AllJoyn™ Media Control Service High-Level Design

Revision 0.3

October 27, 2014

Contents

Figures

1 Purpose and Scope

1.1 Introduction

The AllJoyn™ Media Delivery framework provides a standardized infrastructure for media applications. It consists of the following components:

- AllJoyn Media Content service—Makes media content available to be browsed and searched.
- AllJoyn Media Control service—Controls the playing of media content.
- Streaming—Enables the media content to be played by the media control.

Together, they allow for rich media experiences across multiple devices.

This document describes the high-level design for the Media Control service framework.

1.2 References

The following are reference documents:

- *AllJoyn™ Media Content Service High-Level Design Document*, Rev 0.2, October 27, 2014
- *AllJoyn™ About Feature Interface Definition*

1.3 Revision history

Table 1 provides the revision history for this document.

Table 1. Revision history

Version	Date	Description
0.1	August 7, 2014	Initial draft
0.2	September 7, 2014	Legal review
0.3	October 27, 2014	■ Updated the APIs. ■ Removed the description of discovery as this described in the About feature documentation. ■ Updated the overview. ■ Removed deprecated interfaces.

1.4 Acronyms

Acronym	Definition
AJAS	AllJoyn Audio Service framework
AJ-MCS	AllJoyn Media Content Service

Acronym	Definition
AJ-MCTRLS	AllJoyn Media Control Service
NAS drive	Network-Attached Storage drive

1.5 Common terms

Term	Definition
AllJoyn Media Delivery Framework	An AllJoyn framework consisting of separate AllJoyn components that together are used for the discovery, distribution and rendering of media content within a network of AllJoyn-enabled devices.
AllJoyn Media Content service	Service that exposes media-centric files to the AllJoyn network. Includes file type, file metadata, and preferred streaming protocol.
AllJoyn Media Content Agent	A “smart aggregator” that can act as negotiator and mapper between different "media SW stacks.” It is used between Media Delivery Framework components and AllJoyn bridge-supported components for other existing platforms/protocols.
AllJoyn Media Control service	A set of semantic AllJoyn Control interfaces for media controls, targeted at application developers. Controls typically include play, plause, forward, rewind, grouping, channel up/down, volume up/down.
AllJoyn Media application	An application that is created by an application developer for the purpose of discovering media content on AllJoyn devices within an IP-based network and rendering that content to another AllJoyn device on the same network. The application also serves as the controller for media content – play, pause, fast forward, etc.
Content Source	A device that is a content server and stores media content files. A cloud streaming service that provides media content can also be categorized as a Content Source.
Player	A device that is a renderer of media content and can play back media content files (e.g., mobile phones, tablets, televisions, speakers).
PropertyStore	Interface used by an application using the AboutService that maintains the values returned as AboutData.
Streaming protocol	Protocol for streaming media content between AllJoyn-enabled devices.

2 System Design

2.1 Overview

The goal of the Media Control service is to standardize the way players are exposed on the AllJoyn network, queried and controlled. This document describes the Media Control service interfaces and metadata mappings.

This chapter discusses the system design.

- An MDF player must implement the mediaPlayer interface.
- The Player API is MIME type-agnostic. The supported MIME types depends on the player implementation. To enable this, the mediaPlayer interface exposes the GetPlayerInfo method which returns the information about the player. This method must returns the player’s supported MIIME types. The application responsible to match content source to the player capabilities.
- The sample player provided in the open source implementation will accept the play commands and logs the event. It does not play back the file content.

2.2 System architecture

Figure 1 illustrates the Media Delivery Framework components.

