Dialogic® PowerMedia™ XMS JSR 309 Connector Software Release 3.4

Developer's Guide

October 2017 Rev 5.0

www.dialogic.com

Copyright and Legal Notice

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation and its affiliates or subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in certain safety-affecting situations. Please see http://www.dialogic.com/company/terms-of-use.aspx for more details.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 3300 Boulevard de la Côte-Vertu, Suite 112, Montreal, Quebec, Canada H4R 1P8. **Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.**

Dialogic, Dialogic Pro, Veraz, Brooktrout, Diva, BorderNet, PowerMedia, PowerVille, PowerNova, MSaaS, ControlSwitch, I-Gate, Cantata, TruFax, SwitchKit, Eiconcard, NMS Communications, SIPcontrol, Exnet, EXS, Vision, inCloud9, and NaturalAccess, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation and its affiliates or subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 3300 Boulevard de la Côte-Vertu, Suite 112, Montreal, Quebec, Canada H4R 1P8. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

The names of actual companies and products mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Table of Contents

Revision History

Revision	Release Date	Notes
5.0	October 2017	Updates for JSR 309 Release 3.4 Service Update 1. Release Issues: Updated the section.
4.0	May 2017	Updates for JSR 309 Release 3.3. MediaMixer: Updated the section. MediaSession: Updated the section. Player: Updated the section. Recorder: Updated the section. SdpPortManager: Updated the section.
3.0	December 2016	Updates for JSR 309 Release 3.2.
2.0	July 2016	Updates for JSR 309 5.2 Service Update 1. JSR 309 Connector Requirements: Updated the minimum requirements. JSR 309 Connector APIs: • Removed joinInitiate and unjoinInitiate. • Updated NetworkConnection, PlayerEvent, and SignalDetector.
1.0	March 2016	Initial release of this document.
Last modified: October 2017		

Refer to www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

1. Welcome

This Developer's Guide is written for developers of the Dialogic® PowerMedia™ XMS JSR 309 Connector Software (also referred to herein as "JSR 309 Connector"), which is used in conjunction with the Dialogic® PowerMedia™ Extended Media Server (also referred to herein as "PowerMedia XMS" or "XMS").

This Developer's Guide describes the JSR 309 Connector, provides information on its features, and describes any extensions added to the JSR 309 Connector (based on JSR 309 specification) in addition to which methods/parameters are supported.

The following are some of the benefits offered by the JSR 309 Connector for PowerMedia XMS Media Server Control:

- The underlying communication with the PowerMedia XMS may change, but the application using the JSR 309 interface may not need changes.
- The JSR 309 implementation hides all protocol communication transactions and synchronizations to and from the Media Server; thus, the application development is made simple by allowing the application developer to concentrate on the main business logic and service implementations.
    - For example: One application request can result in multiple underlining protocol requests to the Media Server. The connector hides such complexity by returning to this API request when all internal related transactions are completed.
- Async Communication to the Media Server using the JSR 309 Connector is trivial, applications are only required to register JSR 309 Listeners.
- The JSR 309 Connector is the Dialogic implementation built with serialization in mind. This allows for the connector to work in platforms that support data replication aka. Cluster setup.
- The connector intelligent state machine alleviates the application owns state machine.

Assumptions

This Developer's Guide assumes that you have the following knowledge and experience:

- Familiarity with the Java Specification Request (JSR) 309 documentation version 1.0.
- Familiarity with the JSR 309 Overview of Media Server Control API document provided with JSR 309 specification download at http://www.jcp.org.
- Familiarity with application server platform development and administration of choice.
- Prior experience with JSR 289 (SIP Servlets).
- Prior experience with Java Platform Enterprise Edition (Java EE) development.
- Familiarity with PowerMedia XMS administration and configuration.

Related Information

See the following for more information:

- PowerMedia XMS datasheet at http://www.dialogic.com/products/media-server-software/xms.
- PowerMedia XMS documentation at http://www.dialogic.com/manuals.
- Dialogic technical support at http://www.dialogic.com/support.
- JSR 309 documentation on the Java Community Process website at http://www.jcp.org/aboutJava/communityprocess/final/jsr309.

2. Overview

This section provides an overview of the Java Specification Request (JSR) 309 and describes the JSR 309 Connector features and limitations.

Terminology

A brief description of terminology used in this document is provided for reference.

- **Servlet** – A Java class which conforms to the Java Servlet Interface, by which a Java class may respond to HTTP requests. Applications using servlets may be packaged in a WAR file as a web application.
- **Java Specification Request (JSR)** – A formal document created by members of the Java Community Process (JCP) that adds features and functionality to the Java platform.
- **JEE or J2EE** – Java Platform, Enterprise Edition. A platform for server programming in the Java programming language.
- **Web Application Server** – Application Server based on JEE or J2EE.
- **MSML** – Media Server Markup Language.
- **AS** – Application Server.
- **TCK** – Technology Compatibility Kit.

JSR 309 Media Server Control API

Java Specification Request (JSR) 309 is a standard Java media server control API for multimedia application development. It provides a generic media server abstraction interface that is independent of the underlying media server control protocol. The multimedia applications, such as IVR, voice-mail, audio conferencing, and call center, are typically deployed in a SIP-based infrastructure.

The JSR 309 API provides three areas of functionality:

- Network Connection (NC) to establish media streams.
- Media Group (MG) functions such as to play, record, and control media content.
- Media Mixer (MM) functions to join media functions to a network connection so as to create conferences and call bridges.

For details on the JSR 309 API, refer to the documentation on the Java Community Process website at http://www.jcp.org.

For a list of JSR 309 APIs and parameters supported by the JSR 309 Connector, refer to the Supported Features.

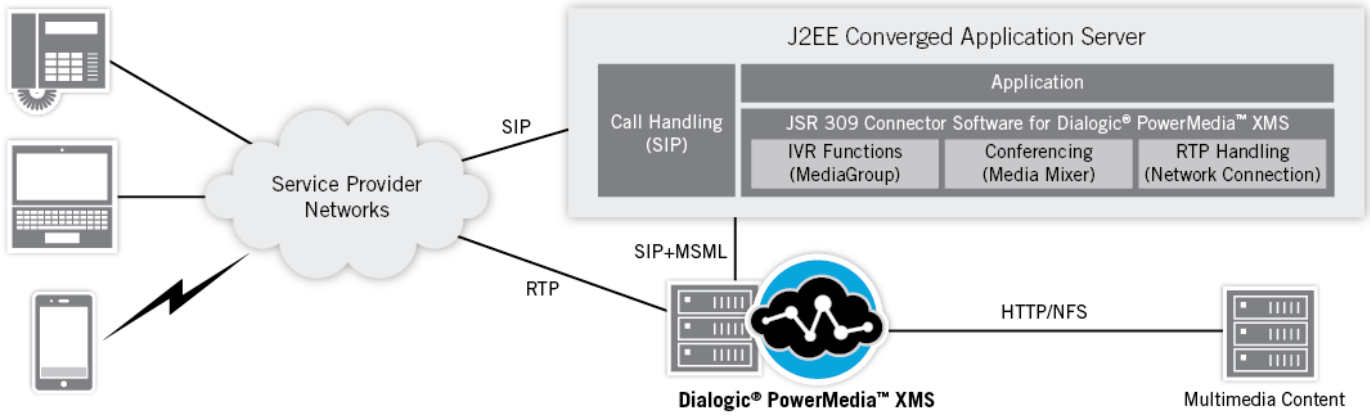
JSR 309 Connector

The JSR 309 Connector is the Dialogic implementation of the JSR 309 version 1.0 final specification. This software runs on various supported application servers and enables a multimedia application on the application server to control the PowerMedia XMS using the JSR 309 API. For a list of supported application servers, refer to the Supported Platforms.

The JSR 309 Connector is designed to support an asynchronous programming model. Applications using the connector should be designed to interface using Listener objects for events on operation completion.

System Overview

The following figure illustrates the role of the JSR 309 Connector in a typical deployment:



The following components are included in the figure:

- **J2EE Converged Application Server** – Handles SIP call control and other aspects of real-time multimedia communications using a Java EE environment with JSR 289 support.
- **Application** – Runs on the J2EE Converged Application Server. Examples of applications: IVR, conferencing, announcements, and call centers.
- **JSR 309 Connector** – Software connector that enables the J2EE Converged Application Server to control PowerMedia XMS through JSR 309-compliant API calls.
- **PowerMedia XMS** – Performs the multimedia operations required to establish and maintain real-time communications while providing a high-quality user experience.
- **External Servers** – Used for storing and streaming multimedia content.
- **SIP Servlet API for Call Handling** – SIP stack used to communicate with SIP compliant user agents. JSR 289 is the standard API servlet used by Converged Communication Application Servers.

3. Supported Features

The JSR 309 Connector is compatible with the JSR 309 Media Server Control API version 1.0.

The JSR 309 Connector supports the following functionality:

- Driver loading with driver property support
- Video conference layout
- Audio recording
- Conference mixing
- Bridge conference
- Basic prompt and collect
- Video conference
- Signal detection
- Setup of media server URI programmatically
- Redundant media servers
- Serialization/cluster support
- Video streaming to network connections
- Media mixing with video layout
- Media play/record
- Media handling for WebRTC
- TCK pre-certification at 80% (see Appendix B – TCK Conformance Details)

4. JSR 309 Connector Requirements

The following requirements are needed to be in place before installing the JSR 309 Connector:

- A functional J2EE application server platform for development and testing.
- A functional PowerMedia XMS Release 3.4 system.
- SIP phones or soft clients.

The *MANIFEST.MF* file has been included in the JSR 309 Connector JAR files with version and build number. The versions of these files need to be exactly the same and should not be mixed and matched with older or newer versions of the JAR files.

In addition, the *MANIFEST.MF* file describes the version of PowerMedia XMS that the JSR 309 Connector was tested on.

Supported Platforms

The JSR 309 Connector was developed using the Java SDK version 1.6.x, 1.7.x, and 1.8.x platform dependent.

The JSR 309 Connector has been deployed and tested on the following application server platforms:

Minimum Requirements

Oracle

1. Oracle Communications Converged Application Server (OCCAS) 5.1.0
2. Oracle Communications Converged Application Server (OCCAS) 7.0.0.1.0

IBM

1. IBM Liberty Application Server

TeleStax

- TelScale JBoss and Apache-Tomcat Application Server:
  - *TelScale-SIP-Servlets-7.0.2.GA-jboss-as-7.2.0.Final*
  - *TelScale-SIP-Servlets-7.0.2.GA-apache-tomcat-7.0.50*
- Mobicents JBoss and Apache-Tomcat Application Server:
  - *mss-3.0.xxx-jboss-as-7.2.0.Final*
  - *mss-3.0.xxx-apache-tomcat-7.0.50*

  Note: *xxx* = 536 or higher.

5. Development Considerations

This section provides application development guidelines for the JSR 309 Connector.

Dialogic® PowerMedia XMS JSR 309 Connector Software Installation and Configuration Guide for each supported platform.

Configuring Network Time Protocol (NTP) for Accurate SIP Timers

As with any distributed architecture it is important that every component is synchronized to a common system clock. It is highly recommended for each distributed component to synchronize to a common NTP server. Differences in system clock settings can cause a number of severe issues such as:

- SIP timers firing prematurely on servers with the fast clock settings.
- Poor distribution of timer processing between two distributed elements. For example, because the initial T1 timer value of 500 milliseconds controls the retransmission interval for INVITE request and responses, and also sets the initial values of other timers, even small differences in system clock settings can cause improper SIP protocol behavior.

Configuring the Application Server

For each supported J2EE Application Server Platform, it is expected that the developer has a good understanding of its configuration, setup, and administration. However, there are step-by-step installation instructions in the Appendix of the *Dialogic® PowerMedia™ XMS JSR 309 Connector Software Installation and Configuration Guide* for each supported platform. These instructions make it easier to quickly install and configure the Application Server platform in order to start using it with the JSR 309 Connector.

Application Server Serialization

Java object serialization is used to support replication of Java objects to another process. Serialization enables an application in a distributed or clustered environment to support application replication. By default, serialization is turned off in the JSR 309 Connector. More details are available in the *Dialogic® PowerMedia™ XMS JSR 309 Connector Software Installation and Configuration Guide* for each supported platform.

Note: Application Server must support serialization clustering.

SIP Servlet Initialization

This section recommends the proper way to write your SIP Servlet Initialization when using the JSR 309 Connector.

Since the JSR 309 Connector supports various Application Server Platforms and with each platform differing a bit in how the SIP Servlet Initialization is executed, the following is recommended for application to write the SIP Servlet Initialization code.

Consult the specific Application Server Installation and Configuration documentation for detail explanation on a typical Application SipServlet Initialization procedure.

Setting the Media Server URI Programmatically

The preferred way is to use Dynamic Connector techniques.

Multiple NCs per Media Session versus Single NC per Media Session Model

The JSR 309 API allows multiple Network Connections per a Media Session. The connector by default, only allows one Network Connection per Media Session.

It is recommended for application developers to use one to one relation between NC and MS; however, the connector has the ability to enable handling of multiple Network Connections per Media Session. In order to enable the Multiple Network Connections to a single Media Session mode, you must enable this mode via the connector property file:

```
multiple.network.connection.enabled=true
```

Application Call Leg to Media Session Facts

This section describes some important object relational models that may be useful for application developers:

1. When an application receives an invite to be a proxy to the Media Server, it is recommended to create dedicated JSR 309 Media Session (i.e., Media Session per Application Call Leg). This means if a second invite is received by the application, the application creates a second Media Session.
2. A Media Session is composed of two Session Application Sessions as explained in the advanced section.
3. A Media Session contains the call leg connector state machine. Thus there is a state machine per Media Session.
4. A Media Session only is connected to one and only one Media Server.
5. The connector internally serializes the Media Session across a cluster. Note, the Media Session is the top object that contains all specific object and including state machine for the specific call leg.
6. Release the Media Session, the connector releases all associated resources related to the Media Session from the Media Server.

6. JSR 309 Connector APIs

API Overview

This section describes the JSR 309 Connector functionality that is either unsupported or supported extensions to a standard JSR 309 specification.

The standard specification can be found at http://www.jcp.org/aboutJava/communityprocess/final/jsr309.

