Peer Group Manager Usage Guide

HT80-BA065-1 Rev. A

February 8, 2013

Submit technical questions at:

http://www.alljoyn.org/forums

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

Contents

Figures

Tables

1 Overview

The Peer Group Manager is a library that provides application developers with an easy entry point into peer-to-peer programming using AllJoyn. A peer group is simply an established connection between peers that allows them to communicate over AllJoyn. With this library, developers can use AllJoyn with minimum knowledge of AllJoyn concepts, which is great for rapid prototyping. The Peer Group Manager provides a mapping between high-level AllJoyn concepts and underlying implementation details abstracting out some of the more subtle aspects that are not always relevant to new developers of AllJoyn. It also encapsulates a lot of boilerplate AllJoyn code for typical applications including setting default parameters and enforcing a naming convention for advertisements, making the code simpler and cleaner. This component only covers common use cases for typical AllJoyn applications, so if your project is advanced in nature, then this component might not be right for you.

1.1 Audience

This guide will be helpful to Android/Java developers who want an easy way to get started using the AllJoyn peer-to-peer software development framework.

1.2 Glossary

Term	Definition
Advertised Name	■ Group prefix + session port + group name. ■ The session port is automatically assigned when creating the session and is preceded by "sp" in the advertisement. ■ Example: *org.alljoyn.chat.sp42.chatroom1*
Group Prefix	■ The prefix of the advertised name that will be prepended to all group names ■ NOTE: Your PeerGroupManager will only be able to communicate with other PeerGroupManagers that use the same group prefix. ■ Example: *org.alljoyn.chat*
Group Name	■ The suffix of the advertised name that comes after the group prefix ■ Example: *chatroom1*
Peer Group	■ An established connection between peers that allows them to communicate over AllJoyn ■ Multiple peer groups can exist at the same time ■ Applications may be a part of more then one group at any given time
Peer ID	■ The unique name of the BusAttachment

Term	Definition
	■ You can obtain your own Peer Id from the getMyPeerId() method ■ You can obtain the Peer Id of a peer from the following places: ❒ getPeers() when participating in a group with the peer ❒ peerAdded() from the PeerGroupListener ❒ peerRemoved() from the PeerGroupListener ❒ getSenderPeerId() ❒ getGroupHostPeerId()

1.3 Revision History

Table 1 provides the revision history for this document.

Table 1: Revision history

Version	Date	Description
A	February 2013	Initial release

1.4 Related Documents

Table 2 is a list of supplemental documents that relate to the information in this guide.

Table 2: Related documents

Location	Title
http://www.alljoyn.org	*Introduction to AllJoyn*
http://www.alljoyn.org	*AllJoyn Android Environment Setup Guide*

2 Setup and Installation

2.1 Setup

Download and install the following:

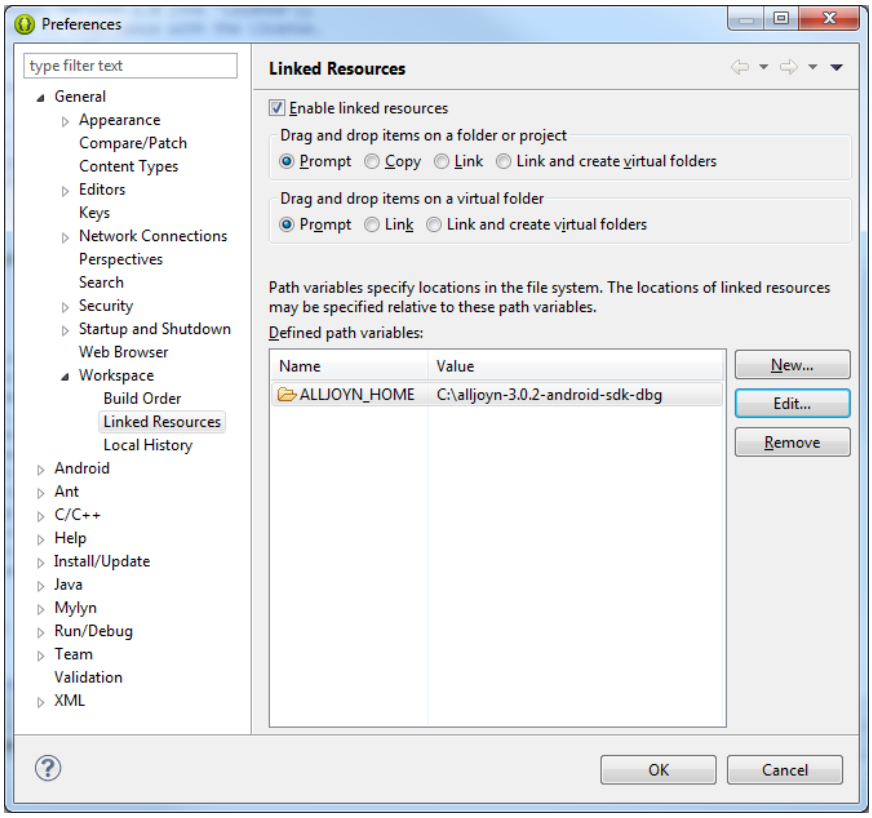
- Eclipse
- ADT Plugin for Eclipse
- Android SDK
  - API Level 8+ (Android 2.2+)
- AllJoyn Android SDK
  - Release 2.3.3 or higher
- Java JDK
  - Version 1.6
  - This is used to generate the Javadocs