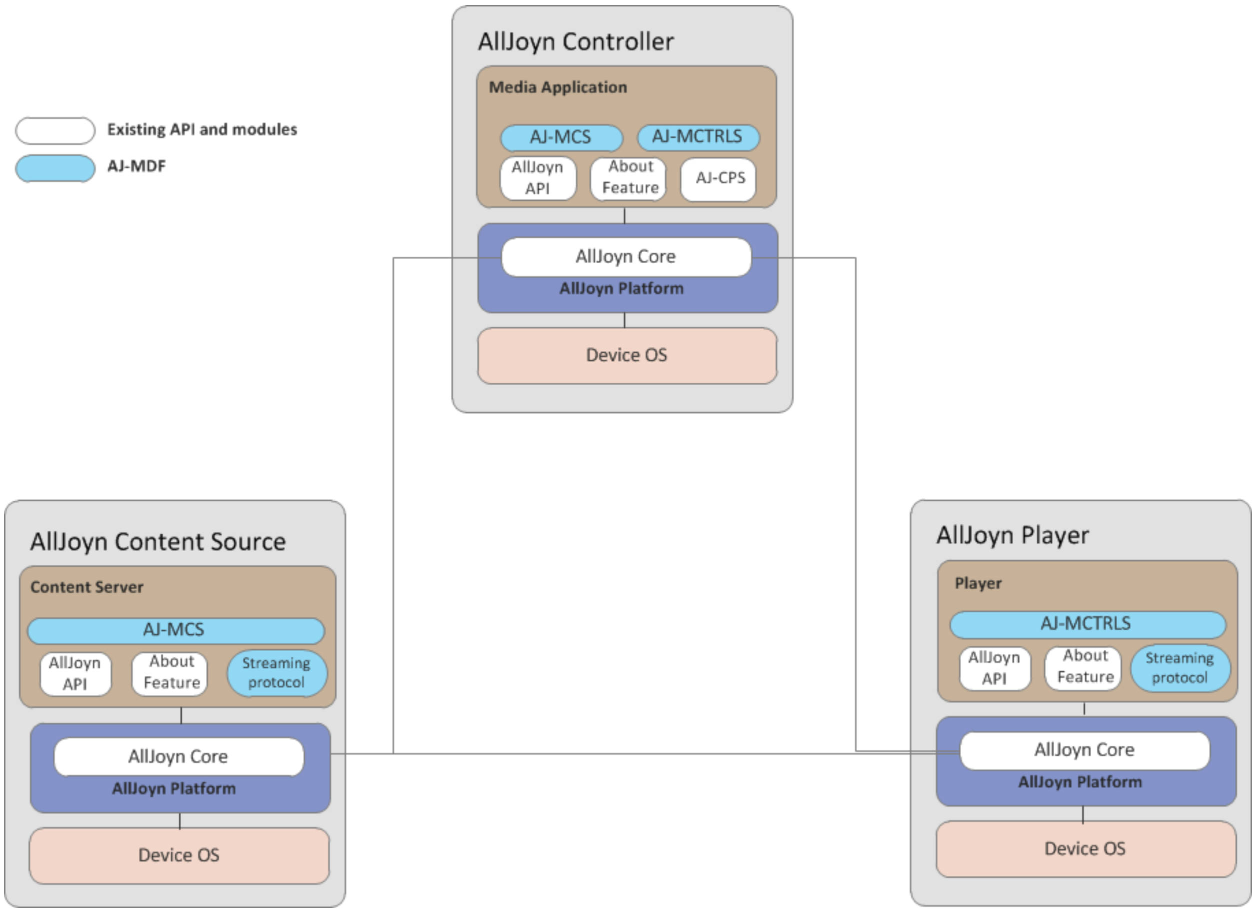

Figure 1. Media Delivery Framework system architecture

2.3 Network layer architecture

Figure 2 illustrates the proposed architecture for the Media Delivery framework. The following subsections define the relevant components.