List of Packages

Package	Supported	Description
javax.media.mscontrol	Yes	Provides a Java Application Server with a Media Server Control API.
javax.media.mscontrol.join	Yes	Joins media objects to establish media streams between them.
javax.media.mscontrol.mediagroup	Yes	MediaGroup is mostly a player/recorder.
javax.media.mscontrol.mediagroup.http	No	MediaGroup parameters specialized for HTTP file play and record.
javax.media.mscontrol.mediagroup.signals	Yes	Provides resources for detecting and generating signals (DTMF, speech, etc.) for any type of media.
javax.media.mscontrol.mixer	Yes	MediaMixer has the capacity to combine the media streams coming from multiple sources, into a single media stream.
javax.media.mscontrol.networkconnection	Yes	Manages a network termination point, using RTP for an IP network.
javax.media.mscontrol.resource	Yes	Provides the base definitions for managing media processing resources.

Package	Supported	Description
javax.media.mscontrol.resource.common	No	Manages resources that can be inserted in a different ResourceContainer like MediaGroup, NetworkConnection, or MixerAdapter.
javax.media.mscontrol.resource.video	Yes	Capabilities to handle video resources, layout and rendering.
javax.media.mscontrol.spi	Yes	Provides classes and interfaces defining a driver service provider interface (SPI).
javax.media.mscontrol.vxml	No	Capabilities to control VXML dialogs that are delegated to the Media Server.

List of Methods

Method – provides list of all Methods as defined by the JSR 309 standard specification.

Supported? – provides information about its support in the JSR 309 Connector.

Supported – means that it is fully supported by the JSR 309 Connector.

Partially – means that not all features of the Method are supported by the JSR 309 Connector.

Not Supported – means that there is no current support for this Method by the JSR 309 Connector.

Extension – provides information about a Method to have additional support/extension not provided by the JSR 309 specification.

Method	Supported?	Extension
Action	Not Supported	
AllocationEvent	Supported	Yes
AllocationEventListener	Supported	No
AllocationEventNotifier	Supported	No
CodecConstants	Partially	Yes
CodecPolicy	Supported	No
Configuration	Supported	No

Method	Supported?	Extension
Driver	Supported	No
DriverManager	Supported	No
EventType	Supported	No
FileFormatConstants	Partially	Yes
HttpFilePlayerConstants	Not Supported	
HttpFileRecorderConstants	Not Supported	
Joinable	Supported	No
Joinable.Direction	Supported	No
JoinableContainer	Supported	Yes
JoinableDialog	Not Supported	
JoinableStream	Partially	No
JoinableStream.StreamType	Supported	No
JoinEvent	Supported	Yes
JoinEventListener	Supported	No
JoinEventNotifier	Supported	No
JoinException	Not Supported	
MediaConfig	Partially	No
MediaConfigException	Not Supported	
MediaErr	Supported	No
MediaEvent	Supported	No
MediaEventListener	Supported	No
MediaEventNotifier	Supported	No
MediaException	Supported	No
MediaGroup	Supported	Yes

Method	Supported?	Extension
MediaMixer	Partially	No
MediaObject	Supported	No
MediaServiceException	Not Supported	
MediaSession	Supported	Yes
MixerAdapter	Partially	No
MixerEvent	Not Supported	
MsControlException	Supported	No
MsControlFactory	Supported	Yes
NetworkConnection	Supported	No
Parameter	Supported	No
Parameters	Supported	No
Player	Partially	No
PlayerEvent	Partially	No
PropertyInfo	Partially	No
Qualifier	Partially	No
Recorder	Partially	No
RecorderEvent	Partially	No
Resource	Supported	No
ResourceContainer	Supported	No
ResourceEvent	Supported	No
RTC	Partially	No
SdpException	Not Supported	
SdpPortManager	Partially	Yes
SdpPortManagerEvent	Partially	No

Method	Supported?	Extension
SdpPortManagerException	Partially	No
SignalConstants	Partially	No
SignalDetector	Partially	Yes
SignalDetectorEvent	Partially	No
SignalGenerator	Not Supported	
SignalGeneratorEvent	Not Supported	
SpeechDetectorConstants	Not Supported	
SpeechRecognitionEvent	Not Supported	
SupportedFeatures	Partially	No
TimeoutException	Not Supported	
TooManyJoineesException	Not Supported	
TranscodingException	Not Supported	
Trigger	Not Supported	
UnsupportedException	Supported	No
UnsupportedLayoutException	Not Supported	
Value	Supported	No
VideoLayout	Supported	
VideoRenderer	Supported	
VideoRendererEvent	Partially	
VideoRenderingException	Supported	
VolumeConstants	Not Supported	
VxmlDialog	Not Supported	
VxmlDialogEvent	Not Supported	
WrongStateException	Partially	No

Method Details

This section provides detailed information of each Method and its parameters focused specifically on "**What's not supported**" and any extensions which were added by the JSR 309 Connector. For the complete listing of Methods and its parameters, refer to the JSR 309 standard documentation available at http://www.jcp.org.

AllocationEvent

What's not supported

Field Summary	
static EventType	ALLOCATION_CONFIRMED This EventType is returned by getEventType() to signal the completion of ResourceContainer.confirm(). **Note:** This is only supported on MediaMixer. Refer to the MediaMixer for further details.

JSR 309 Connector Extension:

NC join to mixer followed by nc.processSDP or nc.generateSDP. Provided that this is the first join to the mixer the AllocationEvent will be sent to application notifying it of successful conference creation on Media Server.

CodecConstants

What's not supported

Audio Codecs

ADPCM_32K
ADPCM_32K_OKI
G723_1B
G729_A
EVRC
GSM

JSR 309 Connector Extension:

Video Codecs

Field Summary	
mp4	Dialogic supports MP4 container through INFERRED file format. **Note:** The file extension needs to be defined as .mp4.

DriverManager

Note: In the OCCAS Application Server platform, the driver is registered by a connector implicitly where in other platforms it is done explicitly.

FileFormatConstants

What's not supported

Field Summary	
static Value	FORMAT_3G2 3GPP2 file format.
static Value	GSM GSM file format.

JSR 309 Connector Extension:

Field Summary	
static Value	INFERRED Used when application wants to use MP4 file format. Once the Player.FILE_FORMAT or Recorder.FILE_FORMAT is set to this Value and the file extension is set to mp4, it indicates that the MP4 media container is used.
static Value	RAW Only supports AUDIO type of media. If video is desired INFERRED – for MP4 or 3GP needs to be used as FILE_FORMAT.

JoinableContainer

JSR 309 Connector Extension:

The JSR 309 API has three core JSR API resource containers, NC, MG, and MX, that can be joined together in three different modes, SEND, RECV, and DUPLEX. However, not all combinations are supported.

NC refers to Network Connection and provides functionality to establish media streams. MG refers to Media Group and provides functionality to play, record, and control media content. MX refers to Media Mixer and provides functionality to join media functions to a network connection so as to create conferences and call bridges.

SEND - the media streams can flow from joiner to joinee only.

RECV - the media streams can flow from joinee to joiner only.

DUPLEX - the media streams can flow both ways.

The following table shows the supported configurations:

	NC	MG	MX
NC (DUPLEX)	YES	YES	YES
NC (SEND)	YES	NO	YES
NC (RECV)	YES	NO	YES
MG (DUPLEX)	YES	NO	YES
MG (SEND)	NO	NO	Yes
MG (RECV)	NO	NO	NO
MX (DUPLEX)	YES	YES	NO
MX (SEND)	YES	NO	NO
MX (RECV)	YES	NO	NO

The following join combinations are the same and can be used to set up a full duplex connection. In the examples below, a Media Mixer is connected in full duplex to a Network Connection for play/record capability.

```
networkConnection.join(Direction.RECV, mixer)
mixer.join(Direction.SEND, networkConnection)

networkConnection.join(Direction.SEND, mixer)
mixer.join(Direction.RECV, networkConnection)

networkConnection.join(Direction.DUPLEX, mixer)
mixer.join(Direction.DUPLEX, networkConnection)
```

As per the JSR 309 Connector implementation for any above join combination, make note that the Allocation Event is always sent to the Network Connection component.

JoinEvent

What's not supported

Field Summary	
static MediaErr	NO_TRANSCODER Error sent by media server when it is unable to perform the transcoding required by the join.
static MediaErr	TOO_MANY_JOINEES Error sent by media server when the number of joined objects is too high (value is implementation dependent).

MediaConfig

What's not supported

Method Summary	
MediaConfig	createCustomizedClone(Parameters params) Create a new MediaConfig, altering the given parameters with the given values.
java.lang.String	marshall()

MediaGroup

What's not supported

Field Summary	
static Configuration<MediaGroup>	PLAYER_RECORDER_SIGNALDETECTOR_SIGNALGENERATOR MediaGroupConfig containing Player, Recorder, SignalDetector, and SignalGenerator.
static RTC	SIGDET_STOPPLAY The common RTC to stop a prompt when a DTMF is detected.
static RTC	SIGDET_STOPRECORD The common RTC to stop a recording when a DTMF is detected.
static Configuration<MediaGroup>	SIGNALDETECTOR Defines a MediaGroupConfig containing only a SignalDetector. **Note:** Refer to the JSR 309 Connector Extension below.

Method Summary	
SignalGenerator	getSignalGenerator() Returns the SignalGenerator of this MediaGroup.
void	stop() Stops all operations currently in progress on this MediaGroup.

Methods inherited from javax.media.mscontrol.resource.ResourceContainer	
confirm	Request that all pending allocations/initializations are completed.
getConfig	
triggerAction	Triggers a RTC action, requested by the application.

Methods inherited from javax.media.mscontrol.MediaObject	
release	Release the resources associated to this media object.

JSR 309 Connector Extension:

static Configuration<MediaGroup>	SIGNALDETECTOR **Note:** The JSR 309 Connector creates a MediaGroup as a PLAYER_SIGNALDETECTOR.

Note: The getPlayer(), getRecorder(), getSignalDetector() will return the same Object reference for each resource request of the same MediaGroup.

MediaMixer

What's not supported

Field Summary	
static Configuration<MediaMixer>	AUDIO_EVENTS This Configuration supports audio mixing, plus the events related to the active talkers.
static Configuration<MediaMixer>	AUDIO_VIDEO_RENDERING This Configuration supports the features of AUDIO_VIDEO, plus a VideoRenderer.
static Parameter	ENABLED_EVENTS An array of Mixer EventTypes, indicating which events are generated and delivered to the application, including MOST_ACTIVE_INPUT_CHANGED.

Methods inherited from javax.media.mscontrol.resource.ResourceContainer	
getConfig	
getResource	Return a handle on a resource of type resource.
triggerAction	Triggers a RTC action, requested by the application.

JSR 309 Connector Extension:

Method Summary	

void setAttribute	connector.asn.louder.sample.time
	Set the asn sampling time by using the Mixer Media Session. For example:
	`myMs.setAttribute("connector.asn.louder.sample.time", new Integer(5));`
	CONFERENCE_VIDEO_SIZE
	Set the root conference size. For example:
	`myMixerMediaSession.setAttribute("CONFERENCE_VIDEO_SIZE", "720P");`

void setAttribute	mediaMixerMode set to "sfu" configures a mixer to be configured as an SFU conference instead of the default MCU. For example: `myMixerMediaSession setAttribute("mediaMixerMode", "sfu");` **Note:** When not set, the mode is MCU.
Event	NETWORK_STREAM_FAILURE This event extends JSR 309 by providing an application with notifications from Media Server of SIP Session Timer expiration. This event tells the application that the refresher SIP UPDATE has not been received within a negotiated time. The session is released, and the application should be cleaned up accordingly. When SIP Session Timeout is triggered, an application using JSR 309 will be notified with above event with following details: Event.getError() = "TIMEOUT" Event.getErrorText() = "RFC-4028 Session Timeout - releasing session" **Note:** This event is generated on the AllocationEventListener of the mixer.

Mixer Limitations:

Side-bar and coach-pupil conference to conference formats are not supported.

MediaSession

What's not supported

Field	
static Parameter	TIMEOUT The value of this Parameter is an Integer, in milliseconds, after which an API call (on the objects created by this MediaSession) should timeout.

Method	
VxmlDialog	createVxmlDialog (Parameters parameters) VxmlDialog is a Joinable object that can interpret a VXML script.

JSR 309 Connector Extension:

Method	
createNetworkConnection()	Supported only for MediaSession which is not dedicated to a mixer (i.e., where createMediaMixer() method is to be used).

Parameter	
NC_VIDEO_SIZE	Parameter which allows to set the ceiling for video resolution for NC. If video resolution already at the same or level SDP is unchanged. Applicable to h264, VP8, and mp4v-es ONLY. Default: "720p" Supported values: • "720p" • "VGA" • "CIF" • "QCIF"

Parameter	
PLAYER_MODE	Defines the player mode for the MediaSession. Default: "AUDIO_VIDEO" Supported values: • "AUDIO" • "VIDEO" • "AUDIO_VIDEO"
RECORDER_MODE	Defines the recorder mode for the MediaSession. Default: "AUDIO_VIDEO" Supported values: • "AUDIO" • "VIDEO" • "AUDIO_VIDEO"

CAPTION	Used in video connection for a caption text to be displayed towards the NC joined to MS. Used to provide text identifying each participant in a bridge type of connection (NC joined to another NC). For example:
	To set it:
	`part.mMediaSession.setAttribute("CAPTION", "My Name");`
	`part.mNetworkConnection.joinInitiate(Direction.DUPLEX, partSecond.mNetworkConnection, this);`
	To remove it:
	`part.mMediaSession.removeAttribute("CAPTION");`
	`part.mNetworkConnection.joinInitiate(Direction.DUPLEX, partSecond.mNetworkConnection, this);`
	Note: CAPTION can be manipulated throughout the entire call duration by calling a join or joinInitiate method again.

MixerAdapter

What's not supported

Field Summary	
static Configuration<MixerAdapter>	DTMFCLAMP_VOLUME This config clamps the DTMF's that would otherwise enter the Mixer, and include a volume control.

Field Summary	
static Configuration<MixerAdapter>	EMPTY This config is a pass-through, no media processing is performed on any stream.

Methods inherited from javax.media.mscontrol.resource.ResourceContainer	
confirm	Request that all pending allocations/initializations are completed.
getConfig	
getResource	Return a handle on a resource of type resource.
triggerAction	Triggers a RTC action, requested by the application.

Methods inherited from javax.media.mscontrol.MediaObject	
release	Release the resources associated to this media object.

