Independent Submission
Request for Comments: 5707
Category: Informational
ISSN: 2070-1721

A. Saleem
Y. Xin
RadiSys
G. Sharratt
Consultant

February 2010

Media Server Markup Language (MSML)

Abstract

The Media Server Markup Language (MSML) is used to control and invoke many different types of services on IP media servers. The MSML control interface was initially driven by RadiSys with subsequent significant contributions from Intel, Dialogic, and others in the industry. Clients can use it to define how multimedia sessions interact on a media server and to apply services to individuals or groups of users. MSML can be used, for example, to control media server conferencing features such as video layout and audio mixing, create sidebar conferences or personal mixes, and set the properties of media streams. As well, clients can use MSML to define media processing dialogs, which may be used as parts of application interactions with users or conferences. Transformation of media streams to and from users or conferences as well as interactive voice response (IVR) dialogs are examples of such interactions, which are specified using MSML. MSML clients may also invoke dialogs with individual users or with groups of conference participants using VoiceXML.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5707.

IESG Note

This RFC is not a candidate for any level of Internet Standard. The IETF disclaims any knowledge of the fitness of this RFC for any purpose and in particular notes that the decision to publish is not based on IETF review for such things as security, congestion control, or inappropriate interaction with deployed protocols. The RFC Editor has chosen to publish this document at its discretion. Readers of this document should exercise caution in evaluating its value for implementation and deployment. See RFC 3932 for more information.

Table of Contents

1. Introduction

Media servers contain dynamic pools of media resources. Control agents and other users of media servers (called media server clients) can define and create many different services based on how they configure and use those resources. Often, that configuration and the ways in which those resources interact will be changed dynamically over the course of a call, to reflect changes in the way that an application interacts with a user.

For example, a call may undergo an initial IVR dialog before being placed into a conference. Calls may be moved from a main conference to a sidebar conference and then back again. Individual calls may be directly bridged to create small n-way calls or simple sidebars. None of these change the SIP [n1] dialog or RTP [i3] session. Yet these do affect the media flow and processing internal to the media server.

The Media Server Markup Language (MSML) is an XML [n2] language used to control the flow of media streams and services applied to media streams within a media server. It is used to invoke many different types of services on individual sessions, groups of sessions, and conferences. MSML allows the creation of conferences, bridging different sessions together, and bridging sessions into conferences.

MSML may also be used to create user interaction dialogs and allows the application of media transforms to media streams. Media interaction dialogs created using MSML allow construction of IVR dialog sessions to individual users as well as to groups of users participating in a conference. Dialogs may also be specified using other languages, VoiceXML [n5], which support complete single-party application logic to be executed on the media server.

MSML is a transport independent language, such that it does not rely on underlying transport mechanisms and language semantics are independent of transport. However, SIP is a typical and commonly used transport mechanism for MSML, invoked using the SIP URI scheme. This specification defines using MSML dialogs using SIP as the transport mechanism.

A network connection may be established with the media server using SIP. Media received and transmitted on that connection will flow through different media resources on the media server depending on the requested service. Basic Network Media Services with SIP [n7] defines conventions for associating a basic service with a SIP Request-URI. MSML allows services to be dynamically applied and changed by a control agent during the lifetime of the SIP dialog.

MSML has been designed to address the control and manipulation of media processing operations (e.g., announcement, IVR, play and record, automatic speech recognition (ASR), text to speech (TTS), fax, video), as well as control and relationships of media streams (e.g., simple and advanced conferencing). It provides a general-purpose media server control architecture. MSML can additionally be used to invoke other more complex IVR languages such as VoiceXML.

The MSML control interface has been widely deployed in the industry, with numerous client-side and server-side implementations, since 2003. The in-service commercial deployments cover a wide variety of applications including, but not limited to, IP multimedia conferencing, network voice services, IVR, IVVR (interactive voice and video response), and voice/video mail.

2. Glossary

Media Server: a general-purpose platform for executing real-time media processing tasks. This is a logical function that maps either to a single physical device or to a portion of a physical device.

Media Server Client: an application that originates MSML requests to a media server and also referred to as a control agent in this specification.

Network Connection: a participant that represents the termination on a media server of one or more RTP [i3] sessions (for example, audio and video) associated with a call. Network connections are established and removed using a session establishment protocol such as SIP. An instance of a network connection is independent of MSML processing instructions applied to it.

Dialog: an automated IVR participant. Examples of dialogs may be announcement players, IVR interfaces, or voice recorders. Dialogs may be defined in MSML or using VoiceXML [n5].

Conference: an intermediary function that provides multimedia mixing and other advanced conferencing services. This specification currently considers conferences with audio and/or video media types, but is extensible to other media types.

Identifier: a name that is used to refer to a specific instance of an object on the media server, such as a conference or a dialog. Identifiers are composed of one or more terms where each term identifies an object class and instance.