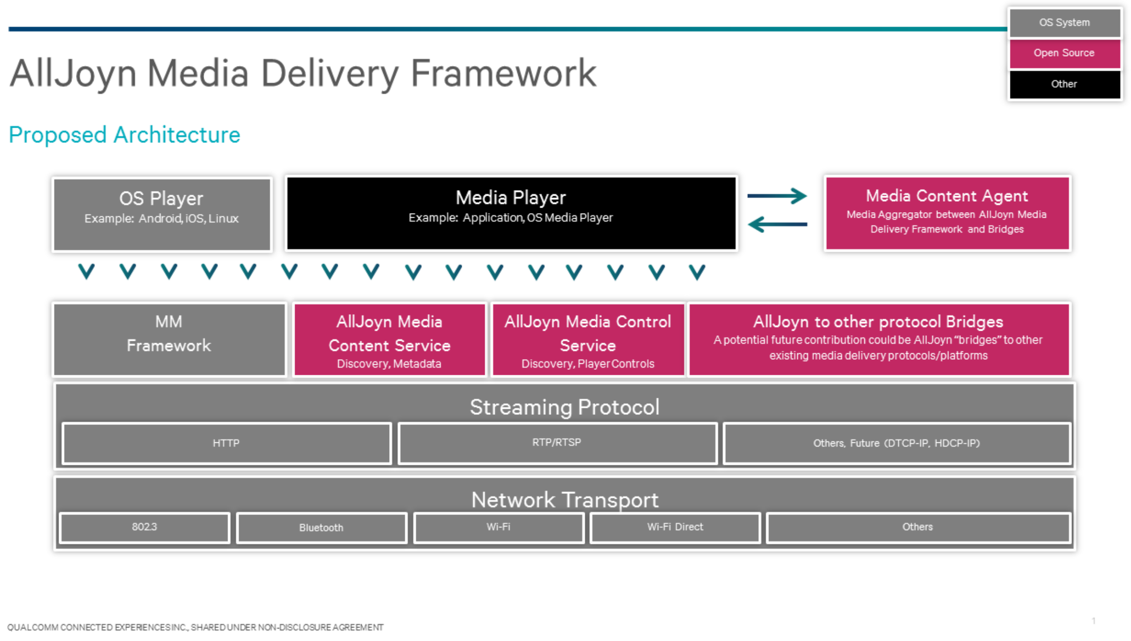
OS System
Open Source
Other
AllJoyn Media Delivery Framework
Proposed Architecture
OS Player
Example: Android, iOS, Linux
Media Player
Example: Application, OS Media Player
Media Content Agent
Media Aggregator between AllJoyn Media Delivery Framework and Bridges
MM Framework
AllJoyn Media Content Service
Discovery, Metadata
AllJoyn Media Control Service
Discovery, Player Controls
AllJoyn to other protocol Bridges
A potential future contribution could be AllJoyn "bridges" to other existing media delivery protocols/platforms
Streaming Protocol
HTTP
RTP/RTSP
Others, Future (DTCP-IP, HDCP-IP)
Network Transport
802.3
Bluetooth
Wi-Fi
Wi-Fi Direct
Others
QUALCOMM CONNECTED EXPERIENCES INC., SHARED UNDER NON-DISCLOSURE AGREEMENT

Figure 2. AllJoyn Media Delivery Framework network layer architecture

2.3.1 Media Content service

The Media Content service consists of a content source (server) and a query component that is used to discover content sources.

The content source part of the Media Content service API is used to create and expose a content source on the AllJoyn network, describing its media content's metadata and methods in a standardized format. The query part of the API is used to discover content sources and subsequently query them. It is an implementation for devices that are content servers and for application developers who wants to discovery content sources on the network and query their metadata.

An AllJoyn Media application uses the Media Content service to discover or search for content sources on the network, and retrieve the metadata for media content on the source to present to the end user for selection. It also uses the content source component of the Media Content service API if it wants to expose its own media content on the network.

2.3.2 Media Control service

The Media Control service of a controller and a player. The controller part is used to discover players on the AllJoyn network and to control them i.e., play, pause etc. The player component is used to create a player, describing its methods and capabilities in a standardized way, and advertising it on the network. It is an implementation for devices that are players and for application developers who want to detect players and their playback capabilities on the network.