MsControlFactory

What's not supported

Field Summary	
static java.lang.String	MEDIA_SERVER_URI **Note:** Refer to the JSR 309 Connector Extension below.

Method Summary	
MediaConfig	getMediaConfig(java.io.Reader xmlDef) Create an instance of MediaConfig, from an xml byte stream (i.e., a file or a String). For example: Reader xmlDoc = new StringReader("<?

JSR 309 Connector Extension:

Field Summary	
createMediaSession()	It is strongly recommended that created MediaSession is one-to-one with its NetworkConnection or Mixer. Calling createNetworkConnection() on MediaSession which is dedicated to a mixer (i.e., createMediaMixer() is not supported).
static java.lang.String	MEDIA_SERVER_URI MsControlFactory property defining an URI of a media server, for example sip:ms@192.168.1.2:5060. Preferred way is to use the factory property configuration. See dynamic Media Server connector configuration section for more details. When NULL, the Media Server definition from the property file is used. **Note:** The JSR 309 Connector Media Server Redundancy feature cannot be used and needs to be turned off. Refer to Setting the Media Server URI Programmatically for further details.

NetworkConnection

What's not supported

Field Summary	
static Configuration<NetworkConnection>	DTMF_CONVERSION Contains what BASIC contains, plus a SignalDetector and a SignalGenerator for forwarding DTMFs carried by the signaling channel (advanced feature).
static Configuration<NetworkConnection>	ECHO_CANCEL Contains what BASIC contains, plus an echo canceler feature.

Note: Refer to the Multiple NCs per Media Session versus Single NC per Media Session Model section under Development Considerations.

Methods inherited from javax.media.mscontrol.resource.ResourceContainer	
confirm	Request that all pending allocations/initializations are completed. See JSR 309 Connector Extension table for a usage of this method.

Methods inherited from javax.media.mscontrol.resource.ResourceContainer	
getConfig	
triggerAction	Triggers a RTC action, requested by the application.

JSR 309 Connector Extension:

Method Summary	
confirm	On a supported MediaSession NC (not dedicated to a mixer) when created, an application can use the confirm() method when the application wants to use Media Server "Ping" feature. It is recommended for application to create a dedicated MediaSession and then create a NC for this "Ping" feature. Using confirm() on a NC which has not been established/connected yet will generate Out of Dialog OPTIONS "Ping" SIP message. Calling confirm() on already established connection will result in In Dialog OPTIONS "Ping" SIP message. A successful response to confirm() ("Ping") will be represented by the ALLOCATION_CONFIRMED event. A failed "Ping" response to a desired NC will be represented by the IRRECOVERABLE_FAILURE event.

Method Summary	
setParameters	JSR 309 Connector uses several propriaiatary parameters to expand the connector capabilities from its specification: "SIP_REQ_URI_USERNAME" - (string) Allows an application to change the userpart of URI used to communicate with Dialogic PowerMedia XMS. By default the connector uses "**msml=<rand>**@<MS IP>:<MS_PORT>" where <rand> is connector random generated number. By setting this parameter application can modify userpart of URI to any desired value for each networkConnection. **Note:** PowerMedia XMS by default expects a user part to start with "msml" which requires the application, when setting this parameter, to prepend it with "msml". For information how to change PowerMedia XMS default behavior, refer to the PowerMedia XMS documentation. Use cases: It is often used as a method which allows particular call to be "custom tagged" so that equipment sitting between Application Server and Media Server is able to make appropriate decisions. For example, direct specific connections to appropriate set of PowerMedia XMS, etc. "cpa" – ("yes") Sets the Call Progress Analysis flag to true during CPA "on" when processOfferSdp() method is called. Below is an example of how to enable the CPA and Media Route Profile features of PowerMedia XMS by utilizing an extension of the JSR 309 NetworkConnection setParameters API. "mediarouteprofile" – (string) This parameter provides support for PowerMedia XMS Multi NIC Support via Media Route Profile Configuration. "Session-ID" – (string) This parameter is used when it is desired to customize a standard based SIP header random value. This can be used as a custom tag a specific networkConnection. It is a same concept as "SIP_REQ_URI_USERNAME" referenced above, but instead defining a unique string as a standards based SIP header: "Session-ID". Default: JSR 309 Connector automatically identifies each call leg with a random generated value. Refer to RFC 7329 for details on the Session-ID value.

Example

"SIP_REQ_URI_USERNAME"

```
Parameters hParameters = mediaSession.createParameters();
Map<String,String> configurationData = new HashMap<String,String>();
configurationData.put("SIP_REQ_URI_USERNAME", "msml=TimeZoneEST");
log.debug("setting SIP_REQ_URI_USERNAME: " + configurationData.get("SIP_REQ_URI_USERNAME"));
hParameters.put(SdpPortManager.SIP_HEADERS, configurationData);
networkConnection.setParameters(hParameters);
```

"cpa"

To enable Call Progress Analysis detection on this specific PowerMedia XMS connection, the following steps must be taken:

```
Parameters sdpConfiguration = mediaSession.createParameters();
Map<String,String>  configurationData = new HashMap<String,String>();
configurationData.put("cpa","yes"); //Enable CPA
sdpConfiguration.put(SdpPortManager.SIP_HEADERS, configurationData);
networkConnection.setParameters(sdpConfiguration);
```

Once the Call Progress Analysis completes, the application will be notified of its results as seen in the example below:

```
public void onEvent(T anEvent)
{
  log.debug("received event =" + anEvent.getEventType().toString() );
  Resource r = (Resource)anEvent.getSource();
  ResourceContainer container = r.getContainer();
  MediaSession ms = container.getMediaSession();
  else if ( anEvent.getEventType() == SdpPortManagerEvent.UNSOLICITED_OFFER_GENERATED ) {
    log.debug("Received UNSOLICITED_OFFER_GENERATED");
    String cpaData = (String)ms.getAttribute("CPA_DATA");
    if ( cpaData == null ) {
      log.debug("Media Server re-Invite received");
    }
    else{
      log.debug("Event indicating completion of CPA: " + cpaData);
       ms.removeAttribute("CPA_DATA");       //clear attribute after using it.
    }
  }
}
```

"mediarouteprofile"

Media Server Multi NIC Support via Media Route Profile Configuration. JSR 309 Connector extends the capability of its API so that each call can be mapped to a specific network interface as defined by the PowerMedia XMS Media Route Profiles. A custom value is defined for the pre-defined key (mediarouteprofile), which is then inserted as part of the Request-URI.

The application can specify a custom value for a "mediarouteprofile" key. For example: "nic1".

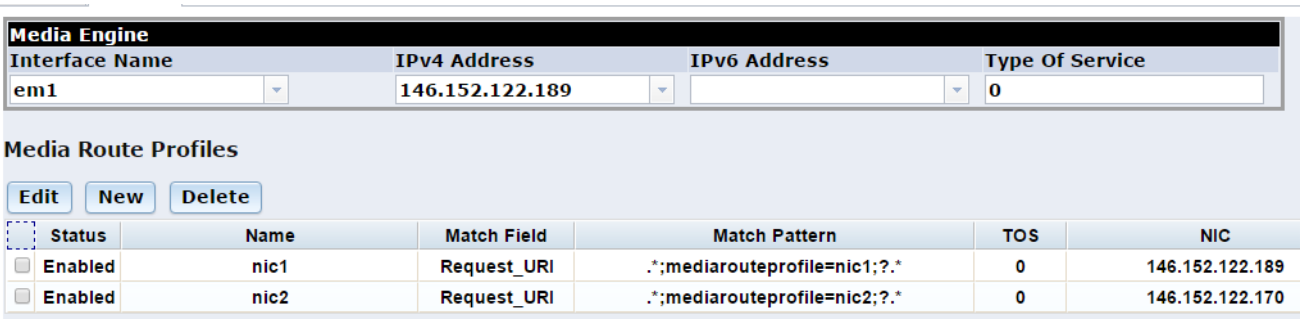
```
try {
```

```
        MediaSession mediaSession = mscFactory.createMediaSession();
        Parameters parameters = mediaSession.createParameters();
        Map<String,String> configurationData = new HashMap<String,String>();
        configurationData.put("mediarouteprofile", "nic1");
        log.debug("Test setting SIP_HEADER" + mediarouteprofile);
        parameters.put(SdpPortManager.SIP_HEADERS, configurationData);
        networkConnection = mediaSession.createNetworkConnection(NetworkConnection.BASIC);
        networkConnection.setParameters(parameters);
        networkConnection.getSdpPortManager().addListener(sdpPortsListener);
}
```

This will create a key/value pair to be inserted as part of Request-URI for this processSdpOffer as follows:

```
INVITE sip:msml=395@<XMSMediaServerIP>:5060;mediarouteprofile=nic1;transport=udp SIP/2.0
"nic1"
```

This now allows that call to be mapped to a specific NIC as defined by Media Route Profiles in PowerMedia XMS.



Note: If the incoming call request does not match any defined Media Route Profiles pattern, the default interface for media will be used (as defined in the **Interface Name** field shown above).

"Session-ID"

Standards based SIP Header used to set custom ID when sending requests to Media Server. Below is an example on how to set it:

```
Parameters hParameters = mediaSession.createParameters();
Map<String,String> configurationData = new HashMap<String,String>();
configurationData.put("Session-ID", "MyCustomSessionId");
log.debug("Setting SIP_HEADERS - Session-ID: " + configurationData.get("Session-ID"));
hParameters.put(SdpPortManager.SIP_HEADERS, configurationData);
networkConnection.setParameters(hParameters);
```

Application can retreieve value of Session-ID anytime when it receives an event from connector as shown below. If application did not set, the value of the retrieval will provide a value automatically set by JSR 309 Connector.

```
static class SdpPortsListener implements MediaEventListener<SdpPortManagerEvent>, Serializable{
  public void onEvent(final SdpPortManagerEvent event){
    final MediaSession mediaSession = event.getSource().getMediaSession();
    NetworkConnection networkConnection = (NetworkConnection)
mediaSession.getAttribute("NETWORK_CONNECTION");
    try{
```

```
      if (event.isSuccessful()){
        log.debug("Getting SIP_HEADERS....");
        Parameters localParameters = networkConnection.getParameters(null);
        HashMap<String, String> myHashMap = (HashMap<String, String>)
localParameters.get(SdpPortManager.SIP_HEADERS);
        log.debug("Getting SIP_HEADERS - Session-ID: " + myHashMap.get("Session-ID"));
    }
```

Player

What's not supported

Field Summary	
static Parameter	BEHAVIOUR_IF_BUSY Indicates the action to take if Player is busy when play() is invoked.
static Parameter	ENABLED_EVENTS An array of Player EventTypes, indicating which events are generated and delivered to the application.
static Value	FAIL_IF_BUSY Value for BEHAVIOUR_IF_BUSY: signal an error, throw a MediaResourceException.
static Action	JUMP_BACKWARD Jump backward by the amount specified by JUMP_TIME.
static Action	JUMP_BACKWARD_IN_PLAYLIST Jump backward the number of items indicated by JUMP_PLAYLIST_INCREMENT.
static Action	JUMP_FORWARD Jump forward by the amount specified by JUMP_TIME.
static Action	JUMP_FORWARD_IN_PLAYLIST Jump forward the number of play list items indicated by JUMP_PLAYLIST_INCREMENT.
static Parameter	JUMP_PLAYLIST_INCREMENT Integer number of items to jump forward or backward in the play list, for either a JUMP_FORWARD_IN_PLAYLIST or JUMP_BACKWARD_IN_PLAYLIST.
static Parameter	JUMP_TIME Integer number of milliseconds by which the current play list items offset is changed by JUMP_FORWARD or JUMP_BACKWARD.

Field Summary	
static Action	JUMP_TO_PLAYLIST_END Jump to the last item in the play list.
static Action	JUMP_TO_PLAYLIST_ITEM_END Jump to the end of the current play list item.
static Action	JUMP_TO_PLAYLIST_ITEM_START Jump to the start of the current play list item.
static Action	JUMP_TO_PLAYLIST_START Jump to the first item in the play list.
static Action	NORMAL_SPEED Set speed to normal.
static Action	NORMAL_VOLUME Set volume to normal.
static Action	PAUSE Pause the current Play operation, maintaining the current position in the play list item and play list.
static Trigger	PLAY_COMPLETION Trigger when a Play is completed.
static Trigger	PLAY_START Trigger when a Play is started.
static Value	QUEUE_IF_BUSY Value for BEHAVIOUR_IF_BUSY: wait for previous requests to complete.
static Action	RESUME Resume the current Play operation if paused.
static Action	SPEED_DOWN Decrease speed.
static Action	SPEED_UP Increase speed.
static Parameter	START_IN_PAUSED_MODE Boolean indicating that this or subsequent play should be started in the Paused state.

Field Summary	
static Action	STOP Stop the current Play operation.
static Action	STOP_ALL Stop the current Play operation and all pending play operations in queue.
static Value	STOP_IF_BUSY Value for BEHAVIOUR_IF_BUSY: stop any previous play.
static Action	TOGGLE_VOLUME Toggle volume between normal and previous adjusted value.
static Parameter	VOLUME_CHANGE Determines the amount by which the volume is changed by the RTC actions VOLUME_UP and VOLUME_DOWN.
static Action	VOLUME_DOWN Decrease volume by value of parameter VOLUME_CHANGE.
static Action	VOLUME_UP Increase volume by value of parameter VOLUME_CHANGE.

JSR 309 Connector Extension:

Field Summary		
static Parameter	MAX_DURATION	Integer indicating maximum duration of a play request. Value is a positive Integer, in milliseconds, or Resource.FOR_EVER (default). *FOR_EVER is related to the duration of the file being played, the FOR_EVER truly means play till the end of the file. JSR 309 implementation extended the connector to provide capability for application to play a file forever. The play continues, up until the time the application issues a stop. To achieve the play forever this parameter needs to be set to -2.*
static Parameter	REPEAT_COUNT	JSR 309 Connector does not support repeat the play forever. If such functionality is needed, refer to the MAX_DURATION extension above.

