

AllJoyn™ WinJS Chat Sample Walkthrough

December 10, 2013

This work is licensed under a Creative Commons Attribution 4.0 International License.

<http://creativecommons.org/licenses/by/4.0/>

AllJoyn is a trademark of Qualcomm Innovation Center, Inc. AllJoyn is used here with permission to identify unmodified materials originating in the AllJoyn project.

Contents

1 Introduction.....	4
1.1 Scope.....	4
1.2 Audience.....	4
1.3 Prerequisites.....	4
1.4 Related documents.....	4
2 Getting Started with the Sample Application.....	5
2.1 Declare capabilities.....	5
2.2 Add reference to AllJoyn.dll.....	6
2.3 Add reference to Windows Visual C++ Runtime Package.....	7
2.4 Casing and naming rules in WinJS.....	7
2.5 Register AllJoyn bus event handlers.....	7
2.6 AllJoyn initialization.....	8
3 Walkthrough of the Sample Chat Application.....	10
3.1 Create ChatSessionObject class.....	10
3.2 Create Listeners class.....	11
3.3 Retrieve WinJS event arguments.....	11
3.4 Start a chat session.....	12
3.5 Join a chat session.....	12
3.6 Stop a chat session.....	13
3.7 Leave a chat session.....	14
3.8 Send chat message using signal call.....	14
3.9 Handle Suspending/Resuming event of the application.....	14
3.10 Enable Location permission for the application.....	15

Figures

Figure 1: Declare application capabilities.....	5
Figure 2: Declare file type association for debug logging.....	6

Tables

Table 1: Related documents..... 4

1 Introduction

1.1 Scope

This document shows how to use AllJoyn to build a Windows Store application chat in JavaScript on the Windows 8/Windows RT platform. It describes how to use AllJoyn APIs to do mobile peer-to-peer chat communication. The source code of the Chat sample can be found in AllJoyn Sample Apps - Windows 8 in www.allseenalliance.org.

The Chat application sample described in this document 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 using bus signal calls.

1.2 Audience

This guide is for developers who plan to create JavaScript-based AllJoyn-enabled Windows Store applications on Windows 8/Windows RT.

1.3 Prerequisites

This guide assumes that you have set up the Windows Store application development environment. Visual Studio Professional/Express 2012 is required. This guide also assumes that you know how to call a WinRT component from JavaScript.

1.4 Related documents

Table 1: Related documents

Document	Location
Introduction to AllJoyn	www.allseenalliance.org
AllJoyn Programming Guide for WinRT Platform	www.allseenalliance.org

2 Getting Started with the Sample Application

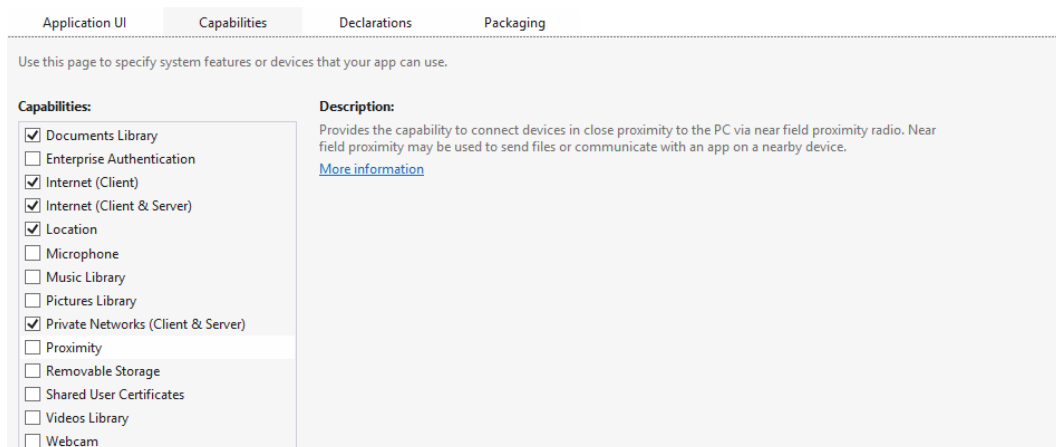
Follow the Microsoft Windows 8 UI application development guide to create a C#-based Windows 8 Store application with the project name 'chat.'

2.1 Declare 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 the file `Package.appxmanifest`.
3. Select the **Capabilities** tab.
4. Check these boxes:
 - **Documents Library**
 - **Internet (Client & Server)**
 - **Internet (Client)**
 - **Location**
 - **Private Networks (Client & Server)**

The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties.