An AllJoyn Media application uses the Media Control service to discover or search for players on the network and to control them i.e., play, pause etc. It also uses the player component of the Media Control service API if it wants to expose its own player on the network.

2.3.3 Streaming protocol

The Streaming protocol consists of a source (server) and a player. The source is used by the Media Content service API to listen for incoming connection requests from the players and serve the requested content. The player is used by the Media Control service to assemble the packages received from the source before passing the data on to the player.

NOTE

Streaming is not part of the Media Delivery Framework. It is required by the Control application to obtain data from the Content application that is built on top of the Media Delivery Framework.

2.4 Discovery using About announcement

The control services are discovered using the AllJoyn AboutService. For a full description, refer to the *AllJoyn™ About Feature Interface Definition* document.

3 Control Service API

3.1 Overview

The Control Service API enables an application to control the playback of media content. It consists of the following interfaces:

- mediaPlayer
- zoneManager
- Volume

3.2 mediaPlayer

This is used to control the playing of media content.

3.2.1 Interface name

Interface name	Version	Secured
org.allseen.media.control.mediaPlayer	1	no

3.2.2 Properties

Property name	Type	List of values	Writable	Description
PlayState	(sxuuuiia(ssssxsssa{ss}a{sv}v))	N/A	no	Current play state information. ■ s: playstate (STOPPED, ...) ■ x: position (ms) ■ u: current sample rate ■ u: audio channels ■ u: bits per sample ■ i: index current item ■ i: index next item ■ a(ssssxsssa{ss}a{sv}v): ❒ size=0 (no item), ❒ size=1 (current item) ❒ size=2 (current & next item) ❒ see Item in UpdatePlaylist
LoopMode	s	■ “ONE” ■ “ALL” ■ “NONE”	yes	Loop mode setting
ShuffleMode	s	■ “LINEAR” ■ “SHUFFLE”	yes	Shuffle mode setting

3.2.3 Methods

Method name	Parameter name	Mandatory	Type	List of values	Direction	Description
GetPlayerInfo	See below					Return information about the player.
	displayName	yes	s	N/A	out	Player name.
	capabilities	yes	as	N/A	out	Array of key value pairs. See Capability data for a list of the possible keys and values.
	maximumVolume	yes	i	N/A	out	Maximum setable volume.
	zoneInfo	yes	(siv)	N/A	out	Current zone that the player belongs to. ■ s=zoneId ■ i=zone timestamp ■ v=STRING (lead player known-name) | DICTIONARY {players' known-name, timestamp}
Play	See below					Start playing the item at the index at the specified start position. If Play() is called while the playlist is playing, it will restart playback from the start of the current track.
	itemIndex	yes	i	N/A	in	Index in the playlist of the item to play.
	startPositionMsecs	yes	x	N/A	in	Start position in milliseconds.
	pauseStateOnly	yes	b	■ true ■ false	in	Indicates whether to start streaming (false) or just pause at the specific position (true). This is used for transferring of playlists.
Next	N/A					Start playing the next item in the playlist.
Previous	N/A					If currently at the start of the item, then play the previous item (if it exists). Otherwise, rewind to the start of the item.
ForcedPrevious	N/A					Always move to the previous item regardless of the position in the current item.
Pause	N/A					Pause the playback.
Resume	N/A					Resume the playback.
Stop	N/A					Stop the playback and set the playback postion to the start of the item.
SetPosition	See below					Set the current postion in a track.
	PositionMsecs	yes	x	N/A	in	Position offset in the play item (in miliseconds).
UpdatePl	See below					Update the current play list.