When playing, the following steps need to be taken:

1. Verify that MediaSession PLAYER_MODE is set appropriately
2. Desired Player FILE_FORMAT is chosen

3. Desired Player AUDIO_CODEC are specified – required only for headerless media file formats – otherwise ignored
4. URI

Example

Below is an example extracted from Dialogic Verification Demo source, which is available as a project as part of JSR 309 distribution.

```
static class PlayerJoinEventListener implements JoinEventListener, Serializable {
        private static final long serialVersionUID = 1L;
        public void onEvent(final JoinEvent event) {
                log.debug("Entering");
                final MediaGroup mg = (MediaGroup) event.getSource();
                log.debug("Join operation completed SUCCESSFULLY: "  + event.isSuccessful());
                try {
                        String promptFileName = (String)event.getContext();
                        log.info("Play(" + promptFileName + ")");
                        try {
                                mg.getPlayer().play(prompt,RTC.NO_RTC, Parameters.NO_PARAMETER);
                        } catch (MsControlException e) {
                                log.error("MsControlException" + e.toString(), e);
                        }
                } catch (Exception e) {
                        log.error("Exception" + e.toString(), e);
                }
                log.debug("Exiting");
        }
}
```

PlayerEvent

What's not supported

Field Summary	
static EventType	PAUSED Play has been paused by RTC.
static EventType	RESUMED Play has been resumed by RTC.
static EventType	SPEED_CHANGED Playback speed has been changed due to RTC.
static EventType	VOLUME_CHANGED Playback volume has been changed due to RTC.

Method Summary	
Action	getChangeType() Return an Action that identifies the type of Speed or Volume adjustment.
int	getIndex() Return the index into the play list, indicating which play list item was stopped, or paused, etc.
int	getOffset() Return the milliseconds offset (from start) where the play stopped or paused.

Methods inherited from javax.media.mscontrol.resource.ResourceEvent	
getRTCTrigger	Get the RTC Trigger that caused this transaction completion.

Example

Below is an example extracted from Dialogic Verification Demo source, which is available as a project as part of JSR 309 distribution.

```
static class PlayerEventListener implements MediaEventListener<PlayerEvent>, Serializable
{
        private static final long serialVersionUID = 1L;
        public void onEvent(final PlayerEvent event) {
                String logMsg = null;
                log.debug("event = " + event);
                if (event.getEventType().equals(PlayerEvent.PLAY_COMPLETED)) {
                        logMsg = "PLAY_COMPLETED";
                        if (event.getQualifier().equals(PlayerEvent.DURATION_EXCEEDED)) {
                                logMsg = logMsg + " - DURATION_EXCEEDED";
                        }
                        else if (event.getQualifier().equals(PlayerEvent.END_OF_PLAY_LIST)) {
                                logMsg = logMsg + " - END_OF_PLAY_LIST";
                                final MediaSession mediaSession =
event.getSource().getMediaSession();
                                final SipSession sipSession = (SipSession)
mediaSession.getAttribute("SIP_SESSION");
                                try {
                                        terminateSession(sipSession, mediaSession);
                                } catch (Exception e) {
                                        log.error("Exception" + e.toString(), e);
                                }
                        }
                }
```

```
                else if (event.getEventType().equals(PlayerEvent.PAUSED)) {
                        logMsg = "PAUSED";
                }
                else if (event.getEventType().equals(PlayerEvent.RESUMED)) {
                        logMsg = "RESUMED";
                }
                else if (event.getEventType().equals(PlayerEvent.SPEED_CHANGED)) {
                        logMsg = "SPEED_CHANGED";
                }
                else if (event.getEventType().equals(PlayerEvent.VOLUME_CHANGED)) {
                        logMsg = "VOLUME_CHANGED";
                }
                log.info("Play terminated: " + logMsg);
        }
}
```

Recorder

What's not supported

Field Summary	
static Parameter	APPEND Indicates that recording should append to the end of an existing TVM rather than overwrite it.
static Parameter	AUDIO_FMTP A string-valued list of detailed audio codec parameters, in the format described by RFC 4566.
static Parameter	AUDIO_MAX_BITRATE The maximum accepted bitrate for the audio stream.
static Parameter	BEEP_FREQUENCY The frequency of the start beep.
static Parameter	BEEP_LENGTH Length of Beep preceding recording.
static Action	CANCEL Cancel the current recording operation.
static Value	DETECT_ALL_OCCURRENCES Value for SPEECH_DETECTION_MODE.
static Value	DETECT_FIRST_OCCURRENCE Value for SPEECH_DETECTION_MODE.

Field Summary	
static Value	DETECTOR_INACTIVE Value for SPEECH_DETECTION_MODE.
static Parameter	ENABLED_EVENTS An array of Recorder EventTypes, indicating which events are generated and delivered to the application.
static Action	PAUSE Pause the current operation on a recorder, maintaining the current position in the Media Stream being recorded.
static Trigger	RECORD_COMPLETION Trigger when a Record is completed.
static Action	RESUME Resume the current operation on a recorder if paused.
static Parameter	SIGNAL_TRUNCATION_ON Boolean indicating whether signal(DTMF) truncation is enabled.
static Parameter	START_IN_PAUSED_MODE Boolean indicating whether subsequent record will start in PAUSE mode.
static Action	STOP Stop the current recording operation.
static Parameter	VIDEO_MAX_BITRATE The maximum accepted bitrate for the video stream.

JSR 309 Connector Extension:

Field Summary		
static Parameter	START_BEEP	Record with beep is not supported while a party is joined to another participant or a conference. Doing so might result in frozen video or join connection not taking place. **Note:** JSR 309 Connector enables beep in record by default. In the above scenario, the application needs to modify the parameter to disable beep in record in order to avoid this limitation.

When recording, the following steps need to be taken:

1. Verify that MediaSession RECORDER_MODE is set appropriately
2. Desired Recorder FILE_FORMAT is chosen
3. Desired Recorder AUDIO_CODEC and/or VIDEO_CODEC are specified
4. URI

JSR 309 Connector Extension:

Supported AUDIO_CLOCKRATE values when AUDIO_CODEC is set to AMR

In Hz
4750 - default
5150
5900
6700
7400
7950
10200
12200

Supported AUDIO_CLOCKRATE values when AUDIO_CODEC is set to AMR_WB

In Hz
6600 - default
8850
12650
14250
15850
18250
19850
23050
23850

Note: For other codecs, AUDIO_CLOCKRATE is chosen automatically based on AUDIO_CODEC and will be ignored when set.

VIDEO_FMTP following key-value pairs are supported for various video codecs:

H.263 / H.263- 1998

Resolution	profile	level	width	height	framerate
CIF	0	10	352	288	10
CIF	0	20	352	288	15
CIF	0	30	352	288	30
QCIF	0	10	176	144	15
QCIF	0	20	176	144	30

H.264

Resolution	profile	level	width	height	framerate
CIF	0	1.2	352	288	15
CIF	0	1.3	352	288	30
CIF	0	1.3	352	288	30
CIF	0	2.0	352	288	30
CIF	0	2.1	352	288	30
CIF	0	2.2	352	288	25
HD720p	0	3.1	1280	720	30
QCIF	0	1.0	176	144	15
QCIF	0	1.1	176	144	15
QCIF	0	1.1	176	144	30
VGA	0	1.2	640	480	15
VGA	0	3.0	640	480	25

MPEG-4

Resolution	profile	level	width	height	framerate
CIF	0	2	352	288	15
CIF	0	3	352	288	15
QCIF	0	0	176	144	15
QCIF	0	1	176	144	15
QVGA	0	3	320	240	30
VGA	0	4	640	480	30

Example

```
public void startRecording() throws MsControlException {
        logger.debug("Entering");
        this.getmMediaSession().setAttribute("RECORDER_MODE", "AUDIO_VIDEO");
        Integer nanoSecs =  new Integer( (int) System.nanoTime());
        String  sNanoSecs = nanoSecs.toString();
```

```
        String  participantType = "-Customer-";

        if ( this instanceof Agent ) {
                participantType = "-Agent-";
        }

        String m_currentRecFile = BridgeRecordingServlet.record_prompt + this.getConfId() +
this.getID() + participantType + sNanoSecs +".mp4";
        logger.debug("Recording File Name: " + m_currentRecFile);
        URI recordingDestURI = URI.create(m_currentRecFile);
        Parameters params = BridgeRecordingServlet.theMsControlFactory.createParameters();
        params.put(SignalDetector.PATTERN[0], "#");
        //params.put(Recorder.FILE_FORMAT, FileFormatConstants.WAV);
        params.put(Recorder.AUDIO_CODEC, CodecConstants.AMR);
        params.put(Recorder.VIDEO_CODEC, CodecConstants.H264);
        params.put(Recorder.START_BEEP, Boolean.FALSE);
        String sVideoFMTP="";
        sVideoFMTP+="profile="+"66";
        sVideoFMTP+=";level="+"3.1";
        sVideoFMTP+=";width="+"1280";
        sVideoFMTP+=";height="+"720";
        sVideoFMTP+=";framerate="+"15";
        params.put(Recorder.VIDEO_FMTP, sVideoFMTP);
        params.put(Recorder.VIDEO_MAX_BITRATE, 2000);

        logger.debug("Calling Record Transaction Start");
        this.getmMediaGroup().getRecorder().record(
                recordingDestURI,
                new RTC[] {
                        // barge-in
                        new RTC(SignalDetector.DETECTION_OF_ONE_SIGNAL, Player.STOP),
                        // stop recording key
                        new RTC(SignalDetector.PATTERN_MATCH[0], Recorder.STOP)
                }, params);
        logger.debug("Exiting");
}
```

RecorderEvent

What's not supported

Field Summary	
static EventType	PAUSED Indicates that recording has been paused.

Field Summary	
static EventType	RESUMED Indicates that recording has been resumed.

Methods inherited from javax.media.mscontrol.resource.ResourceEvent	
getRTCTrigger	Get the RTC Trigger that caused this transaction completion.

SdpPortManager

JSR 309 Connector Extension:

Method Summary

void

generateSdpOffer()

PowerMedia XMS, in response to the generateSdpOffer() method, will generate an answer with offer SDP. By default, the standard SDP will be generated. In order to request a specific SDP configuration (like SDP for WebRTC client as an example), the application needs to specify what it needs. From each category, the application can choose a desired value for supported key.

If not set, the default values for each category are disabled. No encryption, NAT, and RTCP feedback will be configured in SDP.

Category	Value	Description
Encryption	dlgc-encryption-sdes	Enables sdes-srtp
	dlgc-encryption-dtls	Enables dtls-srtp
NAT	dlgc-ice	Enables ICE (Lite)
Feedback	dlgc-rtcp-feedback-audio	Enables AVPF/SAVPF for audio (not currently supported)
	dlgc-rtcp-feedback-video	Enables AVPF/SAVPF for video
	dlgc-rtcp-feedback-audiovideo	Enables AVPF/SAVPF for audio and video (only video is currently supported)
	dlgc-rtcp-feedback-none	Disables RTCP feedback messages for both audio and video.

Example: Setting MediaServer to generate offer SDP for WebRTC

```
nc = m_ncMediaSession.createNetworkConnection(NetworkConnection.BASIC);
Parameters sdpConfiguration = m_ncMediaSession.createParameters();
Map<String,String>  configurationData = new HashMap<String,String>();
//Enable DTLS, ICE Lite, and RTCP feedback
configurationData.put("Supported","dlgc-encryption-dtls, dlgc-ice, dlgc-rtcp-feedback-audiovideo");
```

```
sdpConfiguration.put(SdpPortManager.SIP_HEADERS, configurationData);
networkConnection.setParameters(sdpConfiguration);….

//Send request to XMS to GENERATE SDP
networkConnection.getSdpPortManager().generateSdpOffer();
```

SdpPortManagerEvent

What's not supported

Field Summary	
static Qualifier	OFFER_PARTIALLY_ACCEPTED ResourceEvent.getQualifier() returns this Qualifier to indicate that the offer is partially accepted (at least 1 Media description is accepted). The application must use SdpPortManager.getMediaServerSessionDescription() to know what was accepted and what was rejected.
static Qualifier	RESOURCE_PARTIALLY_AVAILABLE ResourceEvent.getQualifier() returns this Qualifier to indicate that a media resource cannot be allocated from the Media Server.
static MediaErr	RESOURCE_UNAVAILABLE Error sent by media server in case of resource shortage.
static MediaErr	SDP_GLARE Error reported by the media server when both sides of the media channel attempted to re-negotiate the SDP at the same time.
static MediaErr	SDP_NOT_ACCEPTABLE Error sent when the offer/answer procedure does not find any matching codec for any stream.

Methods inherited from javax.media.mscontrol.resource.ResourceEvent	
getRTCTrigger	Get the RTC Trigger that caused this transaction completion.

JSR 309 Connector Extension:

Field Summary	
Event	NETWORK_STREAM_FAILURE: JSR 309 extends this event: 1. Provides application with notifications from Media Server of RTP/RTCP timeouts. When RTP or RTCP alarm is triggered, an application using JSR 309 will be notified with above event with following details: event.getError() = "TIMEOUT" event.getErrorText() = "AUDIO_RTP_ON" "AUDIO_RTCP_ON" "VIDEO_RTP_ON" "VIDEO_RTCP_ON" "AUDIO_RTP_OFF" "AUDIO_RTCP_OFF" "VIDEO_RTP_OFF" "VIDEO_RTCP_OFF" **Note:** RTP/RTCP alarms need to be enabled on Media Server. Refer to the PowerMedia XMS documentation for details. 2. Provides application with notifications from Media Server of SIP Session Timer expiration. This event tells the application that the refresher SIP UPDATE has not been received with in a negotiated time. With that, the session is released and application should cleaned up accordingly. When SIP Session Timeout is triggered, an application using JSR 309 will be notified with above event with following details: Event.getError() = "TIMEOUT" Event.getErrorText() = "RFC-4028 Session Timeout - releasing session"

Example: NETWORK_STREAM_FAILURE

```
if (SdpPortManagerEvent.NETWORK_STREAM_FAILURE.equals(event.getEventType()))
{
  log.debug("Event: NETWORK_STREAM_FAILURE");
  inv = (SipServletRequest) sipSession.getAttribute("UNANSWERED_INVITE");
  if (SdpPortManagerEvent.RESOURCE_UNAVAILABLE.equals(event.getError())){
    log.debug("Event Error: RESOURCE_UNAVAILABLE - " + event.getErrorText());
    SipServletRequest sipServletRequest = sipSession.createRequest("BYE");
```

```
    sipServletRequest.send();
    mediaSession.release();
  }else if (SdpPortManagerEvent.SDP_GLARE.equals(event.getError())){
    log.debug("Event Error: SDP_GLARE - " + event.getErrorText());
    inv.createResponse(SipServletResponse.SC_SERVER_INTERNAL_ERROR).send();
    sipSession.removeAttribute("MEDIA_SESSION");
    mediaSession.release();
  }else if (SdpPortManagerEvent.SDP_NOT_ACCEPTABLE.equals(event.getError())){
    log.debug("Event Error: SDP_NOT_ACCEPTABLE - " + event.getErrorText());
    inv.createResponse(SipServletResponse.SC_NOT_ACCEPTABLE_HERE).send();
    sipSession.removeAttribute("MEDIA_SESSION");
    mediaSession.release();
  }else if (event.getError().equals("TIMEOUT")){
    log.info("MS TIMEOUT: " + event.getErrorText());
  }
}
```

SignalConstants

What's not supported

Field Summary	
static Value	CED_TONE Value for CED. 2100Hz tone answer to CNG_TONE.
static Value	CNG_TONE Value for CNG. 1100Hz tone answered by CED_TONE.
static Value	VFU_REQUEST Value representing a request for a **V**ideo**F**ast**U**pdate.