Application UI Capabilities Declarations Packaging

Use this page to specify system features or devices that your app can use.

Capabilities:	Description:
<input checked="" type="checkbox"/> Documents Library	<p>Provides the capability to connect devices in close proximity to the PC via near field proximity radio. Near field proximity may be used to send files or communicate with an app on a nearby device.</p> <p>More information</p>
<input type="checkbox"/> Enterprise Authentication	
<input checked="" type="checkbox"/> Internet (Client)	
<input checked="" type="checkbox"/> Internet (Client & Server)	
<input checked="" type="checkbox"/> Location	
<input type="checkbox"/> Microphone	
<input type="checkbox"/> Music Library	
<input type="checkbox"/> Pictures Library	
<input checked="" type="checkbox"/> Private Networks (Client & Server)	
<input type="checkbox"/> Proximity	
<input type="checkbox"/> Removable Storage	
<input type="checkbox"/> Shared User Certificates	
<input type="checkbox"/> Videos Library	
<input type="checkbox"/> Webcam	

Figure 1: Declare application capabilities

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.

The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the p

The screenshot shows the 'Declarations' tab in the Windows Manifest Designer. The 'Available Declarations' list on the left contains 'File Type Associations'. The 'Supported Declarations' list on the right also contains 'File Type Associations'. The 'Description' section states: 'Registers file type associations, such as .jpeg, on behalf of the app. Multiple instances of this declaration are allowed in each app. [More information](#)'. The 'Properties' section includes fields for 'Display name', 'Logo', 'Info tip', and 'Name' (set to '.log'). Under 'Edit flags', there are checkboxes for 'Open is safe' and 'Always unsafe'. The 'Supported file types' section has a note: 'At least one file type must be supported. Enter at least one file type; for example, ".jpg"'. Below this, a table shows one entry with 'Content type' as 'text/plain' and 'File type' as '.log'. At the bottom, there are 'App settings' and an 'Executable' field.

Figure 2: Declare file type association for debug logging

2.2 Add reference to AllJoyn.dll

The AllJoyn core logic is built as a WinRT component `AllJoyn.dll` and can be accessed from C#, JavaScript, and C++. The chat project has to add a reference to `AllJoyn.winmd` to use it. 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. Use the namespace 'AllJoyn' to call APIs in AllJoyn, for example:


```
bus = new AllJoyn.BusAttachment(APPLICATION_NAME, true, 4);
```

2.3 Add reference to Windows Visual C++ Runtime Package

The AllJoyn runtime component depends on the Windows C++ Runtime Package, so the application must reference Windows Visual Runtime Package, otherwise it may cause exception "(0xc0000005) Access violation."

Follow these steps to add the reference:

1. In **Solution Explorer** of Visual Studio 2012, select the project.
2. Right-click **Reference** and select **Add Reference**.
3. When prompted, select **Windows** and then **Extensions**.
4. Check Windows Visual C++ Runtime Package.
5. Click **OK** to save.

2.4 Casing and naming rules in WinJS

Because WinJS is case-sensitive (see [Using the Windows Runtime in JavaScript](#)), you must use the following casing conventions in WinJS programming with WinRT components:

1. Namespaces and class names are Pascal-cased, for example, `AllJoyn.BusAttachment`.
2. Class member names are camel-cased, so the first letter of each member name is changed from uppercase to lowercase, for example, `bus.bindSessionPort()`.
3. Event names are always all lowercase. For example, JavaScript should use the event name `foundadvertisename`, while it is `FoundAdvertisedName` in the Windows Runtime component.
4. A series of initial uppercase letters normally appears as lowercase. For example, `AllJoyn.TrafficType.traffic_MESSAGES`. But, if three uppercase letters are followed by a lowercase letter, only the first two letters appear in lowercase. For example, a member named `IDString` appears as `idString`.

This transformation is part of the support that JavaScript provides to enable the natural use of the Windows Runtime. In Visual Studio, you can use IntelliSense in your JavaScript project to see the correct casing.