Method name	Parameter name	Mandatory	Type	List of values	Direction	Description
ayList	playlistItems	yes	a(ssssxsssa{ss}a{sv}v)	N/A	in	Items in the play list. ■ s: url ■ s: title ■ s: artist ■ s: thumbnail url ■ x: duration (ms) ■ s: mediaType ■ s: album ■ s: genre ■ a{ss}: other data (country, channel, ...) ■ a{sv}: medium description (codec, container, protocol, ...) ■ v: userData
	index	yes	i	N/A	in	New index of the current item.
	controllerType	yes	s	N/A	in	User-defined string to identify the controller type.
	playlistUserData	yes	s	N/A	in	User-defined information.
GetPlayList	See below					Read the current play list.
	items	yes	a(ssssxsssa{ss}a{sv}v)	N/A	out	See UpdatePlaylist.
	controllerType	yes	s	N/A	out	User-defined string to identify the controller type.
	playlistUserData	yes	s	N/A	out	User-defined data.

3.2.3.1 Capability data

This data is used to enable filtering of players. An OEM / application developer can define their own capability values. These are returned in the GetPlayerInfo method.

Key	Type	List of values	Mandatory	Description
mimeTypes	as	Standard or non-standard MIME types	yes	Array of the MIME types (string) that the player supports, e.g., {“audio/mpeg”,video/mp4”,”image/jpeg”}.

Key	Type	List of values	Mandatory	Description
transports	aq	■ 1 – HLS ■ 2 – MPEG DASH ■ 3- RTSP ■ 4- MMS ■ 5- RTP ■ 6- RTCP ■ 7- UDP ■ 8- TCP ■ 9- RTMP ■ 10- MPEG-TS ■ 11- RDT ■ 12- WebM	yes	Array of transports that the player supports.

3.2.4 Signals

Signal name	Parameters			Sessionless	Description
PlayListChanged	N/A			no	The play list has changed
PlayStateChanged	**Parameter name**	**Mandatory**	**Type**	no	The play state has changed.
	state	yes	(sxuuuiia(ssssxsssa{ss}a{sv}v))		See PlayState property
LoopModeChanged	**Parameter name**	**Mandatory**	**Type**	no	The loop mode has changed.
	loopMode	yes	s		Loop mode setting
ShuffleModeChanged	**Parameter name**	**Mandatory**	**Type**	no	The shuffle mode has changed.
	shuffleMode	yes	s		Shuffle mode setting
OnPlayBackError	**Parameter name**	**Mandatory**	**Type**	no	Error is sent when a playback issue is detected.
	index	yes	i		Index—Item index in the play list.
	error	yes	s		Error message
	description	yes	s		Description of the error