SignalDetector

What's not supported

Field Summary	
static Parameter	BUFFER_SIZE Size of the signal buffer.
static Parameter	BUFFERING Enable/disable buffering of signals.

Field Summary	
static Action	CANCEL RTC Action to cause the cancellation of the current receiveSignals operation.
static Trigger	DETECTION_OF_ONE_SIGNAL Some signal was detected.
static Parameter	ENABLED_EVENTS Array of SignalGenerator event types, indicating which events are generated for the application.
static Action	FLUSH_BUFFER RTC Action to cause the buffer to be flushed.
static Trigger	FLUSHING_OF_BUFFER Buffer has been flushed.
static Trigger	RECEIVE_SIGNALS_COMPLETION The receiveSignals transaction has completed.
static Action	STOP RTC Action to cause the current receiveSignals operation to stop.

JSR 309 Connector Extension:

Field Summary	
Static Parameter	PROMPT Indicates a prompt to be played allowing signal detection in parallel but delaying INITIAL_TIMEOUT until after prompt completes.

Method Summary	
void	flushBuffer() Removes all signals from the signal buffer. Refer to the details on specific JSR 309 Connector implementation below.

Method Summary	
void	receiveSignals(int numSignals, Parameter[] patternLabels, RTC[] rtcs, Parameters optargs) Refer to the details on specific JSR 309 Connector implementation below.
void	stop() Refer to details on specific JSR 309 Connector implementation below.

The flushBuffer() method is supported by the JSR 309 Connector and used to indicate to the SignalDetector to flush any digits before executing receiveSignals request. When the flushBuffer() method is not called, the next receiveSignals() request will take existing digits (if they exist) as part of its signal detection logic.

Note: The flushBuffer() method needs to be called each time to clear the buffer before signal detection takes place. Otherwise, the default path is executed where existing digits in the buffer will not be cleared before collection.

Example

```
mg.getSignalDetector().flushBuffer();
mg.getSignalDetector().receiveSignals(testPropNumOfSign, detectDigitPattern, rtcs,
collectOptions);
```

The receiveSignals() method is supported by the JSR 309 Connector in the following variations:

- **Collect**
  The receiveSignals() collects any numSignals > 0 with optional patterns. Once a signal is detected, the detection process will be terminated. When terminated, the RECEIVE_SIGNALS_COMPLETED event will be sent.
- **Prompt and Collect**
  The receiveSignals() will collect specified signals >0 with optional patterns during the play. Once either appropriate signals are detected or play has reached the end, the receiveSignals() method will be terminated. When terminated, the RECEIVE_SIGNALS_COMPLETED event will be sent.
- **Async DTMF Detection**
  The receiveSignals() extends the signal detection functionality by providing what is known as Asynchronous (Async) DTMF detection. Async DTMF detection is supported by the JSR 309 Connector by extending the SignalDetector receiveSignals() semantics. Async DTMF allows the application to program the Media Server to generate a single DTMF event every time a DTMF digit is pressed. The single digit detection is enabled until the application decides to turn it off by using stop() method. Once the stop() method is called, the RECEIVE_SIGNALS_COMPLETED event will be sent to notify application that Async DTMF detection has been stopped.

  The following method parameters need to be defined in order to use the receiveSignals() method for Async DTMF detection:

```
//Setup Detector callback
configuration = MediaGroup.PLAYER_SIGNALDETECTOR;
sigDetListener = new MySignalDetectorListener();

// Setup the options for receiveSignals
collectOptions = mscFactory.createParameters();

// Initialize
// Indicate that we want to do DTMF Async Continuous Detection
// note all timeouts are default to FOREEVER
// These parameters must be set in order to trigger the forever detection

collectOptions.put(SignalDetector.BUFFERING, Boolean.FALSE );
EventType[] arrayEnabledEvents = {SignalDetectorEvent.SIGNAL_DETECTED} ;
collectOptions.put(SignalDetector.ENABLED_EVENTS, arrayEnabledEvents);

receiveSignals(-1, null, null, collectOptions);
```

The stop() method can be used during "collect" or "prompt and collect". When used before the signal is detected, the receiveSignals() method will be terminated and the RECEIVE_SIGNALS_COMPLETED event will be sent to notify application. If the stop() is called after receiveSignals() has completed, it will result in an error event being sent to the application.

Note: The reason for the error is that when the receiveSignals() method is called (except for Async DTMF detection), it is executed for one time functionality (i.e., once the detection takes place it is terminated automatically).

Pattern Detection Sample Code

Below is an example extracted from Dialogic Verification Demo source, which is available as a project as part of JSR 309 distribution.

```
public static void collectPattern(MediaGroup mg)
{
        log.debug("Entering");
        Parameters mgParametersLocal = null;
        Parameters localParameters = mg.createParameters();
        mgParametersLocal = mg.getParameters(null);
        if (mgParametersLocal.get(SignalDetector.INTER_SIG_TIMEOUT) != null){
                log.info("SignalDetector.INTER_SIG_TIMEOUT = " +
Integer.parseInt(mgParametersLocal.get(SignalDetector.INTER_SIG_TIMEOUT).toString()));
                localParameters.put(SignalDetector.INTER_SIG_TIMEOUT,
Integer.parseInt(mgParametersLocal.get(SignalDetector.INTER_SIG_TIMEOUT).toString()));
        }
        else{
                log.info("SignalDetector.INTER_SIG_TIMEOUT = " +
Integer.parseInt(mgParametersLocal.get(SignalDetector.INTER_SIG_TIMEOUT).toString()));
        }
        if (mgParametersLocal.get(SignalDetector.INITIAL_TIMEOUT) != null){
                log.info("SignalDetector.INITIAL_TIMEOUT = " +
Integer.parseInt(mgParametersLocal.get(SignalDetector.INITIAL_TIMEOUT).toString()));
```

```
                localParameters.put(SignalDetector.INITIAL_TIMEOUT,
Integer.parseInt(mgParametersLocal.get(SignalDetector.INITIAL_TIMEOUT).toString()));
        }
        else{
                log.info("SignalDetector.INITIAL_TIMEOUT = " +
Integer.parseInt(mgParametersLocal.get(SignalDetector.INITIAL_TIMEOUT).toString()));
        }
        if (digitPattern != null){
                log.info("Detecting digit pattern: = " + digitPattern);
                localParameters.put(SignalDetector.PATTERN[1], digitPattern);
                Parameter[] detectDigitPattern = { SignalDetector.PATTERN[1] };
                try {
                        mg.getSignalDetector().receiveSignals(
                                        0,
                                        detectDigitPattern,
                                        null,
                                        localParameters);
                }
                catch (Exception e)     {
                        log.error("Exception: " + e.toString(), e);
                }
        }
        else{
                log.info("SignalDetector.PATTERN[1] = can not be 'null'");
        }
        log.debug("Exiting");
}
```

Supported Patterns and Examples

The following tables provide a list of supported patterns and examples:

- Supported patterns where "Number of Signals" is set to 1 or more
- Supported patterns where "Number of Signals" is set to 0 or less

The "Number of Signals" column defines the number of signals to be detected.

The "Match Pattern" column defines the DTMF pattern to look for. Note, the definitions of this parameter depend on "Number of Signals" used.

The "DTMF Entered" column shows the string of DTMF which were pressed during the test. Note, the digit highlighted in **bold** shows the DTMF which caused a completion of collection.

Supported patterns where "Number of Signals" is set to 1 or greater

"Number of Signals" defines the maximum required number of DTMF signals to look for. Note, the minimum value is always set to 1.

"Match Pattern" defines a digit which will terminate the DTMF signal detection. Note, the only valid values for "Match Pattern" are a single digit: 0-9, A, B, C, D, *, or #.

The DTMF defined in "Match Pattern" will be returned as part of matched string. Note, the DTMF used in Match Pattern can only be used as a DTMF that terminates detection.

Sample Number	Number of Signals	Match Pattern	DTMF Entered	Outcome
1	5	4	**4**56	No Match: 4 Match Pattern value was detected but minimum number of signals was not satisfied min=1 max=5 Same issue as hash not part of pattern
2	5	4	56**4**	Match 564
3	5	#	12**#**	Match 12#
4	3	#	258	No Match: 258 As "#" was expected but not present.
5	5	#	12345**6**	No Match: 123456 As "#" was expected but "6" was received instead.

Supported patterns where "Number of Signals" is set to 0 or less

"Number of Signals" needs to be set to "0".

"Match Pattern" defines a pattern string which will be used to detect DTMFs.

There are two supported types of match pattern strings:

- String which will define minimum (min) and maximum (max) number of digits to be looked for with optional termination key (rtk).

  Example: "min=X;max=Y" or "min=X;max=Y;rtk=Z"

  X - defines minimum number of digits to be collected
  Y - defines maximum number of digits to be collected
  Z - defines a single digit type to be used as a termination digit of expected DTMF pattern (optional)

  Note: This is similar to the table above except this option allows setting the minimum value for the pattern to be detected.

- String defining specific number and type of DTMFs to look for with an option of using a wild card "X". Supported characters in this string are: 0-9, A, B, C, D, *, or # as well as "X" which is used as a wild card representing any of the supported DTMF characters.

  Example: "1234", "123x4", "xxxx", etc.

  Note: The only valid values for "Match Pattern" are a single digit: 0-9, A, B, C, D, *, or # DTMF defined in "Match Pattern" will be returned as part of matched string.

Test Case #	Number of Signals	Match Pattern	DTMF Entered	Outcome
1	0	min=3;max=5	123	Match: 123 As soon as Inter Digit Timeout expires after digit "3" was pressed.
2	0	min=3;max=5; rtk=#	12345**#**	Match: 12345#
3	0	min=2;max=5; rtk=#	123456	No Match: 123456 Expected rtk "#" but received "6" instead.
4	0	min=1;max=5; rtk=#	12345**#**	Match: 12345#
5	0	123	12**3**	Match: 123
6	0	XXXX	123**4**	Match: 1234
7	0	min=2;max=5; rtk=3	**3**33	No Match: 3 Received rtk digit but min/max range was not satisfied.
8	0	12X33	1293**3**	Match: 12933 **Note:** DTMF "9" was matched by a wild card "x".

SignalDetectorEvent

What's not supported

Field Summary	
static EventType	FLUSH_BUFFER_COMPLETED FlushBuffer completion event.
static EventType	OVERFLOWED The SignalDetector signal buffer has overflowed.
static Qualifier	PROMPT_FAILURE Returned by SignalDetectorEvent.getQualifier() to indicate that the receiveSignals operation was aborted, because the prompt could not be played.

Methods inherited from javax.media.mscontrol.resource.ResourceEvent	
getRTCTrigger	Get the RTC Trigger that caused this transaction completion.

SpeechDetectorConstants

What's not supported

Field Summary	
static Parameter	BARGE_IN_ENABLED Controls whether the caller can start speaking before prompts have ended.
static Trigger	END_OF_SPEECH Trigger when end of speech has been detected (i.e., final timeout popped).
static Qualifier	END_OF_SPEECH_DETECTED SpeechDetector stopped because end of speech has been detected.
static Parameter	FINAL_TIMEOUT Controls the length of a period of silence after callers have spoken to conclude they finished.
static Parameter	INITIAL_TIMEOUT Controls how long the recognizer should wait after the end of the prompt for the caller to speak before sending a Recorder event (COMPLETED, INITIAL_TIMEOUT_EXPIRED, null, NO_ERROR).
static Trigger	INITIAL_TIMEOUT_EXPIRATION Trigger when no speech has been detected before the initial timeout popped.
static Qualifier	INITIAL_TIMEOUT_EXPIRED SpeechDetector stopped because no speech has been detected before the initial timeout popped.
static Action	PROMPT_DONE Advise the Speech Detector that the prompt is finished.
static Parameter	SENSITIVITY Sensitivity of the speech detector when looking for speech.
static Qualifier	SPEECH_DETECTED SpeechDetector detected speech.
static Trigger	START_OF_SPEECH Trigger when speech has been detected.

VideoRendererEvent

What's not supported

Methods inherited from javax.media.mscontrol.resource.ResourceEvent	
getRTCTrigger	Get the RTC Trigger that caused this transaction completion.

7. Release Issues

This section lists the issues that may affect the JSR 309 Connector.

Issues Table

The table below lists issues that affect the JSR 309 Connector. The following information is provided for each issue:

Issue Type

This classifies the type of release issue based on its effect on users and its disposition:

- Known – A minor issue. This category includes interoperability issues and compatibility issues. Known issues are still open but may or may not be fixed in the future.
- Known (permanent) – A known issue or limitation that is not intended to be fixed in the future.
- Resolved – An issue that was resolved (usually either fixed or documented) in this release.

Defect No.

A unique identification number that is used to track each issue reported.

Release No.

For defects that were resolved in a release, the release number is shown.

Product or Component

The product or component to which the problem relates; for example, an API.

Description

A summary description of the issue. For non-resolved issues, a workaround is included when available.

Issues Sorted by Type, JSR 309 Connector

Issue Type	Defect No.	Release No.	Product or Component	Description
Resolved	XMS-7573	3.4 SU1	JSR 309 Connector	When sip.connector.transport is set to TCP, the interaction between the JSR 309 Connector and PowerMedia XMS continues to be over UDP.
Resolved	XMS-7572	3.4 SU1	JSR 309 Connector	When using joinInitiate with both audio and video attributes, only audio stream is used.