Instructions can be found on *https://www.alljoyn.org/docs-and-downloads*

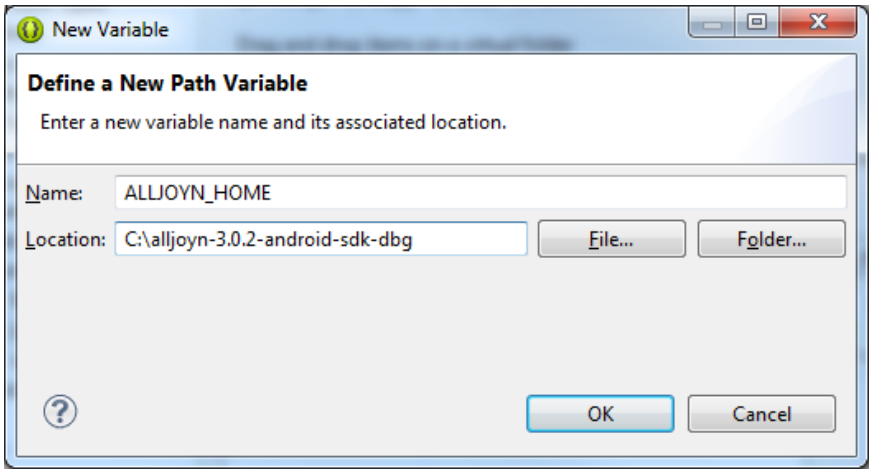
2.2 Modifying the PeerGroupManager Source

2.2.1 Importing the PeerGroupManager into Eclipse

1. On the top menu, select **File > Import**.
2. Select **General > Existing Projects into Workspace** and click **Next**.
3. Click **Browse** next to the **Select root directory** field and navigate to the **PeerGroupManager** Eclipse project within the Git repository that you cloned, then click **OK** and **Finish**.
4. On the top menu, select **Window > Preferences**.
5. On the left hand side, select **General > Workspace > Linked Resources** as shown below:



6. Notice that in the above image there is a path variable called ALLJOYN_HOME that points to the root folder of the AllJoyn Android SDK. You need to create this same variable in your environment, and set it to the path where you have installed the AllJoyn Android SDK.
7. Make sure the **Enable linked resources** option is checked, and then click **New…** to add a new Path Variable.
8. A window should appear to allow you to create a new Path Variable. The name should be “ALLJOYN_HOME” and the path should point to the location of the root folder for the AllJoyn Android SDK. An example is shown below:



9. Click **OK** to create the new Path Variable. You should see it added to the list of defined path variables.
10. Click **OK** to close the Preferences window.

2.2.2 Regenerating the PeerGroupManager JAR

The project is marked as an Android library and the jar is located in the `bin` directory of the project. If you have the automatic build feature enabled in Eclipse under **Projects > Build Automatically** then the project is rebuilt and the jar is updated every time changes are made to the source code and you hit save.

2.2.3 Regenerating the Javadocs

1. Select the PeerGroupManager project in the Package Explorer
2. From the menu up top, select **Project > Generate Javadoc…**.
3. Click **Configure…** next to the **Javadoc command** field and navigate to the `javadoc.exe` within your Java JDK installation.