3.2.5 Introspection XML

```
<node>
      <interface name=“ org.allseen.media.control.mediaPlayer”>
      <!-- <annotation name=“org.alljoyn.Bus.Secure” value=“true”/> -->
      <method name=“GetPlayerInfo”>
            <arg name=“displayName” type=“s” direction=“out”/>
            <arg name=“capabilities” type=“as” direction=“out”/>
            <arg name=“maximumVolume” type=“i” direction=“out”/>
            <arg name=“zoneInfo” type=“(siv)” direction=“out”/>
                  <!-- s=zoneId
                  i=zone timestamp
```

```
                    v=STRING (lead player known-name) | DICTIONARY {players' known-
name, timestamp} -->
        </method>

        <!-- Player -->
        <method name=“Play”>
                <arg name=“itemIndex” type=“i” direction=“in”/>
                <arg name=“startPositionMsecs” type=“x” direction=“in”/>
                <arg name=“pauseStateOnly” type=“b” direction=“in”/>
        </method>
        <method name=“Next”/>
        <method name=“Previous”/>
        <method name=“ForcedPrevious”/>
        <method name=“Pause”/>
        <method name=“Resume”/>
        <method name=“Stop”/>
        <method name=“SetPosition”>
                <arg name=“positionMsecs” type=“x” direction=“in”/>
        </method>

        <method name=“UpdatePlaylist”>
                <arg name=“playlistItems” type=“a(ssssxsssa{ss}a{sv}v)” direction=“in”/>
                        <!-- array of item. Item:
                                s: url
                                s: title
                                s: artist
                                s: thumbnail url
                                x: duration (ms)
                                s: mediaType
                                s: album
                                s: genre
                                a{ss}: other data (country, channel, ...)
                                a{sv}: medium description (codec, container, protocol, ...)
                                v: userData
                        -->
                <arg name=“index” type=“i” direction=“in”/>
                <arg name=“controllerType” type=“s” direction=“in”/>
                <arg name=“playlistUserData” type=“s” direction=“in”/>
        </method>
        <signal name=“PlaylistChanged”/>
        <method name=“GetPlaylist”>
                <arg name=“items” type=“a(ssssxsssa{ss}a{sv}v)” direction=“out”/>
                        <!-- see UpdatePlaylist -->
                <arg name=“controllerType” type=“s” direction=“out”/>
                <arg name=“playlistUserData” type=“s” direction=“out”/>
        </method>
        <method name=“GetPlaylistInfo”>
                <arg name=“controllerType” type=“s” direction=“out”/>
                <arg name=“playlistUserData” type=“s” direction=“out”/>
        </method>

        <property name=“PlayState” type=“(sxuuuiia(ssssxsssa{ss}a{sv}v))”
access=“read”/>
                <!--
                        s: playstate (STOPPED, ...)
                        x: position (ms)
                        u: current sample rate
                        u: audio channels
                        u: bits per sample
                        i: index current item
                        i: index next item
                        a(ssssxsssa{ss}a{sv}v): size 0 (no item), 1 (current item) or 2
```

```
(current and next item)
                          see Item in UpdatePlaylist
             -->
      <signal name=“PlayStateChanged”>
             <arg name=“state” type=“(sxuuuiia(ssssxsssa{ss}a{sv}v))”/> <!-- see
PlayState property -->
      </signal>

      <property name=“LoopMode” type=“s” access=“readwrite”/>
      <signal name=“LoopModeChanged”>
             <arg name=“loopMode” type=“s”/>
      </signal>
      <property name=“ShuffleMode” type=“s” access=“readwrite”/>
      <signal name=“ShuffleModeChanged”>
             <arg name=“shuffleMode” type=“s”/>
      </signal>

      <signal name=“OnPlaybackError”>
             <arg name=“index” type=“i”/>
             <arg name=“error” type=“s”/>
             <arg name=“description” type=“s”/>
      </signal>

    </interface>
</node>
```

3.3 zoneManager interface

This interface stores information about the zone the player belongs to. A player can only be part of one zone at any given time.

3.3.1 Interface name

Interface name	Version	Secured
org.allseen.media.control.zoneManager	1	no

3.3.2 Properties

N/A

3.3.3 Methods

Method name	Parameter name	Mandatory	Type	List of values	Direction	Description
CreateZone	See below					Create a zone of players. Each player is added as a player to the main player.
	players	yes	as	N/A	in	Array of player names.
	zoneId	yes	s	N/A	out	Zone ID.
	timestamp	yes	i	N/A	out	Time the zone was created.

Method name	Parameter name	Mandatory	Type	List of values	Direction	Description
	failedPlayers	yes	a{si}	N/A	out	Players that failed to be added to the zone.

3.3.4 Signals

Signal name	Parameters			Sessionless	Description
OnZoneChanged	**Parameter name**	**Mandatory**	**Type**	no	The zone has changed
	zoneId	yes	s		Zone ID
	timestamp	yes	i		Time the zone was created
	players	yes	a{si}		Players that failed to be added to the zone.

3.3.5 Introspection XML

```
<node name="">
     <interface name="net.allplay.zoneManager">
       <method name="CreateZone">
              <arg name="players" type="as" direction="in"/>
              <arg name="zoneId" type="s" direction="out"/>
              <arg name="timestamp" type="i" direction="out"/>
              <arg name="failedPlayers" type="a{si}" direction="out"/>
       </method>
       <signal name="OnZoneChanged">
              <arg name="zoneId" type="s"/>
              <arg name="timestamp" type="i"/>
              <arg name="players" type="a{si}"/>
       </signal>

     </interface>
</node>
```

3.4 Volume interface

The Volume interface is implemented by an AllJoyn application on a multimedia device to allow an AllJoyn client to control its audio volume.

