ALLSEEN ALLIANCE

Gateway Controller Framework API Guide

(Java)

January 4, 2015

Contents

1 Introduction

1.1 Purpose

This document describes the AllJoyn™ Gateway Controller framework and API and shows how to create an application to control and configure The Gateway Agent. This API is for developing a Control Application which is able to configure Service Profiles and manage the status of Connector Plug-ins to the Gateway Agent embedded Linux application.

1.2 Scope

This document is intended for software engineers and assumes familiarity with the AllJoyn SDK.

1.3 References

The following are reference documents.

- *AllJoyn™ Framework Tutorial*
- *Introduction to AllJoyn™ Framework*
- *GateWay Agent HLD*
- *Gateway Service Framework Interface Definition*
- *Getting Started with the AllJoyn™ Service Framework for Java*
- *About Feature API Guide (Java)*

1.4 Acronyms and terms

Term	Definition
AllJoyn framework	Open source peer-to-peer framework that allows for abstraction of low-level network concepts and APIs
AllJoyn Device	A device existing on the network and using Alljoyn framework to communicate with another AllJoyn devices
ACL	Access Control List. Lists the set of devices/apps/objects/interfaces for which the user would like to enable the remote access.
GM App	Gateway Management Application. Enables end user to define and manage ACL(s) via the Controller app.
Controller App	An application which exists on the same network as the GM App and communicates with it using the Controller Framework to control Connector Apps and configure its ACL(s)
Connector App	Connector Application. An application that is controlled by the Controller App. via the GM App. and provides capabilities of remote access to the AllJoyn devices (Remoted Apps) and Exposed Services
Exposed services	Part of an ACL. Lists the AllJoyn services provided by the Connector App, that are being exposed for the AllJoyn Devices.
Remoted Apps	Part of an ACL. List of AllJoyn applications with remote access permission.
Manifest	Manifest file provides metadata about the Connector Apps as well as the Exposed Services that the Connector App exposes and the object path/interfaces the Connector App is ready to remote.

2 Overview

2.1 Reference code

The reference code consists of a Gateway Controller sample application provided as part of the Gateway Controller Framework.

2.1.1 Source code

https://git.allseenalliance.org/cgit/gateway/gwagent.git/tree/java/GatewayController lists the repositories used to build the Gateway Controller Framework and the Gateway Controller sample application.

Table 1. Gateway Controller Framework components

Package	Description
Alljoyn	The Standard Client Alljoyn code.
AboutService	About feature code.
GatewayController	Gateway Controller Framework

2.2 Obtain the Gateway Controller Framework

See the *Getting Started with the AllJoyn™ Gateway Controller Framework (Android)* section for instructions on compiling the *Gateway Controller Framework*.

2.3 Build a Gateway Controller Application

The following steps provide the high-level process to build a Gateway Controller application.

1. Create the base for the AllJoyn application.
2. Implement the security authentication mechanism.
3. Retrieve list of discovered GM Apps
4. Join session with a GM App
5. From the GM App retrieve the list of its Connector Apps
6. Perform the desired action on a Connector App such as: restart, create ACL, update ACL, retrieve list of existing ACL(s) etc…

2.4 Setting up the AllJoyn framework and About feature

The steps required for this service framework are universal to all applications that use the AllJoyn framework and for any application using one or more AllJoyn service

frameworks. Prior to use of the Gateway Controller Framework, the About feature must be implemented.

Complete the procedures in the following sections to guide you in this process:

- *Getting Started with the AllJoyn™ Service Framework (Java)*
- *AllJoyn™ About Feature API Guide (Java)*

3 Steps to create Gateway Controller App

3.1 Initialize the AllJoyn framework

See the *Getting Started with the AllJoyn™ Service Framework (Java)* section for instructions to set up the AllJoyn framework.

3.1.1 Create bus attachment and add authentication

```
BusAttachment bus = new BusAttachment("GatewayController",
                            BusAttachment.RemoteMessage.Receive);
bus.connect();
bus.registerAuthListener("ALLJOYN_PIN_KEYX ALLJOYN_SRP_KEYX ALLJOYN_ECDHE_PSK",
authListener);
```

3.2 Start Listening to Announcements

3.2.1 Register an Announcement Listener

```
try {

       AboutService about = AboutServiceImpl.getInstance();
       about.startAboutClient(bus);

}

catch(Exception e){…}
```

3.3 Initialize the Gateway Controller Framework

```
GatewayController gwCtrl = GatewayController.getInstance();

gwCtrl.init(bus);
```

3.4 Receive GM App's Announcement signals

The framework listens for the Announcement signals sent from a GM App. In order to be notified about announcements sent from a GM App, implement the `GatewayMgmtAppListener` interface and provide it to the framework. The framework will notify the application every time it receives an announcement form a GM App so that the application may retrieve the list of the discovered GM Apps.

```
gwCtrl.setAnnounceListener( new GatewayMgmtAppListener () {
	@Override
	public void gatewayMgmtAppAnnounced() {
		List<GatewayMgmtApp> gmApps = gwCtrl.getGatewayMgmtApps();
	}
});
```