   An example path is `C:\Program Files (x86)\Java\jdk1.6.0_23\bin\javadoc.exe`

4. Expand the dropdown menu for the PeerGroupManager in the **Select types for which Javadoc will be generated** field and uncheck the `gen` directory leaving only the `src` directory checked.
5. Click **Finish** and the Javadocs will be generated in the `doc` directory of the PeerGroupManager project.

Note Sometimes if APIs are removed from the code, they will still appear in the Javadocs. In this case, simply delete all of the old Javadocs in the `docs` directory and regenerate everything.

2.3 Adding the PeerGroupManager to your Application

1. Copy the `peergroupmanager.jar` from `PeerGroupManager/bin` into the `libs` directory of your Android project.
2. Right-click your project in the package explorer of Eclipse and select **Build Path > Configure Build Path…**.
3. Select the **Libraries** tab and click **Add JARs…**.
4. Navigate to the `peergroupmanager.jar` in the `libs` directory of your project and click **OK** followed by **OK** again.

3 APIs

For the APIs, please refer to the Javadocs located in the `docs` subdirectory of the **PeerGroupManager** project.

4 Usage

This is an example of how a typical Android application might use the Peer Group Manager. The sample code used in the following example is based off of the Simple and Chat Java samples located in the AllJoyn SDK.

4.1 Define the Interface

Define the AllJoyn interface. In this example the interface is called `SimpleInterface` and is composed of a method called `Ping` and a signal called `Chat`.

```
@BusInterface(name = "org.alljoyn.bus.samples.simple.SimpleInterface")
public interface SimpleInterface {
    @BusMethod
    String Ping(String inStr) throws BusException;

    @BusSignal
    public void Chat(String str) throws BusException;
}
```

4.2 Declare your Peer Group Manager member

Declare your Peer Group Manager member so that it can be used throughout your code.

```
private PeerGroupManager pgm = null;
```

4.3 Implement the Bus Object

Create the `BusObject` class that implements your interface, `SimpleInterface` in this example, and the `BusObject` interface. In this example, the `Ping` method simply sends the message received to the UI and then echoes it back to the caller. Notice that the implementation for the `Chat` signal is empty. That is because when signals are received, they trigger a signal handler to execute so our implementation for `Chat` will reside in its signal handler.

Note The `BusObject` class is defined in AllJoyn. Please refer to the AllJoyn documentation for more details.

```
class SimpleService implements SimpleInterface, BusObject {

    public String Ping(String inStr) {
        /* Display the message received */
        sendUiMessage(MESSAGE_PING, inStr);

        /*
```

```
             * Display the reply message which is the same string
             * as the message received.
             */
            sendUiMessage(MESSAGE_PING_REPLY, inStr);
            /* Echo back to the caller */
            return inStr;
        }

        public void Chat(String str) throws BusException {
            /*
             * Intentionally Blank. Implementation is
             * contained within the signal handler.
             */
        }
    }
```

4.4 Implement the Signal Handlers

Implement the signal handlers for any signals that are defined in your interface. The code below checks the sender of the signal to filter out your own chat signals. Then it simply sends the chat string to the UI with the sender prepended to the message. Note that the signal handler is annotated with the `BusSignalHandler` annotation with the interface name and signal name that it is going to handle.

```
@BusSignalHandler(iface = "org.alljoyn.bus.samples.simple.SimpleInterface",
                  signal = "Chat")
public void Chat(String string) {
    String myPeerId = pgm.getMyPeerId();
    String senderPeerId = pgm.getSenderPeerId();

    if (senderPeerId.equals(myPeerId)) {
        Log.i(TAG, "Chat(): ignoring our own signal”);
        return;
    }

    Log.i(TAG, "Chat(): signal " + string + " received from sender " +
          senderPeerId);
    String message = senderPeerId + ": " + string;
    sendUiMessage(MESSAGE_CHAT, message);
}
```

4.5 Connect to the AllJoyn Bus

Connecting your application to the AllJoyn bus is easy. Simply instantiate your Peer Group Manager and the connection happens automatically. It is recommended that you surround

your Peer Group Manager instantiation in a try-catch block in case there was an `IllegalArgumentException` thrown in the constructor.

```
private SimpleService mSimpleService = new SimpleService();
ArrayList<BusObjectData> busObjects = new ArrayList<BusObjectData>();
busObjects.add(new BusObjectData(mSimpleService, OBJECT_PATH));

try {
    pgm = new PeerGroupManager(GROUP_PREFIX, null, busObjects);
}
catch(IllegalArgumentException e) {
    e.printStackTrace();
}
```

4.6 Add a PeerGroupListener (Optional)

If your application needs to know when a new peer joins a group, when a peer leaves a group, when a new group is discovered, when a group stops advertising, or if a group becomes disconnected, you can add a `PeerGroupListener` to listen for these events. The listener can be passed into the constructor of the Peer Group Manager or can be added via the `addPeerGroupListener()` method as shown in the code below. `PeerGroupListeners` can be added at any time and you can add as many listeners as needed.