3.4.1 Interface name

Interface name	Version	Secured
org.alljoyn.Control.Volume	1	no

3.4.2 Properties

Property name	Type	List of values	Writable	Description
version	q	positive	no	Interface version number
Volume	n	N/A	yes	The volume.

Property name	Type	List of values	Writable	Description
VolumeRange	nnn	■ Low ■ High ■ Increment	no	The volume range.
Mute	b	■ true ■ false	yes	The volume's mute state.
Enabled	b	■ true ■ false	no	Indicates if the volume control is enabled.

3.4.3 Methods

Method name	Parameter name	Mandatory	Type	List of values	Direction	Description
AdjustVolume	See below					Adjust the volume by the given number. The adjustment can be up (positive value) or down (negative value).
	delta	yes	q	N/A	in	Number of increments to adjust.
AdjustVolumePercent	See below					Adjust the volume to a percentage of the maximum value.
	change	yes	d	-1.0 – 1.0	out	See Change algorithm information.

3.4.3.1 Change algorithm information

The change has floating point values between -1.0 and 1.0 to represent volume changes between -100% to 100%.

A positive value (respectively negative), will increase (respectively decrease) the volume by the percentage of the “remaining range” towards the maximum (respectively minimum) value, i.e. difference between the current volume and the maximum (respectively minimum) volume.

For example, when the volume range is [0-100] and we want to adjust by +50%:

- If the current volume is 25, the increment will be:

  “(100-25)*50%=75*0.5=38” (once rounded) so the new volume will be 63.
- Another adjustment by +50% will be “(100-63)*0.5=19” to a volume of 82.

If we want instead to adjust by -50%, the decrement would be “(25-0)*0.5=13” to a volume of 12, and another adjustment by -50% would be “(12-0)*0.5=6” to a volume of 6.

This behavior provides a better user experience when changing the volume of multiple speakers (group). At the same time, although each speaker has a different starting point, all the players will reach 100% (or 0%) at the same time.

3.4.4 Signals

Signal name	Parameters			Sessionless	Description
VolumeChanged	**Name**	**Mandatory**	**Type**	no	The volume has changed.
	newVolume	yes	n		The new volume level.
MuteChanged	**Name**	**Mandatory**	**Type**	no	The mute value has changed.
	newMute	yes	b		The new mute level.
Enabled	**Name**	**Mandatory**	**Type**	no	The volume control enable value has changed.
	enabled	yes	b		Indicates if volume control is enabled.

3.4.5 Introspection XML

The following XML defines the org.alljoyn.Control.Volume interface.

```
<node name="“">
      <interface name=“org.alljoyn.Control.Volume”>
            <property name=“Version” type=“q” access=“read”/>
            <property name=“Volume” type=“n” access=“readwrite”/>
            <property name=“VolumeRange” type=“(nnn)” access=“read”/>
            <property name=“Mute” type=“b” access=“readwrite”/>
            <signal name=“VolumeChanged”>
                  <arg name=“newVolume” type=“n”/>
            </signal>
            <signal name=“MuteChanged”>
                  <arg name=“newMute” type=“b”/>
            </signal>
            <method name=“AdjustVolume”>
                  <arg name=“delta” type=“n” direction=“in”/>
            </method>
            <method name=“AdjustVolumePercent”>
                  <arg name=“change” type=“d” direction=“in”/>
            </method>
            <property name=“Enabled” type=“b” access=“read”/>
            <signal name=“Enabled”>
                  <arg name=“enabled” type=“b”/>
            </signal>
      </interface>
</node>
```