Issue Type	Defect No.	Release No.	Product or Component	Description
Resolved	XMS-7556	3.4 SU1	JSR 309 Connector	Support for multiple NC bridge connections as well as support for native NC to NC joins.
Resolved	XMS-7112	3.4 SU1	JSR 309 Connector	A play issued while another play is already in progress causes 5 seconds timeout exception instead of failing right away.
Resolved	XMS-7044	3.4 SU1	JSR 309 Connector	Global Unique Session Identifier (GUSID) is not maintained for SIP response messages to the media server.
Resolved	XMS-4221	3.4 SU1	JSR 309 Connector	For redundant application server configuration, setOutboundInterface needs to support network interface selection.

8. Appendix A: DLGCSMIL Video Layout <dlgcsmil>

Acronym	Description
DLGCSMIL	Dialogic SMIL like makeup language to define the video layout in the conference
XML	Extended Markup Language
RGB	Red-Green-Blue color model
JPEG	Joint Photographic Experts Group image format
PNG	Portable Network Graphics file format

The DLGCSMIL Video Layout <dlgcsmil> has the following elements:

- <head>
- <layout>
- <region>
- <body>
- <par>
- <ref>
- <text>
- <img>
- <scroll>

```
<dlgcsmil …>
   <head>
      <layout…>
         <region …/>
         <region …/>
         …
      </layout>
   </head>
   <body>
      <par>
         <ref …/>
         <text …>
            <scroll …/>
         </text>
         <img …>
            <scroll …/>
         </img>
      </par>
   </body>
</dlgcsmil>
```

Elements

<head>

Parent: <dlgcsmil>

Child Elements: <layout>

Description

Contents layout info, only one layout element is allowed.

<layout>

Parent: <head>

Child: <region>

Description

A layout is specified using the <layout> element. It is used as a container to hold elements that describe all of the properties of a video mix. When the video mix in composed of multiple panes, the location and characteristics of the panes are defined by one or more <region> elements.

Attributes

Attributes	Description
size	This attribute specifies the resolution of the root window. Supported values are: QCIF, CIF, 4CIF, 16CIF, VGA, and 720p. The attribute is optional when creating the Conference. • Default value = CIF

<region>

Parent: <layout>

Child Elements: None.

Description

The <region> element is used to define video panes (or tiles) that are used to display participant video streams in a video conference. Regions are rendered on top of the root window. Up to 10 regions may be defined for each conference.

The location of the top left corner of a region is specified using the position attributes "top" and "left" and is defined relative to the top left corner of the root window. The size of a region is specified using the "relativesize" attribute and is defined relative to the size of the root window.

An example of a video layout with six regions is:

```
+----------+-----+
|           | 2  |
|        1 +-----+
|           | 3  |
+----+----+------+
| 6  | 5   |  4  |
+---+------+-----+
```

```
< layout size="CIF"/>
         <region id="1" left="0" top="0" relativesize="2/3"/>
         <region id="2" left="67" top="0" relativesize="1/3"/>
         <region id="3" left="67" top="33" relativesize="1/3">
         <region id="4" left="67" top="67" relativesize="1/3"/>
         <region id="5" left="33" top="67" relativesize="1/3"/>
         <region id="6" left="0" top="67" relativesize="1/3"/>
</layout>
```

Portions of regions that extend beyond the root window will be cropped.

For example, a layout specified as:

```
< layout size="CIF"/>
         <region id="foo" left="50%" top="50%" relativesize="2/3"/>
</layout>
```

Would appear similar to:

```
+---------- --+
| root        |
|background   |
|         +--- +--
|         |    |//
|         |foo |//
+--------+----+//
        |////////
```

The area of the root window covered by a region is a function of the region's position and its size. When areas of different regions overlap, they are layered in order of their "priority" attribute. The region with the highest value for the "priority" attribute is below all other regions and will be hidden by overlapping regions. The region with the lowest non-zero value for the "priority" attribute is on top of all other regions and will not be hidden by overlapping regions. The priority attribute may be assigned values between 0 and 1. Note that a value of "0" is currently not supported.

Regions that do not specify a priority will be assigned a priority by a media server when a conference is created. The first region within the <layout> element that does not specify a priority will be assigned a priority of one, the second a priority of two, etc. In this way, all regions that do not explicitly specify a priority will be underneath all regions that do specify a priority. As well, within those regions that do not specify a priority, they will be layered from top to bottom, in the order they appear within the <layout> element.

For example, if a layout was specified as follows:

```
<layout>
         <region id="a" ... priority=".3" .../>
         <region id="b" ... />
         <region id="c" ... priority=".2" ...>
         <region id="d" ... />
</layout>
```

Then the regions would be layered, from top to bottom: c,a,b,d.

Attributes

Attributes	Description
id	Mandatory: This attribute specifies a name that is used to refer to the region. For example, reference to a region is required when modifying the characteristics of a region and when specifying which region a video stream will be displayed in. Note that once a region is created for a conference, that region will continue to exist for the life of the conference. Therefore, only 10 regions with 10 unique ids can be used for regions of a conference. The region can be made invisible and it can be reused with different characteristics. But the region and its id will only be destroyed when the conference that it belongs to is destroyed or the video mix of the conference that it belongs to is destroyed.
left	Mandatory: This attribute specifies the position of the left edge of the region as a relative offset from the left edge of the root window. Values may be expressed either as a percent (%) of the horizontal dimension of the root window.
top	Mandatory: This attribute specifies the position of the top edge of the region as a relative offset from the top edge of the root window. Values may be expressed either as a percent (%)of the horizontal dimension of the root window.
left	Optional: This attribute specifies a priority level determining how regions are visible when they overlap with other regions. Regions with lower priority levels will be layered on top of regions with higher priority levels. Supported values are 0.1 to 0.9 (0.1, 0.2, …, 0.9) For regions with the same priority level, the last region created will be layered on top of previous regions created. Note that a value of "0" is currently not supported. Default value = 1.

<body>

Parent: <dlgcsmil>

Child Elements: <par>

Description

Specify the region's display and text/img overlay in the region.

<par>

Parent: <body>

Child Elements: <ref>, <text>, <img>

Description

A par container, it is a placeholder for now.

<ref>

Parent: <par>

Child Elements: None.

Description

It is a generic media object that displays in the region. In the JSR 309 Connector, it could be the following URIs:

- mscontrol:////Mixer/StreamGroup.__Any__ (Any Stream)

  Example:

  ```
  <ref region="1" src=" mscontrol:////Mixer/StreamGroup.__Any__"/>
  ```

- mscontrol:////Mixer/StreamGroup.__MostActive__ (Most Active Stream)

  Example:

  ```
  <ref region="1" src=" mscontrol:////Mixer/StreamGroup.__Any__"/>
  ```

- network connection object's URI

  Example:

  ```
  <ref region="1" src=" mscontrol://146.152.122.179/app-pko6kgxamxoi_9601_-
  40f42b601b40bc91.MS-458b3131fc991/com.vendor.dialogic.javax.media.mscontrol.
  networkconnection.DlgcXNetworkConnection.XNC-458b3137b7551"/>
  ```

The "Most Active Stream" can only be defined once in a lifetime of the conference, and it cannot be removed once it has been defined in a region.

Attributes

Attributes	Description
region	Mandatory: This attribute specifies the identifier of a video layout region that is to be used to display the video stream.
src	Mandatory: This attribute specifies the URI of a video stream source that is to be displayed in the region.

<text>

Parent: <par>

Child Elements: <scroll>

Description

The text overlay is placed in a region.

The "Most Active Stream" can only be defined once in a lifetime of the conference, and it cannot be removed once it has been defined in a region.

Attributes

region: (mandatory) This attribute specifies the identifier of a video layout region that is to be used to display the text overlay.

src: (mandatory) This attribute specifies the URI of a video stream source that is to be displayed in the region.

leftPosition: the position of the overlay from the left side of the reference window, defined as a % of the overall width of the reference window. Supported values, when expressed as a percent, range from -100 to 100. Default value = 0 if not previously specified.

topPosition: the position of the overlay from the top of the reference window, defined as a % of the overall height of the reference window. Supported values, when expressed as a percent, range from -100 to 100. Default value = 0 if not previously specified.

horizontalSize: the horizontal size of the layout expressed as a % of the reference window horizontal size. Supported values, when expressed as a percent, range from 0% to 100%. Default value = 0 if not previously specified.

verticalSize: the vertical size of the layout expressed as a % or fraction of the reference window vertical size. Supported values, when expressed as a percent, range from -100% to 100%. Supported values, when expressed as a percent, range from -100 to 100. Default value = 0 if not previously specified.

priority: a number between 0 and 1 that is used to define the precedence when rendering overlapping layouts. A value of "0" (zero) deletes the overlay. When areas of different layouts overlap, they are layered in order of their "priority" attribute. The layout with the highest value for the "priority" attribute is below all other layouts and may be hidden by overlapping layouts. The layout with the lowest non-zero value for the "priority" attribute is on top of all other layouts and will not be hidden by overlapping layouts. The priority attribute may be assigned values between 0 and 1. A value of zero disables the layout deleting it and freeing any resources associated with the layout. Note that layouts at the root level will always display on top of layouts specified at the region level, regardless of the priority setting.

borderWidth: Horizontal and vertical border width of the overlay defined as a % of the overall height of the layout window. Supported values, when expressed as a percent, range from 0% to 50%. This specifies the width sizes of the border inside the layout window. This attribute overrides hBorderWidth and vBorderWidth. It should not be specified when using hBorderWidth and vBorderWidth. Default is "0", no border.

hBorderWidth: Horizontal border width of the overlay defined as a % of the overall height of the layout window. This attribute may be used if there is a desire to have different settings for the horizontal and vertical borders. For example, a vertical border may be desired while a horizontal border may not be desired. Supported values, when expressed as a percent, range from 0% to 20%. This specifies the horizontal width size of the border inside the layout window. Default is "0", no border.

vBorderWidth: Vertical border width of the overlay defined as a % of the overall height of the layout window. This attribute may be used if there is a desire to have different settings for the horizontal and vertical borders. For example, a vertical border may be desired while a horizontal border may not be desired. Supported values, when expressed as a percent, range from 0% to 20%. This specifies the vertical width size of the border inside the layout window. Default is "0", no border.

borderColor: Color of the overlay border. For supported values, refer to the Supported Colors. The default value is "gray".

borderOpacity: This attribute defines the opacity of the border of the overlay region. It accepts a percentage value in the range 0-100% or a number in the range 0.0-1.0, with 100% or 1.0 meaning fully opaque. A borderOpacity=0% results in a fully transparent border. The default value of this attribute is 100 (fully opaque).

backgroundColor: Background color of the layout. For supported values, refer to the Supported Colors. The default value is "blue".

backgroundOpacity: This attribute defines the opacity of the background of the overlay region. It accepts a percentage value in the range 0-100% or a number in the range 0.0-1.0, with 100% or 1.0 meaning fully opaque. The default value of this attribute is 100%. A backgroundOpacity=0% results in a fully transparent background.

fontFamily: Name of the font family. Values are as follows: Arial, Courier New, Tacoma, Times New Roman, Verdana, or any font type that is supported on the host platform.

fontStyle: Font style: Values are: normal and italic. The default value is "normal".

fontWeight: Font weight. Values are: normal and bold. The default value is "normal".

fontEffects: Font effects. Values are: none and underlined. The default value is "none".

fontSize: Font size. Values are specified as a % of the vertical size of the layout region. Supported values, when expressed as a percent, range from 0.0% to 100.0%. The default value is 90 (90%).

fontColor: Color of text. For supported values, refer to the Supported Colors. The default value is white.

fontOpacity: This attribute defines the opacity of the font color to be applied to the font when this style is applied. It accepts a percentage value in the range 0-100% or a number in the range 0.0-1.0, with 100% or 1.0 meaning fully opaque. The default value of this attribute is 100%.

textBackgroundColor: Background color of the text. For supported values, refer to the Supported Colors. The default value is "blue".

textBackgroundOpacity: This attribute defines the opacity of the background of the text. It accepts a percentage value in the range 0-100% or a number in the range 0.0-1.0, with 100% or 1.0 meaning fully opaque. The default value of this attribute is 100%. A backgroundOpacity=0% results in a fully transparent background.

textAlignment: Alignment of text within the layout region. Values supported are: center, centerLeft, centerRight, topLeft, bottomLeft, topRight, bottomRight, topCenter, bottomCenter. Default value is center. For the case where content applied is static, textAlignment refers to the positioning of static text within the overlay window. For the value center, text is centered within the layout window. For the case where content applied is scrolled, textAlignment does not apply.

wrapOption: Wrap option. The default value is "nowrap". Values are: "wrap" (word wrap) and "nowrap". Wrap direction is by default, top to bottom when text-direction is either lr or rl, and is left to right when text-direction is either tb or bt.

duration: Specifies time duration for the content to be displayed. Values are specified in seconds and milliseconds. A value of "0" indicates that the content should be displayed indefinitely. Duration does not apply when content is scrolled. Default = "0".

src: Identifies the location of the text to be overlaid. The file and http schemes are supported. This attribute may be omitted if inline text is specified using the text attribute.

type: Used with the src attribute. Specifies the MIME type of the text to be overlaid. Values supported are: text/plain. Default value is text/plain.

encoding: The encoding type of the text. Values are: "UTF8", "ASCII", or "GB18030". Default value is "UTF8".

text: The inline text.

Notes

The fontFamily types are limited to those installed on the host system. The system uses two open source engines for text layout (Pango) and graphical rendering (Cairo). These engines are configured to use the same font libraries as used by the Host system when displaying text. When a font family (fontFamily) type is specified, the system will use the pre-installed libraries (i.e., Fontconfig/Freetype open source libraries) and the UTF-8 formatted input to determine the fontFamily style in which the text is rendered.

The system always requires text in UTF-8 format. For all inline specifications, the text must be in UTF-8 format. For text specifications in which the text is not inline, and the file containing the text is not encoded in UTF-8, the encoding attribute is required to determine if conversion is required by the system prior to rendering.

At least one of the src and text attributes has to be defined. If both are being defined, text has higher priority.

Parent: <par>

Child Elements: <scroll>

Description

The image overlay is placed in a region.