In the code below, we create a new `PeerGroupListener` to listen for the `peerAdded()` and `peerRemoved()` events. In the example below, we show how to control the number of peers per group. We only want to allow a max of four peers per group so we lock the group once a fourth peer joins that group and unlock the group once the number of peers drops down to three.

```
PeerGroupListener pgListener = new PeerGroupListener() {
    private final int MAX_NUM_PEERS = 4;

    @Override
    public void peerAdded(String peerId, String groupName, int numPeers) {
        if(numPeers >= MAX_NUM_PEERS) {
            pgm.lockGroup(groupName);
        }
    }

    @Override
    public void peerRemoved(String peerId, String groupName, int numPeers) {
        if(numPeers == (MAX_NUM_PEERS - 1)) {
            pgm.unlockGroup(groupName);
        }
    }
```

```
};
pgm.addPeerGroupListener(pgListener);
```

4.7 Registering Signal Handlers

If your AllJoyn interface contains any signals, then you will want to register your signal handlers to be triggered when those signals are received. To do so, simply call `registerSignalHandlers()` passing in the class that contains the annotated signal handler methods.

```
Status status = pgm.registerSignalHandlers(this);
if(status != Status.OK) {
    Log.e(TAG, "Failed to register signal handlers - ” + status);
}
```

4.8 Creating a Group

Creating a group is simple. You just call `createGroup()` and pass in the name of the new group you wish to create. You can create as many groups as you want given that all the group names are unique. If you try to create a group with a name that already exists, the call will fail.

Note You are inherently a peer of your own groups so you do not need to call `joinGroup()` on groups you create.

```
Status status = pgm.createGroup(groupName);
if(status != Status.OK) {
    Log.e(TAG, "Failed to create group - ” + status);
}
```

4.9 Destroying a Group

Destroying a group is just as easy as creating a group. Simply call `destroyGroup()` on the group you wish to destroy. Note that you can only destroy groups that you have created. One tip when allowing the user of the app to decide which group to destroy is to call `listHostedGroups()` which returns a list of groups that you have created. Once the user selects a group name from the returned list, call `destroyGroup()` on it.

```
// Display the list of hosted groups to the user
ArrayAdapter<String> groupListAdapter =
        new ArrayAdapter<String>(activity, android.R.layout.test_list_item);
final ListView groupList =
        (ListView)dialog.findViewById(R.id.hostedGroupList);
groupList.setAdapter(groupListAdapter);

List<String> groups = pgm.listHostedGroups();

for (String group : groups) {
```

```
    groupListAdapter.add(group);
}
groupListAdapter.notifyDataSetChanged();

// Once the user selects a group from the list, call destroyGroup() on it
groupList.setOnItemClickListener(new ListView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position,
            long id) {
        // Call destroyGroup() on the selected hosted group
        String groupName = groupList.getItemAtPosition(position).toString();
        Status status = pgm.destroyGroup(groupName);
        if(status != Status.OK) {
            Log.e(TAG, "Failed to destroy group - ” + status);
        }
    }
});
```

4.10 Joining a Group

To join a group, simply call `joinGroup()` with the desired `groupName`. One trick is to call `listFoundGroups()`, let the user select a group, and then call `joinGroup()` on the selected `groupName`.

```
// Display the list of found groups to the user
ArrayAdapter<String> groupListAdapter =
        new ArrayAdapter<String>(activity, android.R.layout.test_list_item);
final ListView groupList =
        (ListView)dialog.findViewById(R.id.hostedGroupList);
groupList.setAdapter(groupListAdapter);

List<String> groups = pgm.listFoundGroups();

// Remove all groups that you have already joined
groups.removeAll(busHandler.listJoinedGroups());

for (String group : groups) {
    groupListAdapter.add(group);
}
groupListAdapter.notifyDataSetChanged();

groupList.setOnItemClickListener(new ListView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position,
            long id) {
        // Call joinGroup() on the selected advertised group
        String groupName = groupList.getItemAtPosition(position).toString();
        Status status = pgm.joinGroup(groupName);
        if(status != Status.OK) {
            Log.e(TAG, "Failed to join group - ” + status);
        }
```

```
    }
});
```

4.11 Leaving a Group

To leave a group, simply call `leaveGroup()` with the desired `groupName`. One trick is to call `listJoinedGroups()` , let the user select a group, and then call `leaveGroup()` on the selected `groupName`.