2.5 Register AllJoyn bus event handlers

There are two ways to register an event handler to AllJoyn events:

1. Use `addEventListener()`, as in the following example:

```
var bl = AllJoyn.BusListener(bus);  
bl.addEventListener('foundadvertisename', BusListenerFoundAdvertisedName);
```

2. Use 'on' plus the event name, as in the following example:

```
var bl = AllJoyn.BusListener(bus);
bl.onfoundadvertisedname = BusListenerFoundAdvertisedName;
```

- Note** When registering event handlers, all the characters in the event name should be lowercase, regardless of how many words the name contains.
- Note** The event handler must be defined before registering it to the event. For example, the handler `BusListenerFoundAdvertisedName` must be defined before you register the `foundadvertisedname` event; otherwise the event handler is undefined.

2.6 AllJoyn initialization

1. (Optional) If the Debug version of `AllJoyn.dll` is used, you can enable debug info logging and set the debug level of each module as described below. "ALL" means the debug level applies to all components. The native log output is stored in a file named `alljoyn.log` in the `Libraries\Documents` directory.

```
AllJoyn.Debug.useOSLogging(true);
AllJoyn.Debug.setDebugLevel("ALL", 1);
AllJoyn.Debug.setDebugLevel("ALLJOYN", 7);
```

2. Set the coercion type mode as "weak," so that AllJoyn will map JS weak type to AllJoyn strict data type.

```
AllJoyn.MsgArg.setTypeCoercionMode("weak");
```

3. Create the `BusAttachment` object.

```
try
{
    bus = new AllJoyn.BusAttachment(APPLICATION_NAME, true, 4);
}
catch (err)
{
    var errCode = AllJoyn.AllJoynException.getErrorCode(ex.HResult);
    var errMsg = AllJoyn.AllJoynException.getErrorMessage(ex.HResult);
}
```

When calling to the AllJoyn API fails, AllJoyn will throw an exception to report the failure. The application has to catch and handle the exception. Call the static method `API int QStatus AllJoyn.AllJoynException.getErrorCode(int hresult)` to convert the exception's `Hresult` property to understandable `QStatus` code. Call `API AllJoyn.AllJoynException.getErrorMessage(int hresult)` to get the 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 to the AllJoyn daemon bus. Connect spec as "null:" denotes connecting to the daemon bundled in the lib Alljoyn.dll.

```
var connectSpec = "null:";
bus.connectAsync(connectSpec).then(function ()
{
    bus.findAdvertisedName(NAME_PREFIX);
});
```

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

```
busListener = new Listeners(bus);
```

7. Register the `BusListener` to the `BusAttachment`.

```
bus.registerBusListener(busListener._bl);
```

8. Create and register the `BusObject`.

```
sessionTestObj = new ChatSessionObject(bus, OBJECT_PATH);
bus.registerBusObject(sessionTestObj._bsObject);
```

9. (Optional) Start to discover service over the bus. The application can 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 using 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(NAME_PREFIX);
```

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`. When registered, it becomes visible to the external bus connections and can respond to method calls and emit signals.

```
this._bsObject = new AllJoyn.BusObject(bus, path, false);
```

2. Create an interface with the `BusAttachment`.

```
var chatIfacArr = new Array(1);
var chatIfac = null;
bus.createInterface(CHAT_SERVICE_INTERFACE_NAME, chatIfacArr, false);
```

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

```
chatIfac = chatIfacArr[0];
chatIfac.addSignal("Chat", "s", "str", 0, "");
chatIfac.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:

```
var chatIfaceXml = "<interface name=\"org.alljoyn.bus.samples.chat\">" +
    "<signal name=\"Chat\">" +
    "<arg name=\"str\" 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.

```
this.chatIfac = bus.getInterface(CHAT_SERVICE_INTERFACE_NAME);
this._bsObject.addInterface(this.chatIfac);
```

5. Create a `signalhandler`. The handler is called by the AllJoyn library to forward AllJoyn received signals to AllJoyn library users. It is important for the application to keep a reference to the `MessageReceiver` object so that it will not be garbage-collected by the system.

```
this.chatSignalReceiver = new AllJoyn.MessageReceiver(bus);
this.chatSignalReceiver.addEventListener('signalhandler',
this.chatSignalHandler);
```

6. Register the `signalhandler` to the "Chat" signal member.