Object: the generic term for a media server entity that terminates, originates, or processes media. This specification defines four classes of objects and specifies mechanisms to create them, join them together, and destroy them.

Participant Object: an object in a media server that sources original media in a call and/or receives and terminates media in a call.

Intermediary Object: an object in a media server that acts on media within a call for the benefit of the participants.

Independent Object: an object that can exist on a media server independent of other objects.

Operator: an intermediary transformer that modifies or transforms a media stream. Examples of operators may be audio gain controls, video scaling, or voice masking. MSML defines operators as media transform objects, which transform media using operations such as gain control, when applied to media streams.

Media Stream: a single media flow between two objects. A media stream has a media type and may be unidirectional or bidirectional.

3. MSML SIP Usage

SIP is used to create and modify media sessions with a media server according to the procedures defined in RFC 3261 [n1]. Often, SIP third party call control [i4] will be used to create sessions to a media server on behalf of end users. MSML is used to define and change the service that a user connected to a media server will receive. MSML clients are application servers, soft-switches, or other forms of control agents, and SHOULD have an authorized security relationship with the media server. MSML itself does not define authorization mechanisms.

MSML transactions are originated based upon events that occur in the application domain. These events may be independent from any media or user interaction. For example, an application may wish to play an announcement to a conference warning that its scheduled completion time is approaching. Applications themselves are structured in many different ways. Their structure and requirements contribute to their selection of protocols and languages. To accommodate differing application needs, MSML has been designed to be neutral to other languages and independent of the transport used to carry it.

MSML is purposely designed to be transport independent. In this release of the specification, SIP INFO [i5] and SIP Control Framework [i11] have been chosen for transport mechanisms for MSML, as described in the following sections.

3.1. SIP INFO

SIP INVITE and INFO [i5] requests and responses MAY be used to carry MSML. INFO requests allow asynchronous mid-call messages within SIP with few additional semantics. In addition, there are existing widely deployed implementations of that method, it aids in initial developments that are closely coupled with SIP session establishment, and it allows MSML to be directly associated with user dialogs when third party call control is used.

Although INFO is sometimes considered not to be a suitable general-purpose transport mechanism for messages within SIP, there have been proposals to make it more acceptable. MSML may evolve to include other SIP usage and/or to work with other protocols or as a stand-alone protocol established through SIP, in future releases of this document.

MSML supports several models for client interaction. When clients use 3PCC to establish media sessions on behalf of end users, clients will have a SIP dialog for each media session. MSML MAY be sent on these dialogs. However the targets of MSML actions are not inferred from the session associated with the SIP dialog. The targets of MSML actions are always explicitly specified using identifiers as previously defined.

An application, after interacting with a user, may want to affect multiple objects within a media server. For example, tones or messages are often played to a conference when connections are added or removed. A separate message may also be played to a participant as they are joined, or to moderators. Explicit identifiers, that is, not inferred from a transport mechanism, allow these multiple actions to be easily grouped into a single transaction sent on any SIP dialog.

MSML also supports a model of dedicated control associations. This supports decoupled application architectures where a client can control media server services without also establishing all of the media sessions itself. Control associations are created using SIP, but they do not have any associated media session. Although initially INFO messages will be sent on this SIP dialog, just as with dialogs associated with media sessions, it is possible that in the future, the SIP dialog will be used to establish a separate control session (defined in SDP [n9]) that does not use SIP as the transport for MSML messages.

A media server using MSML also sends asynchronous events to a client using MSML scripts in SIP INFO. Events are sent based on previous MSML requests and are sent within the SIP dialog on which the MSML request that caused the event to be generated was received. If this dialog no longer exists when the event is generated, the event is discarded.

Events may be generated during the execution of a dialog created by a <dialogstart> element. For example, dialogs can send events based on user input. VoiceXML dialogs, on the other hand, generally interact with other servers outside of MSML using HTTP.

An event is also generated when the execution of a dialog terminates, because of either completion or failure. The exact information returned is dependent on the dialog language, the capabilities of the dialog execution environment, and what was requested by the dialog. Both MSML and VoiceXML [n5] allow information to be returned when they exit. These events may be sent in a SIP INFO or a SIP BYE. SIP BYE is used when the dialog itself specifies that the connection should be disconnected, for example, through the use of the <disconnect> element.

Conferences may also generate events based upon their configuration. An example of this is the notification of the set of active speakers.

3.2. SIP Control Framework

The SIP Control Framework [i11] MAY be used as a transport mechanism for MSML.

The Control Framework provides a generic approach for establishment and reporting capabilities of remotely initiated commands. The framework utilizes many functions provided by the Session Initiation Protocol (SIP) [n1] for the rendezvous and establishment of a reliable channel for control interactions. Compared to SIP INFO, the

SIP Control Framework is a more general-purpose transport mechanism and one that is not constrained by limitations of the SIP INFO mechanism.