Attributes

region: (mandatory) This attribute specifies the identifier of a video layout region that is to be used to display the image overlay.

leftPosition: the position of the overlay from the left side of the reference window, defined as a % of the overall width of the reference window. Supported values, when expressed as a percent, range from -100 to 100. Default value = 0 if not previously specified.

topPosition: the position of the overlay from the top of the reference window, defined as a % of the overall height of the reference window. Supported values, when expressed as a percent, range from -100 to 100. Default value = 0 if not previously specified.

horizontalSize: the horizontal size of the layout expressed as a % of the reference window horizontal size. Supported values, when expressed as a percent, range from 0% to 100%. Default value = 0 if not previously specified.

verticalSize: the vertical size of the layout expressed as a % or fraction of the reference window vertical size. Supported values, when expressed as a percent, range from -100% to 100%. Supported values, when expressed as a percent, range from -100 to 100. Default value = 0 if not previously specified.

priority: a number between 0 and 1 that is used to define the precedence when rendering overlapping layouts. A value of "0" (zero) deletes the overlay. When areas of different layouts overlap, they are layered in order of their "priority" attribute. The layout with the highest value for the "priority" attribute is below all other layouts and may be hidden by overlapping layouts. The layout with the lowest non-zero value for the "priority" attribute is on top of all other layouts and will not be hidden by overlapping layouts. The priority attribute may be assigned values between 0 and 1. A value of zero disables the layout deleting it and freeing any resources associated with the layout. Note that layouts at the root level will always display on top of layouts specified at the region level, regardless of the priority setting.

borderWidth: Horizontal and vertical border width of the overlay defined as a % of the overall height of the layout window. Supported values, when expressed as a percent, range from 0% to 50%.This specifies the width sizes of the border inside the layout window. This attribute overrides hBorderWidth and vBorderWidth. It should not be specified when using hBorderWidth and vBorderWidth. Default is "0", no border.

hBorderWidth: Horizontal border width of the overlay defined as a % of the overall height of the layout window. This attribute may be used if there is a desire to have different settings for the horizontal and vertical borders. For example, a vertical border may be desired while a horizontal border may not be desired. Supported values, when expressed as a percent, range from 0% to 20%. This specifies the horizontal width size of the border inside the layout window. Default is "0", no border.

vBorderWidth: Vertical border width of the overlay defined as a % of the overall height of the layout window. This attribute may be used if there is a desire to have different settings for the horizontal and vertical borders. For example, a vertical border may be desired while a horizontal border may not be desired. Supported values, when expressed as a percent, range from 0% to 20%. This specifies the vertical width size of the border inside the layout window. Default is "0", no border.

borderColor: Color of the overlay border. For supported values, refer to the Supported Colors. The default value is "gray".

borderOpacity: This attribute defines the opacity of the border of the overlay region. It accepts a percentage value in the range 0-100% or a number in the range 0.0-1.0, with 100% or 1.0 meaning fully opaque. A borderOpacity=0% results in a fully transparent border. The default value of this attribute is 100 (fully opaque).

backgroundColor: Background color of the layout. For supported values, refer to the Supported Colors. The default value is "blue".

backgroundOpacity: This attribute defines the opacity of the background of the overlay region. It accepts a percentage value in the range 0-100% or a number in the range 0.0-1.0, with 100% or 1.0 meaning fully opaque. The default value of this attribute is 100%. A backgroundOpacity=0% results in a fully transparent background.

imgBackgroundColor: Background color of the image. For supported values, refer to the Supported Colors. The default value is "blue".

imgBackgroundOpacity: This attribute defines the opacity of the background of the image. It accepts a percentage value in the range 0-100% or a number in the range 0.0-1.0, with 100% or 1.0 meaning fully opaque. The default value of this attribute is 100%. A backgroundOpacity=0% results in a fully transparent background.

applyMode: Fill mode: Specifies how the image will be applied to the overlay window. Values are as follows:

- resizeToFit: The image will be resized to fit within the overlay window while maintaining the aspect ratio of the image.
- resizeToFill: The image will be resized in both the horizontal and/vertical dimensions if necessary.
- maintainSize: The image is not resized. If the overlay image is equal in size or smaller than the overlay window, the image will be displayed in its entirety. If the overlay image is larger than the overlay window, in either the horizontal or vertical dimension, the image will be cropped when displayed.
- Default is resizeToFit.

imgSize: Image size values are specified as a % of the vertical size of the layout region. Supported values, when expressed as a percent, range from 0.0% to 100.0%. The default value is 90 (90%).

imgAlignment: Alignment of the image within the layout region. This attribute does not apply when the applyMode is set to resizeToFill. Values supported are: center, centerLeft, centerRight, topLeft, bottomLeft, topRight, bottomRight, topCenter, bottomCenter. For the case where content applied is static, imgAlignment refers to the positioning of a static image within the overlay window. For the value center, the image is centered within the layout window. For the case where content applied is scrolled, imgAlignment does not apply. Default is center.

duration: Specifies a time duration for the content to be displayed. Values are specified in seconds and milliseconds. A value of "0" indicates that the content should be displayed indefinitely. Duration does not apply when content is scrolled. Default = "0".

src: (mandatory) Identifies the location of the image to be overlaid. The file and http schemes are supported.

type: Specifies the image MIME type of the image to be overlaid. Values supported are: image/png, image/jpeg.

Notes

Lower levels (media processing engine) require that the file extension reflect the correct format of image type (i.e., jpg or png).

The system supports both local and remote file name specifications, with the exception of overlay template files that must be local.

The duration does not apply, if the image is being scrolled.

<scroll>

Parent: <text>, <img>

Child Elements: None.

Description

The <scroll> element is used to describe how text is rendered in an overlay. It describes attributes with respect to how scrolling is performed and the look and feel of the scrolling output (i.e., speed, direction, alignment).

Attributes

mode: The scrolling mode. Values are: scrollOnce (scroll content one time), scrollContinuous (scroll continuously). Default is scrollOnce.

speed: Speed of content scrolling in % per second relative to the text/img layout region. Values supported are from 1 to 100 in increments of 1%. The default value is 25 (25%).

direction: Direction of content to be scrolled. Values supported are: lr (left to right), rl (right to left), tb (top to bottom), bt (bottom to top). The default value is rl (right to left).

padding: Specifies minimum padding to be added before the scrolled text/img . Values are in % relative to the text layout region in the dimension (width, height) of scrolling and supported values are from 1 to 100 in increments of 1%. The default value is set to a value of 5 (5%).

Supported Colors

Attributes that specify color are as follows:

- For <overlay>, attributes backgroundColor and borderColor.
- For <textStyle>, attributes fontColor and backgroundColor.
- For <imgStyle>, attributes backgroundColor.

Supported values for these attributes are as follows:

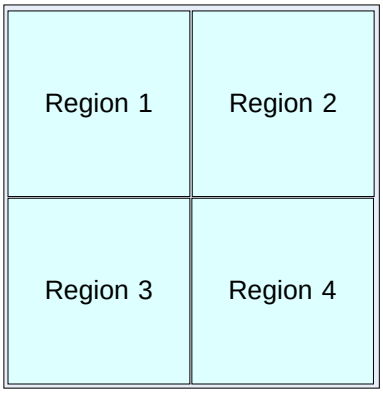
aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque,
black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse,
chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue,
darkcyan, darkgoldenrod, darkgray, darkgrey, darkgreen, darkkhaki, darkmagenta, darkoliv
egreen, darkorange, darkOrchid, darkred, darksalmon,
darkseagreen, darkslateblue, darkslategray, darkslategrey, darkturquoise, darkviolet,
deeppink, deepskyblue, dimgray, dimgrey, dodgerblue, firebrick, floralwhite,
forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod,
gray, grey, green, greenyellow, honeydew, hotpink, indianred, indigo,
ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral,
lightcyan, lightgoldenrodyellow, lightgray, lightgrey, lightgreen, lightpink, lightsalmon,
lightseagreen, lightskyblue, lightslategray, lightslategrey, lightsteelblue, lightyellow, lime,
limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid,
mediumpurple, mediumseagreen, mediumslateblue, mediumspringgreen, mediumturquoise,
mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy,
oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen,
paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple,
red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen,
seashell, sienna, silver, skyblue, slateblue, slategray, slategrey, snow, springgreen,
steelblue, tan, teal, thistle, tomato, turquoise, violet, wheat, white, whitesmoke, yellow, and
yellowgreen

Also supported are colors defined in compliance with W3C CSS2 recommendation section 4.3.6 subclause. The section describes a list of color keywords that can be specified in the RGB color space. For example, the rgb(255,255,255) format. Refer to the specifications at http://www.w3.org/TR/CSS2/syndata.html#color-units for more details.

Color specification using the HTML color code hexadecimal triplets format is also supported (i.e., #FFFFFF). This format represents the colors red, green and blue (#RRGGBB) and can be used with any of the color attributes.

DLGCSMIL Script Examples for Text and Image Overlays Applied to a Conference

All examples in this section apply overlays to a video conference using a 4 party layout and a VGA root size, as shown in the following figure, and the <head> element is used in script that follows to create each region for the participants.



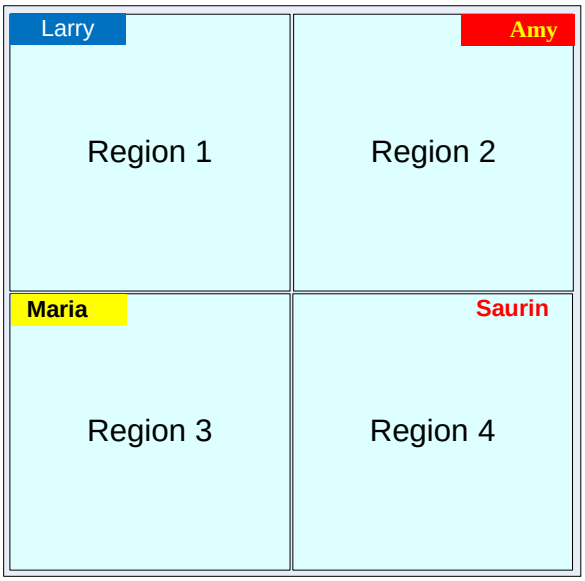
Video Layout 4.1

```
<?xml version="1.0" encoding="UTF-8"?>
<dlgcsmil version="1.0">
<head">
<layout size="VGA"/>>
<region id="1" left="0" top="0" relativesize="1/2"/>
<region id="2" left="50" top="0" relativesize="1/2"/>
<region id="3" left="0" top="50" relativesize="1/2"/>
<region id="4" left="50" top="50" relativesize="1/2"/>
</layout>
</head>
…
</dlgcsmil>
```

Example Adding Static Text Overlays to Regions of a Conference

For this example, a persistent text overlay is added to each region of the conference to display the name of the party in that region. All 4 overlays are added via the same dlgcsmil script.

Figure 1



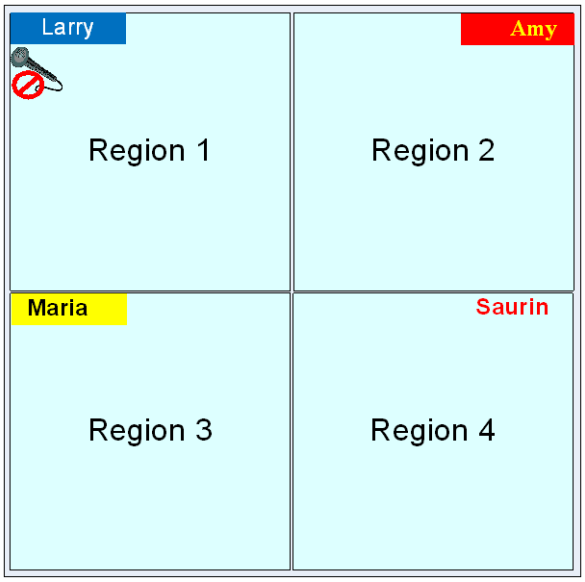
The below dialog script defines the conference layout as illustrated in Figure 1. Each region is defined in addition with an overlay name (i.e., the conferee name) and its display characteristics. Each name overlay has a different color scheme (text color, background) and is located in the upper right or left corner of the region.

```
<?xml version="1.0" encoding="UTF-8"?>
<dlgcsmil version="1.0">
        <head>
                <layout size="VGA">
                        <region id="1" left="0" top="0" relativesize="1/2"/>
                        <region id="2" left="50" top="0" relativesize="1/2"/>
                        <region id="3" left="0" top="50" relativesize="1/2"/>
                        <region id="4" left="50" top="50" relativesize="1/2"/>
                </layout>
        </head>
        <body>
                <par>
                        <ref region="1" src="mscontrol:////Mixer/StreamGroup.__MostActive__" />
                        <ref region="2" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <ref region="3" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <ref region="4" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <text region="1" leftPosition="0" topPosition="0" horizontalSize="40"
verticalSize="10"
                                                backgroundOpacity="0" priority="0.1"
fontFamily="Arial" textBackgroundColor="blue"
                                                text="Larry"/>
                        <text region="2"  leftPosition="60" topPosition="0" horizontalSize="40"
verticalSize="10"
  borderColor="red" backgroundOpacity="0" priority="0.1"
 fontFamily="TimesNewRoman" fontWeight="bold" fontColor="yellow"
 textBackgroundColor="red" textAlignment="centerRight" text="Amy"
 duration />>
                        <text region="3"  leftPosition="0" topPosition="0" horizontalSize="40"
verticalSize="10"
  backgroundOpacity="0" priority="0.1" fontFamily="Arial"  fontWeight="bold"
  fontColor="black" textBackgroundColor="yellow" textAlignment="centerLeft"
  text="Maria"/>
                        <text region="4"  leftPosition="60" topPosition="0" horizontalSize="40"
verticalSize="10"
                                        backgroundOpacity="0" priority="0.1" fontFamily="Arial"
fontWeight="bold"
                                                fontColor="red" textBackgroundOpacity="0"
text="Saurin"/>
                </par>
        </body>
</dlgcsmil>
```

Example Adding an Additional Static Image Overlay to a Region of a Conference

For this example, a persistent image overlay is added to region 1 to indicate that party in that region is currently muted. The resulting displayed video for the conference will be as illustrated in Figure 2 below.