Note You cannot leave groups that you are hosting.

```
// Display the list of joined groups to the user
ArrayAdapter<String> groupListAdapter =
     new ArrayAdapter<String>(activity, android.R.layout.test_list_item);
final ListView groupList =
     (ListView)dialog.findViewById(R.id.hostedGroupList);
groupList.setAdapter(groupListAdapter);

List<String> groups = pgm.listJoinedGroups();

for (String group : groups) {
    groupListAdapter.add(group);
}
groupListAdapter.notifyDataSetChanged();

// Once the user selects a group from the list, call leaveGroup() on it
groupList.setOnItemClickListener(new ListView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position,
            long id) {
        // Call leaveGroup() on the selected joined group
        String groupName = groupList.getItemAtPosition(position).toString();
        Status status = pgm.leaveGroup(groupName);
        if(status != Status.OK) {
            Log.e(TAG, "Failed to leave group - " + status);
        }
    }
});
```

4.12 Making Remote Method Calls

Once you are participating in a group, whether you are hosting it or just joined it, you can make remote method calls to another peer's bus objects over that group. To do so you need to get an interface object from the Peer Group Manager. Once you have the interface object, you can use it to make method calls just like any other object. In the example below, we are grabbing an interface object from a bus object registered by the group host. We then call the `Ping()` method and display the reply to the UI. Notice that the method call is

surrounded by a try-catch statement. That is to catch any bus exceptions that may occur over AllJoyn.

```
SimpleInterface mSimpleSignalInterface = pgm.getRemoteObjectInterface(
        pgm.getGroupHostPeerId(groupName),
        groupName,
        OBJECT_PATH,
        SimpleInterface.class);
try {
    String reply = mSimpleInterface.Ping(message);
    sendUiMessage(MESSAGE_PING_REPLY, reply);
}
catch (BusException e) {
    e.printStackTrace();
    Log.e(TAG, "SimpleInterface.Ping(): ” + e.toString());
}
```

4.13 Sending Signals

To send a signal, you also need an interface object just like when you make a remote method call. However, instead of invoking a method on a remote peer in the group, you are sending a signal from your own bus object to the group or to a specific peer within the group. In the example below we are sending a directed signal to the group host. If we were to call the `getSignalInterface()` without the `peerId` parameter, then the signal will be sent to all peers within the group. To emit the signal, we take the interface object and call `Chat()` just as if it was a method with no return value.

```
SimpleInterface mSimpleSignalInterface = pgm.getSignalInterface(
        pgm.getGroupHostPeerId(groupName),
        groupName,
        mSimpleService,
        SimpleInterface.class);
try {
    mSimpleSignalInterface.Chat(message);
}
catch (BusException e) {
    e.printStackTrace();
    Log.e(TAG, "SimpleInterface.Chat(): ” + e.toString());
}
```

4.14 Cleaning Up

When your application is done and is about to exit, you want to clean up all of your AllJoyn resources to prevent any resouce leaks. To do this, simply call `cleanup()` and you’re set. The `cleanup()` method will leave all joined sessions, destroy all hosted sessions, unregister

your bus objects and signal handlers, and finally disconnect your bus attachment from the AllJoyn bus.

```
@Override
protected void onDestroy() {
    super.onDestroy();

    /* Disconnect to prevent resource leaks. */
    pgm.cleanup();
}
```

4.15 Registering Modules

If your application would like to use modules that implement their own AllJoyn communication over the groups established by the PeerGroupManager then they must implement the PGModule interface and be registered with the PeerGroupManager. Here is an example of how a module might be registered in a typical application:

Define the module class to implement the PGModule interface:

```
public class MyModule implements PGModule {
    private BusAttachment bus;
    private int sessionId
    …
    @Override
    Status register(BusAttachment bus, int sessionId) {
        this.bus = bus;
        this.sessionId = sessionId;
        …
    }
}
```

In the activity file, create your PeerGroupManager, set up a group, and then register your module:

```
public class MainActivity extends Activity {
    …
    PeerGroupManager pgm = new PeerGroupManager("org.aj.myApp",
                                                pgListener, busObjects);
    MyModule module = new MyModule();
    pgm.createGroup("main");
    pgm.registerModule(module, "main");
    …
}
```

5 Diagrams

General Flow

Figure 1 shows the general flow of execution while using the PeerGroupManager in a typical application.