The Control Framework also introduces the concept of a Control Package, which is an explicit usage of the Control Framework for a particular interaction set. This specification has already specified a list of packages for MSML to control the media server in many aspects, including basic dialog, advanced conferencing, advanced dialog, and audit service. Each of these packages has a unique Control Package name assigned in order for MSML to be used with the Control Framework.

This section fulfills the mandatory requirement for information that MUST be specified during the definition of a Control Framework Package, as detailed in SIP Control Framework [i11].

3.2.1. Control Framework Package Names

The Control Framework [i11] requires a Control Package definition to specify and register a unique name.

MSML specification defines Control Package names using a hierarchical scheme to indicate the inherited relationship across packages. For example, package "msml-x" is derived from package "msml", and package "msml-x-y" is derived from package "msml-x".

The following is a list of Control Package names reserved by the MSML specification.

"msml": this Control Package supports MSML Core Package as specified in section 7.

"msml-conf": this Control Package supports MSML Conference Core Package as specified in section 8.

"msml-dialog": this Control Package supports MSML Dialog Core Package as specified in section 9.6.

"msml-dialog-base": this Control Package supports MSML Dialog Base Package as specified in section 9.7.

"msml-dialog-group": this Control Package supports MSML Dialog Group Package as specified in section 9.8.

"msml-dialog-transform": this Control Package supports MSML Dialog Transform Package as specified in section 9.9.

"msml-dialog-speech": this Control Package supports MSML Dialog Speech Package as specified in section 9.10.

"msml-dialog-fax-detect": this Control Package supports MSML Dialog Fax Detection Package as specified in section 9.11.

"msml-dialog-fax-sendrecv": this Control Package supports MSML Dialog Fax Send/Receive Package as specified in section 9.12.

"msml-audit": this Control Package supports MSML Audit Core Package as specified in section 10.1.

"msml-audit-conf": this Control Package supports MSML Audit Conference Package as specified in section 10.2.

"msml-audit-conn": this Control Package supports MSML Audit Connection Package as specified in section 10.3.

"msml-audit-dialog": this Control Package supports MSML Audit Dialog Package as specified in section 10.4.

"msml-audit-stream": this Control Package supports MSML Audit Stream Package as specified in section 10.5.

An application server using the Control Framework as transport for MSML MUST use one or multiple package names, depending on the service required from the media server. The package name(s) are identified in the "Control-Packages" SIP header that is present in the SIP INVITE dialog request that creates the control channel, as specified in [i11]. The "Control-Packages" value MAY be re-negotiated via the SIP re-INVITE mechanism.

3.2.2. Control Framework Messages

The usage of CONTROL, response, and REPORT messages, as defined in [i11], by each Control Package defined in MSML is different and described separately in the following sections.

MSML Core Package "msml"

The application server may send a CONTROL message with a body of MSML request using the following elements to the MS:

<msml>: the root element that may contain a list of child elements that request a specific operation. The child elements are defined in extended packages (e.g., "msml-conf" and "msml-dialog"). This element is also the root element that contains an MSML result and event.

<send>: sends an event to the specified recipient within the media server. Specific event types are defined within the extended packages.

The media server replies with a response message containing a MSML result using the following elements:

<result>: reports the results of an MSML transaction.

The media server MAY send the MSML event to the application server, in a REPORT or CONTROL message, using the element <event>. The actual content of the <event> and which Control Framework message to use are defined within the extended packages.

MSML Conference Core Package "msml-conf"

This package extends the MSML Core Package to define a framework for creation, manipulation, and deletion of a conference.

The AS can send a CONTROL message with a body of the MSML request that contains one or multiple conference-related commands to the MS. The MS then replies with a response message with a body of the MSML result to indicate whether or not the request has been fulfilled.

During the lifetime of a conference, whenever an event occurs, the media server MAY send CONTROL messages containing MSML events to notify the application server. The application server SHOULD reply with a response message with no MSML body to acknowledge the event has been received.

This package does NOT use the REPORT message.

Dialog Core Package "msml-dialog"

This package extends the MSML Core Package to define the structural framework and abstractions for MSML dialogs.

The application server MAY send CONTROL messages containing a MSML request using the following elements:

<dialogstart>: instantiate an MSML media dialog on a connection or a conference.

<dialogend>: terminates an MSML dialog.

<send>: sends an event and an optional namelist to the dialog, dialog group, or dialog primitive.

<exit>: used by the dialog description language to cause the execution of the MSML dialog to terminate.