3.5 Join session with a GM App

The framework provides helper functions to establish session with a GM App. The session may be established synchronously or asynchronously.

```
gwCtrl.joinSession(gwBusName, new GatewayControllerSessionListener() {
	@Override
	public void sessionJoined(SessionResult result) {
		int sessionId  = result.getSid();
		//Verify that status is OK so the session is joined
		Status status  = result.getStatus();
	}
	@Override
	public void sessionLost(int sessionId, int reason) {…}
});
```

3.6 Leave the session with the GM App

```
Status status = gwCtrl.leaveSession(sessionId);
if (status != Status.OK) {
	//Failed to leave the session
}
```

3.7 Retrieve Connector App to be controlled

After GM App is discovered (Section. 3.4) retrieve the list of Connector Apps installed.

```
//For example take the first discovered GM App to work with
GatewayMgmrApp gwMgmtApp         = gwCtrl.getGatewayMgmtApps().get(0);
List<ConnectorApp> connectorApps = gwMgmtApp.retrieveConnectorApps(sessionId);
```

3.8 Retrieve Connector App status

The status of a Connector App consists of three different status: Connection, Operation and Restart. Each status has its own values, for instance Operation status has the values of: Running and Stopped.

In order to retrieve a Connector App status, use the following code snippet:

```
//For example take the first Connector App to work with

ConnectorApp connApp = gwMgmtApp.retrieveConnectorApps(sessionId).get(0);

try {

       ConnectorAppStatus connAppStatus = connApp.retrieveStatus(sessionId);

}

catch (GatewayControllerException gce) {…}
```

3.9 Receive Connector App status changed events

In order to be notified about changes of the Connector App status, implement the `ConnectorAppStatusSignalHandler` interface and provide it to the framework.

Note: `OnStatusChanged` signal is sent only if there is a session established between the Controller App and the GM App.

```
try {

       connApp.setStatusSignalHandler( new ConnectorAppStatusSignalHandler() {

              @Override

              public void onStatusChanged(connectorAppId, status) {

                     …

              }

       });

}

catch (GatewayControllerException gce) {…}
```

3.10 Restart Connector App

When a Connector App is restarted its Operation Status changes from Running to Stopped and back. This status change is followed by the `OnStatusChanged` signal as explained in the Section. 3.9

```
try {
```

```
        connApp.restart(sessionId);
}
catch (GatewayControllerException gce) {…}
```

3.11 Retrieve applicable Connector App capabilities

Connector App capabilities are the rules which may be used for ACL (Access Control List) creation. These rules include “Exposed Services” and “Remoted Apps”. Exposed Services are taken from the Connector App’s Manifest. Remote Apps are created by intersection of the object paths/interfaces from the Connector App’s Manifest with the announced object paths/interfaces of the AllJoyn Devices. If the result of the intersection is empty, such AllJoyn Device will not be included in the resulted Remoted Apps.

```
try {
        AclRules applCap;
        applCap = connApp.retrieveApplicableConnectorCapabilities(sessionId);
}
catch (GatewayControllerException gce) {…}
```

3.12 Create ACL

In order to create a new ACL, provide the framework with the ACL name and the ACL Rules. As explained above in section 3.11, AclRules may be constructed from the rules returned by the call to `retrieveApplicableConnectorCapabilities`.

To create ACL Rules from the first rule of the Exposed Services and the first rule of the first Remoted App:

```
//Retrieve the first rule from the Exposed Services list
RuleObjectDescription expSrvcRule = applCap.getExposedServices().get(0);
expSrvcRule.setConfigured(true);
//Retrieve the first Remoted App
RemotedApp remotedApp = applCap.getRemotedApps().get(0);
//Retrieve the first rule of that remoted app
RuleObjectDescription remAppRule =
                                    remotedApp.getRuleObjectDescription().get(0);
remAppRule.setConfigured(true);


//Create new Exposed Services list
List<RuleObjectDescription> newExposedServices = new ArrayList<>();
newExposedServices.add(expSrvcRule);
```

```
//Create new Remoted Apps list
List<RemotedApp> newRemotedApps = new ArrayList<>();
newRemotedApps.add(new RemotedApp(remotedApp, remAppRule);

AclRules aclRules = new AclRules(newExposedServices, newRemotedAps);
try {
AclWriteResponse response = connApp.createAcl(sessionId, "ACL Name", aclRules);
}
catch (GatewayControllerException gce) {…}
```

Once the ACL is created, its rules are validated against the Connector App's Manifest. All the invalid rules are returned in the AclWriteResponse object, therefore it is important to check the response content also when the status was successful.