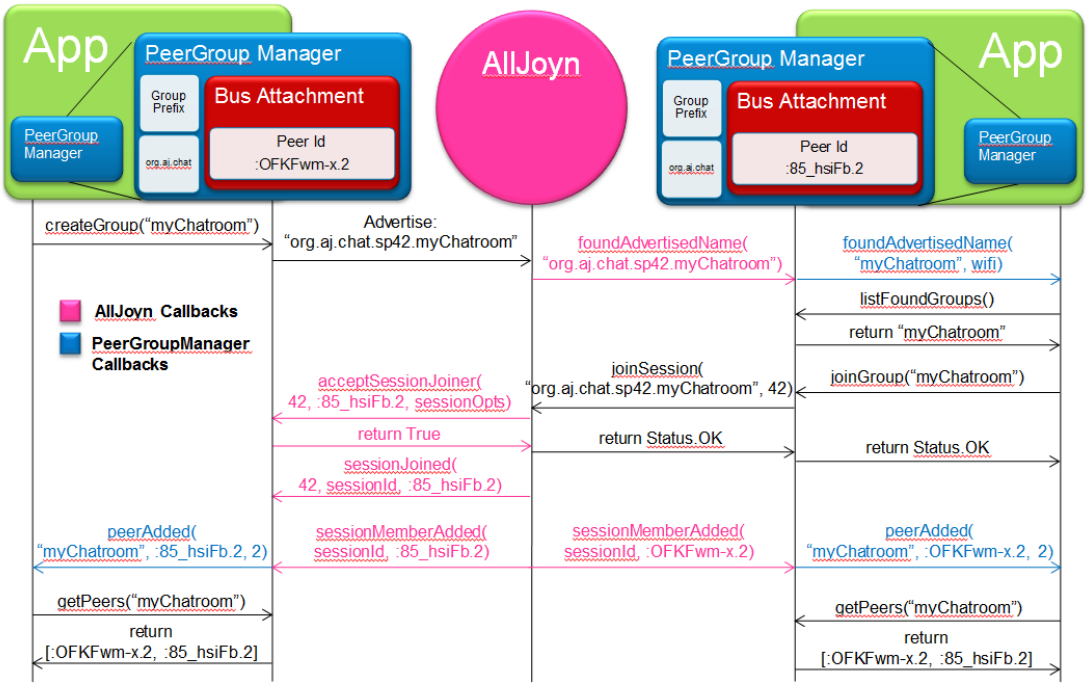


Figure 1: Peer Group Manager general flow

Advertisements

Figure 2 shows how advertisements for groups are constructed.

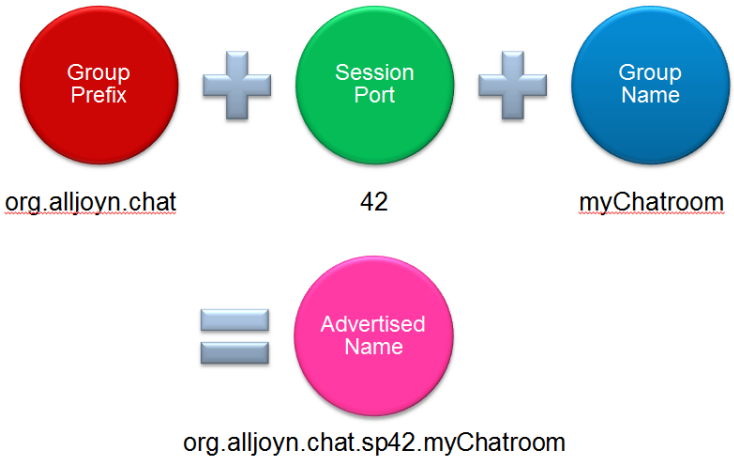


Figure 2: Construction of an Advertisement