Figure 2




```
<dlgcsmil version="1.0">
        <head>
                <layout size="VGA">
                        <region id="1" left="0" top="0" relativesize="1/2"/>
                        <region id="2" left="50" top="0" relativesize="1/2"/>
                        <region id="3" left="0" top="50" relativesize="1/2"/>
                        <region id="4" left="50" top="50" relativesize="1/2"/>
                </layout>
        </head>
        <body>
                <par>
                        <ref region="1" src="mscontrol:////Mixer/StreamGroup.__MostActive__" />
                        <ref region="2" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <ref region="3" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <ref region="4" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <text region="1" leftPosition="0" topPosition="0" horizontalSize="40"
verticalSize="10"
                                                backgroundOpacity="0" priority="0.1"
fontFamily="Arial" textBackgroundColor="blue"
                                                text="Larry"/>
                        <text region="2"  leftPosition="60" topPosition="0" horizontalSize="40"
verticalSize="10"
  borderColor="blue" backgroundOpacity="0" priority="0.1"
 fontFamily="TimesNewRoman" fontWeight="bold" fontColor="yellow"
```

```
 textBackgroundColor="red" textAlignment="centerRight" text="Amy"
 duration />>
                        <text region="3"  leftPosition="0" topPosition="0" horizontalSize="40"
verticalSize="10"
  backgroundOpacity="0" priority="0.1" fontFamily="Arial"  fontWeight="bold"
  fontColor="black" textBackgroundColor="yellow" textAlignment="centerLeft"
  text="Maria"/>
                        <text region="4"  leftPosition="60" topPosition="0" horizontalSize="40"
verticalSize="10"
                                backgroundOpacity="0" priority="0.1" fontFamily="Arial"
fontWeight="bold"
                                                fontColor="red" textBackgroundOpacity="0"
text="Saurin"/>
                        <img region="1" leftPosition="0" topPosition="10" horizontalSize="20"
verticalSize="20"
                                                 backgroundOpacity="0" priority="0.1"
type="image/png"
  src="file:////var/lib/xms/imagemaker/images/diastar-logo.png"/>
                </par>
        </body>
</dlgcsmil>
```

Example Deleting an Overlay Applied to a Region

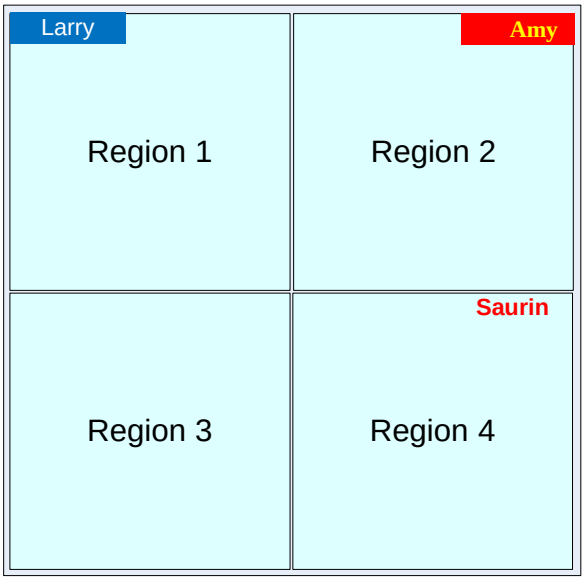
For this example, a persistent image overlay is removed from region 1 to and remove the region 3 text.

The resulting displayed video for the conference will be as illustrated in Figure 3 below.

```
<?xml version="1.0" encoding="UTF-8"?>
<dlgcsmil version="1.0">
        <head>
                <layout size="VGA">
                        <region id="1" left="0" top="0" relativesize="1/2"/>
                        <region id="2" left="50" top="0" relativesize="1/2"/>
                        <region id="3" left="0" top="50" relativesize="1/2"/>
                        <region id="4" left="50" top="50" relativesize="1/2"/>
                </layout>
        </head>
        <body>
                <par>
                        <ref region="1" src="mscontrol:////Mixer/StreamGroup.__MostActive__" />
                        <ref region="2" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <ref region="3" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <ref region="4" src="mscontrol:////Mixer/StreamGroup.__Any__" />
                        <text region="1" leftPosition="0" topPosition="0" horizontalSize="40"
verticalSize="10"
                                                backgroundOpacity="0" priority="0.1"
fontFamily="Arial" textBackgroundColor="blue"
                                                text="Larry"/>
                        <text region="2"  leftPosition="60" topPosition="0" horizontalSize="40"
verticalSize="10"
```

```
  borderColor="blue" backgroundOpacity="0" priority="0.1"
 fontFamily="TimesNewRoman" fontWeight="bold" fontColor="yellow"
 textBackgroundColor="red" textAlignment="centerRight" text="Amy"
 duration />>
                        <text region="4"  leftPosition="60" topPosition="0" horizontalSize="40"
verticalSize="10"
                                    backgroundOpacity="0" priority="0.1" fontFamily="Arial"
fontWeight="bold"
                                                fontColor="red" textBackgroundOpacity="0"
text="Saurin"/>
                </par>
        </body>
</dlgcsmil>
```

Figure 3



9. Appendix B – TCK Conformance Details

This section enumerates all the specific TCK test results; namely, which tests pass and why some tests fails.

TCK is defined as Technology Compatibility Kit, which is a suite of tests that normally checks a particular implementation. In this case, it is the JSR 309 API. This is defined by the Java Community Process group whose job is to define new Java specification such as the JSR 309 Media Server Control.

Dialogic passes over 80% of all TCK 309 conformance tests. In addition, Oracle has added three other conformance application tests that must be passed by the JSR 309 Connector in order to be conformant accepted. These three other conformance applications are:

1. Play Announcement Test
2. Play Asynchronous Announcement Test
3. Comprehensive Audio Conference with Record/DTMF Test

TCK Conformance Test Result Table

TCK Test Name	Result	Failure Explanation
test_2_1_1_1_Play	Success	
test_2_1_1_2_MaxDuration	Success	
test_2_1_1_3_StartOffset	Success	
test_2_1_1_4_RepeatCount	Success	
test_2_1_1_5_Interval	Success	
test_2_1_1_6_Stop	Success	
test_2_1_1_7_PlayURIArray	Success	
test_2_1_1_8_STOP_IF_BUSY	Success	
test_2_1_2_1_Record	Success	
test_2_1_2_2_MaxDuration	Success	
test_2_1_2_3_Append	**Fail**	**To be resolved in next release**
test_2_1_2_4_StartBeep	Success	
test_2_1_2_5_SilenceTerminationOn	Success	
test_2_1_2_6_FinalTimeout	Success	
test_2_1_3_3_Receive123	Success	
test_2_1_3_4_Receive123And456	Success	
test_2_1_3_5_Receive45And9And123	Success	
test_2_1_3_1_ReceiveOneSignal	Success	
test_2_1_3_2_ReceiveFiveSignals	Success	
test_2_1_3_10_ReceivePoundOr4Digits	Success	
test_2_1_3_11_Receive4DigitsOrDTMF3	Success	
test_2_1_12_3_BufferBeforeReceive	Success	

TCK Test Name	Result	Failure Explanation
test_2_1_12_4_ReceiveBufferedOneAtATime	Success	
test_2_1_12_5_flushBuffer	Success	
test_2_1_12_6_BufferAfterFlush	Success	
test_2_1_12_7_BufferParticipantDTMF	**Fail**	**To be resolved in next release**
test_2_1_12_8_BufferParticipantEarlyDTMF	Success	
test_2_1_12_9_FlushParticipantEarlyDTMF	Success	
test_2_1_12_2_BufferBetweenReceive	Success	
test_2_1_12_1_Buffer	Success	
test_2_1_4_1_MultiNCs	Success	
test_2_1_4_2_NCLoopback	Success	
test_2_1_4_3_PlayRecord	Success	
test_2_1_11_6_MediaGroupContainer	Success	
test_2_1_11_7_MediaMixerContainer	Success	
test_2_1_11_8_MediaMixerContainer	Success	
test_2_1_11_9_MediaMixerContainer	Success	
test_2_1_11_10_CreateVxmlDialog	Success	
test_2_1_11_11_SetGetAttribute	Success	
test_2_1_11_12_RemoveAttribute	Success	
test_2_1_11_13_GetAttributeNames	Success	
test_2_1_11_1_NetworkConnectionContainer	Success	
test_2_1_11_2_NetworkConnectionContainer	Success	
test_2_1_11_3_NetworkConnectionContainer	Success	
test_2_1_11_4_MediaGroupContainer	Success	
test_2_1_11_5_MediaGroupContainer	Success	
test_2_1_5_1_CreateMixer	Success	
test_2_1_5_2_JoinNetworkConnectionToMixer	Success	
test_2_1_5_3_UnjoinNetworkConnection	Success	
test_2_1_5_4_DTMFThroughMixer	Success	
test_2_1_8_2_VolumeControl	Success	
test_2_1_8_1_DTMFClamp	Success	
test_2_1_6_1_BasicBridge	Success	
test_2_1_6_2_ThreeWaysBridge	Success	
test_2_1_6_3_FiveWaysSplitter	Success	
test_2_1_6_4_UnjoinedBridge	Success	
test_2_1_6_8_BasicBridgeRecvOK	Success	

TCK Test Name	Result	Failure Explanation
test_2_1_6_9_BasicBridgeRecvNOK	Success	
test_2_1_6_10_BasicBridgeRejoinRecvNOK	Success	
test_2_1_6_11_BasicBridgeUnMute	Success	
test_2_1_13_2_MGJoinMixerTwoParticipants	Success	
test_2_1_13_1_MGJoinMixer	Success	
test_2_1_14_1_EnterAndLeaveConference	Success	
test_2_1_15_1_CatchDTMFOnParticipant	Success	
test_2_1_15_2_MuteAndDTMF	Success	
test_2_1_20_PerSessionId	Success	
test_2_1_18_8_RequireTelephoneEventAmr	Success	
test_2_1_18_9_RequireAbsentCodec	Success	
test_2_1_18_10_PreferPcmu	Success	
test_2_1_18_11_PreferAmr	Success	
test_2_1_18_12_PreferPcmuPcmaTelephoneEvent	Success	
test_2_1_18_13_CapablePcmu	Success	
test_2_1_18_14_CapablePcmuPcmaTelephoneEvent	Success	
test_2_1_18_15_CapableG726TelephoneEvent	Success	
test_2_1_18_16_CapableG726G722	Success	
test_2_1_18_17_CapablePcmuPcmaTeAmrPreferAmr	Success	
test_2_1_18_18_ComplexPolicy	Success	
test_2_1_18_19_AudioCapability	Success	
test_2_1_18_20_AudioVideoCapability	Success	
test_2_1_18_21_VideoCapability	Success	
test_2_1_18_1_ExcludeTelephoneEvent	Success	
test_2_1_18_2_ExcludeOneCodec	Success	
test_2_1_18_3_ExcludeMostCodecs	Success	
test_2_1_18_4_ExcludeAllCodecs	Success	
test_2_1_18_5_ExcludeCodecsMinimalOffer	Success	
test_2_1_18_6_RequireTelephoneEvent	Success	
test_2_1_18_7_RequirePcmuTelephoneEvent	Success	
test_2_1_10_2_Prompt_BargeIn	Success	
test_2_1_10_3_Cancel	Success	
test_2_1_10_4_CancelWithPrompt	Success	
test_2_1_10_5_Cleardigits	Success	
test_2_1_10_1_PROMPT	Success	

TCK Test Name	Result	Failure Explanation
test_2_1_9_1_PROMPT	Success	
test_2_1_9_2_Prompt_BargeIn	Success	
test_2_1_9_3_CANCEL	Success	
test_2_1_9_4_NoCancelWrongDTMF	Success	
test_2_1_9_5_CancelWithPrompt	Success	
test_2_1_9_6_SingleReturnKey	Success	
test_2_1_9_7_MultipleReturnKeys	Success	
test_2_1_9_8_NonMatchingReturnKey	Success	
test_2_1_9_9_NoReturnKey	Success	
test_2_1_9_10_AnyReturnKey	Success	
test_2_1_9_11_Cleardigits	Success	
test_2_1_7_2_PLAYEREvents	Success	
test_2_1_7_3_PLAYERIrrelevant	Success	
test_2_1_7_4_SIGNALDETECTOR	Success	
test_2_1_7_5_PLAYER_SIGNALDETECTOR	Success	
test_2_1_7_2_PLAYER_RECORDER_SIGNALDETECTOR	Success	
test_2_1_7_1_PLAYERParams	Success	
test_2_1_19_1_CustomizedClone	Success	
test_2_1_16_1_InitJoinNCMG	Success	
test_2_1_16_2_InitJoinNCMixer	Success	
test_2_1_17_1_ConfirmMixer	Success	
test_2_1_19_1_VxmlDialog	Success	
test_2_2_3_2_Run_AnyCode_In_Listener	Success	
test_2_2_3_1_remove_Listener	Success	
test_2_2_10_2_InvalidInitialCharacter	Success	
test_2_2_10_3_DuplicateIDs	Success	
test_2_2_10_1_LongId	Success	
test_2_2_4_2_UnjoinMGRecord	Success	
test_2_2_4_3_UnjoinMGSignalDetector	Success	
test_2_2_4_1_UnjoinMGPlay	Success	
test_2_2_6_1_CreateMediaObjectWithNullConfig	Success	
test_2_2_6_2_CreateMixerAdapterWithNullParams	Success	
test_2_2_1_3_Player_Invalid_StartOffset	**Fail**	**To be resolved in next release**
test_2_2_1_4_Invalid_RepeatCount	Success	
test_2_2_1_5_Invalid_Interval	Success	

TCK Test Name	Result	Failure Explanation
test_2_2_1_6_Player_State	Success	
test_2_2_1_1_Play_BadURI	Success	
test_2_2_7_1_InvalidRTC	Success	
test_2_2_7_2_InvalidTriggerRTC	Success	
test_2_2_8_1_ExcludedInRequired	Success	
test_2_2_8_2_ExcludedInCapabilities	Success	
test_2_2_8_3_ExcludedInPreferences	Success	
test_2_2_8_4_BogusCodecNamesExcluded	Success	
test_2_2_8_5_BogusCodecNamesCapable	Success	
test_2_2_8_6_BogusCodecNamesPreferred	Success	
test_2_2_9_1_UnchangedRecordRTC	Success	
test_2_2_9_2_UnchangedSigDetRTC	Success	