3.12.1 ACL Metadata

Controller App may use GM App as a data storage of the Key-Value pairs. Each ACL may be created with the Metadata that can later be read and updated.

```
Map<String, String> aclMetadata = new HashMap<>();
aclMetadata.put("MyKey", "MyValue");
aclRules.setMetadata(aclMetadata);
try {
AclWriteResponse response = connApp.createAcl(sessionId, "ACL Name", aclRules);
}
catch (GatewayControllerException gce) {…}
```

3.13 Retrieve ACL list

```
try {
       List<Acl> acls = connApp.retrieveAcls(sessionId);
}
catch (GatewayControllerException gce) {…}
```

3.14 Delete ACL

The code snippet below shows how to delete the first ACL in the ACLs list returned by the `retrieveAcls` method.

```
try {
       Acl acl = connApp.retrieveAcls(sessionId).get(0);
```

```
        connApp.deleteAcl(sessionId, acl.getId());
}
catch (GatewayControllerException gce) {…}
```

3.15 Activate/Deactivate ACL

Each Connector App may have several ACLs. When all the ACL are deactivated, which means are not used by the Connector App, the Connector App is stopped. To activate the ACL use the following code snippet:

```
try {
        acl.activate(sessionId);
}
catch (GatewayControllerException gce) {…}
```

To deactivate the ACL:

```
try {
        acl.deactivate(sessionId);
}
catch (GatewayControllerException gce) {…}
```

3.16 Retrieve ACL status

Retrieve the ACL status to know whether it is activated or not:

```
try {
        AclStatus status = acl.retrieveStatus(sessionId);
}
catch (GatewayControllerException gce) {…}
```

3.17 Update ACL name

To update the ACL name:

```
try {
        acl.setName("New ACL Name");
}
catch (GatewayControllerException gce) {…}
```

3.18 Retrieve ACL Rules

As explained above, ACL rules are constructed from the “Exposed Services” and the “Remoted Apps”. Retrieved ACL rules are created by the framework from the rules that are part or the current ACL and the rules that haven’t been added to the ACL yet.

To distinguish between the rules that are already part of the ACL from the rules that are not yet, check the `RuleObjectDescription.isConfigured()` method.

For example an ACL that has a “Remoted App” with a rule that has an object path “/MyObj” and an interface implemented by this object “org.allseen.IfaceA”, will have a `RuleObjectDescription.isConfigured()` with value of “true”.

Let’s assume that there is AllJoyn Device that is remoted by this rule and advertises object “/MyObj” that implements three interfaces “org.allseen.IfaceA”, “org.allseen.IfaceB”, “org.allseen.IfaceC”. So the framework will add this Remoted App twice. First with the already configured rule: “/MyObj” and the interface “org.allseen.IfaceA” and second with “/MyObj” and the interfaces “org.allseen.IfaceB”, “org.allseen.IfaceC”. For the second object the `RuleObjectDescription.isConfigured()` method will have a “false” value.

To retrieve the ACL rules, `ConnectorCapabilities` should be provided in order to validate the ACL rules against the Connector App’s Manifest. This is done to eliminate the chance that the Connector App’s Manifest was changed since the ACL was created.

NOTE: We recommend not to cache the `ConnectorCapabilities` on the application layer and to retrieve it every time it’s needed:

```
try {

       ConnectorCapabilities connCap;

       connCap = connApp.retrieveConnectorCapabilities (sessionId);

}

catch (GatewayControllerException gce) {…}
```

Retrieving the ACL rule:

```
try {

       AclRules aclRules = acl.retrieve(sessionId, connCap);

}

catch (GatewayControllerException gce) {…}
```

3.19 Update ACL

Updating ACL is very similar to its creation as it explained in the section 3.11.

Create AclRules of the updated ACL and retrieve the `ConnectorCapabilities`. AclRules may be created from the rules returned by the `acl.retrieve()` method. The rules that are updated should call `setConfigured(true)` method.

Similar to the ACL creation, the invalid rules as well as the ACL update status, are stored in the returned `AclWriteResponse` object.

```
try {
       ConnectorCapabilities connCap;
       connCap = connApp.retrieveConnectorCapabilities (sessionId);
       AclWriteResponse response = acl.update(sessionId, aclRules,connCap);
}
catch (GatewayControllerException gce) {…}
```

3.19.1 Update ACL Metadata

In the section 3.12.1 we explained how to create the metadata for the ACL to be stored in the GM App. The previously stored metadata can be retrieved by the call:

```
try {
       AclRules aclRules = acl.retrieve(sessionId, connCap);
       Map<String, String> aclMetadata = aclRules.getMetadata();
}
catch (GatewayControllerException gce) {…}
```

To update the metadata call the following:

```
try {
       acl.setMetadata(aclMetadata);
}
catch (GatewayControllerException gce) {…}
```