttachment

```
bus.RegisterBusListener(busListeners);
```

8. Create and register the `BusObject`. In the same way, `ChatSessionObject` class defines a static implicit operator to cast itself to `BusObject`.

```
chatService = new ChatSessionObject(bus, OBJECT_PATH, this);
bus.RegisterBusObject(chatService);
```

9. Start to discover service over the bus. The application can either start service discovery by providing the name prefix here or later. The name prefix is used for prefix matching. The service provider advertises its service via a well-known name. If the well-known name matches the prefix, the event `FoundAdvertisedName` will be triggered to notify the service availability.

```
bus.FindAdvertisedName(NamePrefix);
```

3 Walkthrough of the Sample Chat Application

3.1 Create ChatSessionObject class

This class does the work of creating a `BusObject` and adding the interface to associate with the `BusObject`. In the constructor, do the following:

1. Create an AllJoyn bus object. Register the bus object with a `BusAttachment`. Once registered, it becomes visible to external bus connections and can then respond to method calls and emit signals.

   ```
   busObject = new BusObject(bus, path, false);
   ```

2. Create an interface with the `BusAttachment`.

   ```
   InterfaceDescription[] ifaceArr = new InterfaceDescription[1];
   bus.CreateInterface(ChatServiceInterfaceName, ifaceArr, false);
   ```

3. Add the definition of signal member “Chat” to the created interface, and activate the interface. An interface must be activated before it can be used. Activating an interface locks the interface so that it can no longer be modified.

   ```
   ifaceArr[0].AddSignal("Chat", "s", "str", 0, string.Empty);
   ifaceArr[0].Activate();
   ```

   Note AllJoyn also can initialize one more interface description from an XML string in DBus introspection format. For example, Steps *2* and *3* can be replaced with:

   ```
       private const string _chatInterfaceXml = "<interface
   name=\"org.alljoyn.bus.samples.chat\">" +
                       "<signal name=\"Chat\">" +
                       "<arg name=\"srt\" type=\"s\"/>" +
                       "</signal>" +
                       "</interface>";
       bus.CreateInterfacesFromXml(_chatInterfaceXml);
   ```

4. Retrieve the existing activated `InterfaceDescription` and add it to the `BusObject`. Note that once an object is registered, it should not add any additional interfaces.

   ```
   chatIfc = bus.GetInterface(ChatServiceInterfaceName);
   busObject.AddInterface(chatIfc);
   ```

5. Create a signal handler. The handler is called by the AllJoyn library to forward AllJoyn received signals to AllJoyn library users.

   ```
   chatSignalReceiver = new MessageReceiver(bus);
   chatSignalReceiver.SignalHandler += new
        MessageReceiverSignalHandler(this.ChatSignalHandler);
   ```

6. Register the signal handler to the “Chat” signal.

```
chatSignalMember = chatIfc.GetMember("Chat");
bus.RegisterSignalHandler(this.chatSignalReceiver, this.chatSignalMember,
path);
```

3.2 Create Listeners class

The `Listeners` class is essentially a composite of `BusListener`, `SessionListener`, and `SessionPortListener`. `BusListener` is called by AllJoyn to inform users of bus-related events. `SessionPortListener` receives session port-related event information. AllJoyn calls `SessionListener` to inform users of session-related events.

Some of the most important AllJoyn events include:

```
BusListeners.FoundAdvertisedName
BusListeners.LostAdvertisedName
SessionPortListener.AcceptSessionJoiner
SessionPortListener.SessionJoined
SessionListener.SessionLost
```

3.3 Start a chat session

The chat sample can either host a chat session or join a chat session created by a peer. Click **Start Channel** in the sample to start hosting a chat session.

1. Request a well-known name on the daemon bus.

```
string wellKnownName = this.MakeWellKnownName(name);
this.bus.RequestName(wellKnownName,
(int)RequestNameType.DBUS_NAME_DO_NOT_QUEUE);
```

2. Advertise the well-known name. Here, `TRANSPORT_ANY` means it will use any transport that is available.

```
this.bus.AdvertiseName(wellKnownName, TransportMaskType.TRANSPORT_ANY);
```

3. Call `BindSessionPort()` to create a new session and make a `SessionPort` available for external `BusAttachments` to join. A `SessionPort` and bus name form a unique identifier that `BusAttachments` use when joining a session.

```
ushort[] portOut = new ushort[1];
SessionOpts opts = new SessionOpts(TrafficType.TRAFFIC_MESSAGES, true,
       ProximityType.PROXIMITY_ANY, TransportMaskType.TRANSPORT_ANY);
this.bus.BindSessionPort(this.MakeWellKnownName(name);, portOut, opts,
this.busListeners);
```

4. When a peer finds the service and joins this chat session, the event `AcceptSessionJoiner` is triggered. In the event handler `SessionPortListenerAcceptSessionJoiner()`, ‘true’ is returned if the peer is allowed to join the chat session; after that, event `SessionJoined` is triggered to notify that the

peer has joined the session. In the event handler `SessionPortListenerSessionJoined()`, get the session ID assigned for this chat session. From then on, the chat sample can communicate with the peer.

```
if (this.hostPage != null && this.hostPage.SessionId == 0)
{
    this.hostPage.SessionId = id;
}
```

3.4 Join a chat session

The chat sample can join a chat session when it finds an advertised well-known name.

1. Before joining the session, the application registers an event handler for the `SessionJoined` event, which is triggered when the join process has finished. This code is in the Listeners constructor.

   ```
   this.sessionPortListener.SessionJoined += new
   SessionPortListenerSessionJoinedHandler(this.SessionPortListenerSessionJoined);
   ```

2. Call `JoinSessionAsync()` to join the session asynchronously. Note that this API does not return an Async operation as a continuation. Instead, the completion callback is registered as an event handler as described in Step *1*.

   ```
   string wellKnownName = this.MakeWellKnownName(name);
   SessionOpts opts = new SessionOpts();
   SessionOpts[] opts_out = new SessionOpts[1];
   JoinSessionResult result = await
   this.bus.JoinSessionAsync(bus.JoinSessionAsync(wellKnownName, ContactPort,
   this.busListeners, opts, opts_out, null);
   ```

3. Handle the join session result in the `JoinSession` event handler.

   ```
   private void SessionPortListenerSessionJoined (ushort sessionPort, uint id,
   string joiner)
   {
       if (this.hostPage != null && this.hostPage.SessionId == 0)
       {
           this.hostPage.SessionId = id;
   ```

3.5 Stop a chat session

To stop hosting the chat session, the application must cancel the advertised name, unbind the session port, and release the well-known name on the daemon bus.

```
var wellKnownName = this.MakeWellKnownName(this.channelHosted);
this.bus.CancelAdvertiseName(wellKnownName, TransportMaskType.TRANSPORT_ANY);
this.bus.UnbindSessionPort(ContactPort);
this.bus.ReleaseName(wellKnownName);
```

3.6 Leave a chat session

To leave a chat session after joining it, the application must call `LeaveSession()` with the corresponding session ID.

```
this.bus.LeaveSession(this.SessionId);
```

3.7 Send chat message via signal call

Using `Signal()` method in the `BusObject` we can send a chat message. If the first argument is an empty string it means that this message will be broadcast to all members in the chat session.

```
public void SendChatSignal(uint id, string msg)
{
    MsgArg msgarg = new MsgArg("s", new object[] { msg });
    this.busObject.Signal(string.Empty, id, this.chatSignalMember, new
MsgArg[] { msgarg }, 0, 0);
}
```