```
this.chatSignalMember = this.chatIfac.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:

- `BusListener.foundadvertisedname`
- `BusListener.lostAdvertisedName`
- `SessionPortListener.acceptsessionjoiner`
- `SessionPortListener.sessionjoined`
- `SessionListener.sessionlost`

3.3 Retrieve WinJS event arguments

In WinJS, all event handlers have only one argument, `evt`. The first event argument is stored in `evt.target`, and the rest of the arguments are stored in `evt.detail` as an array. To retrieve the actual event arguments, we define a helper function as follows:

```
// A helper function
function GetEventArg(evt, index) {
    if (index == 0)
        return evt.target;
    else if (index > 0)
        return evt.detail[(index - 1)];
    else
        return nullptr;
};
```

Example:

```
BusListenerNameOwnerChanged = function (evt) {
    var name = GetEventArg(evt, 0);
    var previousOwner = GetEventArg(evt, 1);
    var newOwner = GetEventArg(evt, 2);
};
```

3.4 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. 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.

```
var sessionOptIn = new
    AllJoyn.SessionOpts(AllJoyn.TrafficType.traffic_MESSAGES, true,
        AllJoyn.ProximityType.proximity_ANY,
        AllJoyn.TransportMaskType.transport_ANY)
var sessionPortOut = new Array(1);
bus.bindSessionPort(CONTACT_PORT, sessionPortOut, sessionOptIn,
    busListener._spListener);
```

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

```
bus.requestName(wellKnownName,
    AllJoyn.RequestNameType.dbus_NAME_DO_NOT_QUEUE);
```

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

```
bus.advertiseName(wellKnownName, AllJoyn.TransportMaskType.transport_ANY);
```

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.

```
this.SessionPortListenerSessionJoined = function (evt) {
    var sessionPort = GetEventArg(evt, 0);
    if (alljoyn.sessionId == 0) {
        alljoyn.sessionId = GetEventArg(evt, 1);
    }
    var joiner = GetEventArg(evt, 2);
}
```

3.5 Join a chat session

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

Call `joinSessionAsync()` to join the session asynchronously.

```
var sessionOptsArray = new Array(1);
var opts = new AllJoyn.SessionOpts(AllJoyn.TrafficType.traffic_MESSAGES,
    false, AllJoyn.ProximityType.proximity_ANY,
```

```
        AllJoyn.TransportMaskType.transport_ANY);
bus.joinSessionAsync(wellKnownName, CONTACT_PORT,
busListener._sListener, opts, sessionOptsArray, null).
then(function (reply) {
    var status = reply.status;
    sessionId = reply.sessionId;
    if (status ==0) {
        onjoined ();
    } else {
        onerror(status);
    }
});
```

3.6 Stop a chat session

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

```
that.stopChannel = function ()
{
    var wellKnownName = NAME_PREFIX + '.' + myHostChannel;
    bus.cancelAdvertiseName(wellKnownName,
        AllJoyn.TransportMaskType.transport_ANY);
    bus.unbindSessionPort(CONTACT_PORT);
    bus.releaseName(wellKnownName);
    sessionId = 0;
};
```

3.7 Leave a chat session

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

```
that.leaveChannel = function ()
{
    bus.leaveSession(that.sessionId);
};
```

3.8 Send chat message using 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 session.

```
this.SendChatSignal = function(id, msg, flags)
{
    var msgarg = new AllJoyn.MsgArg("s", new Array(msg));
    this._bsObject.signal("", id, this.chatSignalMember,
        new Array(msgarg), 0, flags);
};
```

3.9 Handle Suspending/Resuming event of the application

On Windows 8/Windows RT, a Windows Store application is suspended when it becomes invisible after about 10 seconds and the application is resumed when users bring it back to the foreground, if not terminated. Upon the Suspending/Resuming events, the application

should pass the events to AllJoyn so that it can release exclusively-held resources (e.g., socket file descriptor and network port).

```
WinJS.Application.addEventListener("checkpoint", suspendingHandler);
function suspendingHandler(eventArgs) {
    try {
        if (bus.onAppSuspend != undefined) {
            bus.onAppSuspend();
        }
    } catch (err) {
        console.log(err.stack);
    }
}

Windows.UI.WebUI.WebUIApplication.addEventListener("resuming", resumingHandler);
function resumingHandler()
{
    try {
        if (bus.onAppResume != undefined) {
            bus.onAppResume();
        }
    } catch (err) {
        console.log(err.stack);
    }
}
```

3.10 Enable Location permission for the application

Declaring the **Location** capability does not enable it. This needs to be done from the application as well. To enable location permissions for the application, follow these steps:

1. Start the application.
2. Go to **Settings**.
3. Click **permissions**.
4. Change the **Location** capability to ON.