For the <dialogstart> command, the response message MUST contain an MSML result that indicates that the dialog has been started successfully. The MSML result MAY contain <dialogid> to return the dialog identifier, if the identifier was assigned by the media server. Subsequently, zero or more MSML events MAY be initiated by the media server in (update) REPORT messages to report information gathered during the dialog. Finally, an MSML event "msml.dialog.exit" SHOULD be generated in a (terminate) REPORT message when the dialog terminates (e.g., MSML execution of <exit>).

For the <dialogend> and <send> commands, the response message contains the final MSML result that indicates that the request has either been fulfilled or rejected.

Dialog Base Package "msml-dialog-base"

This package extends the MSML Dialog Core Package to define a set of base functionality for MSML dialogs. The extension defines individual media primitives, including <play>, <dtmfgen>, <tonegen>, <record>, <dtmf> and <collect>, to be used as child element of <dialogstart>. This package does not change the framework message usage as defined by the MSML Dialog Core Package.

Dialog Transform Package "msml-dialog-transform"

This package extends the MSML Dialog Core Package to define a set of transform primitives that works as filter on half-duplex media streams. The extension defines transform primitives, including <vad>, <gain>, <agc>, <gate>, <clamp> and <relay>, that MAY be used as child elements of <dialogstart>. This package does not change the framework message usage as defined by the MSML Dialog Core Package.

Dialog Group Package "msml-dialog-group"

This package extends the MSML Dialog Core, Base, and Transform Packages to define a single control flow construct that specifies concurrent execution of multiple media primitives. The extension defines the <group> element that MAY be used as a child element of <dialogstart> to enclose multiple media

primitives, such that they can be executed concurrently. This package does not change the framework message usage as defined by the MSML Dialog Core Package.

Dialog Speech Package "msml-dialog-speech"

This package extends the MSML Dialog Core and MSML Base Package to define functionality that MAY be used for automatic speech recognition and text to speech. The extension extends the <dialogstart> and the <play> elements.

For <dialogstart>, it defines a new child element <speech> to activate grammars or user input rules associated with speech recognition. For <play>, it defines a new child element <tts> to initiate the text-to-speech service.

This package does not change the framework message usage as defined by the MSML Dialog Core Package.

Dialog Fax Detection Package "msml-dialog-fax-detect"

This package extends the MSML Dialog Core Package to define primitives provide fax detection service. The extension defines a primitive <faxdetect> to be used as a child element of <dialogstart>. This package does not change the framework message usage as defined by the MSML Dialog Core Package.

Dialog Fax Send/Receive Package "msml-dialog-fax-sendrecv"

This package extends the MSML Dialog Core Package to define primitives that allow a media server to provide fax send or receive service. The extension defines new primitives <faxsend> and <faxrcv>, to be used as a child element of <dialogstart>. This package does not change the framework message usage as defined by the MSML Dialog Core Package.

Dialog Audit Core Package "msml-audit"

This package extends the MSML Core Package to define a framework for auditing media resource(s) allocated on the media server.

This package follows a simple request/response transaction, allowing the application server to send CONTROL messages containing MSML <audit> requests. The media server MUST reply with a response message containing the result. The result is contained within the <auditresult> element, returning the queried state information.

This package does NOT use the REPORT message.

Dialog Audit Conference Package "msml-audit-conf"

This package extends the MSML Audit Core Package to define conference specific states that MAY be queried via the <audit> command and the corresponding response MUST be returned by the <auditresult> element. This package does not change the framework message usage as defined by the MSML Audit Core Package.

Dialog Audit Connection Package "msml-audit-conn"

This package extends the MSML Audit Core Package to define connection specific states that MAY be queried via the <audit> command and the corresponding response MUST be returned by the <auditresult> element. This package does not change the framework message usage as defined by the MSML Audit Core Package.

Dialog Audit Dialog Package "msml-audit-dialog"

This package extends the MSML Audit Core Package to define dialog specific states that MAY be queried via the <audit> command and the corresponding response MUST be returned by the <auditresult> element. This package does not change the framework message usage as defined by the MSML Audit Core Package.

Dialog Audit Stream Package "msml-audit-stream"

This package extends the MSML Audit Core Package to define stream specific states that MAY be queried via the <audit> command and the corresponding response MUST returned by the <auditresult> element. This package does not change the framework message usage as defined by the MSML Audit Core Package.

3.2.3. Common XML Support

The XML schema described in [i11] MUST be supported by all Control Packages defined by MSML. However, the "connection-id" value MUST be constructed as defined by MSML (i.e., the identifier MUST contain a local dialog tag only, while the SIP Control Framework [i11] requires that the "connection-id" contain both local and remote dialog tags).

3.2.4. Control Message Body

A valid CONTROL body message MUST conform to the MSML schema, as included in this specification, for the MSML package(s) used.

3.2.5. REPORT Message Body

A valid REPORT body message MUST conform to the MSML schema, as included in this specification, for the MSML package(s) used.

4. Language Structure

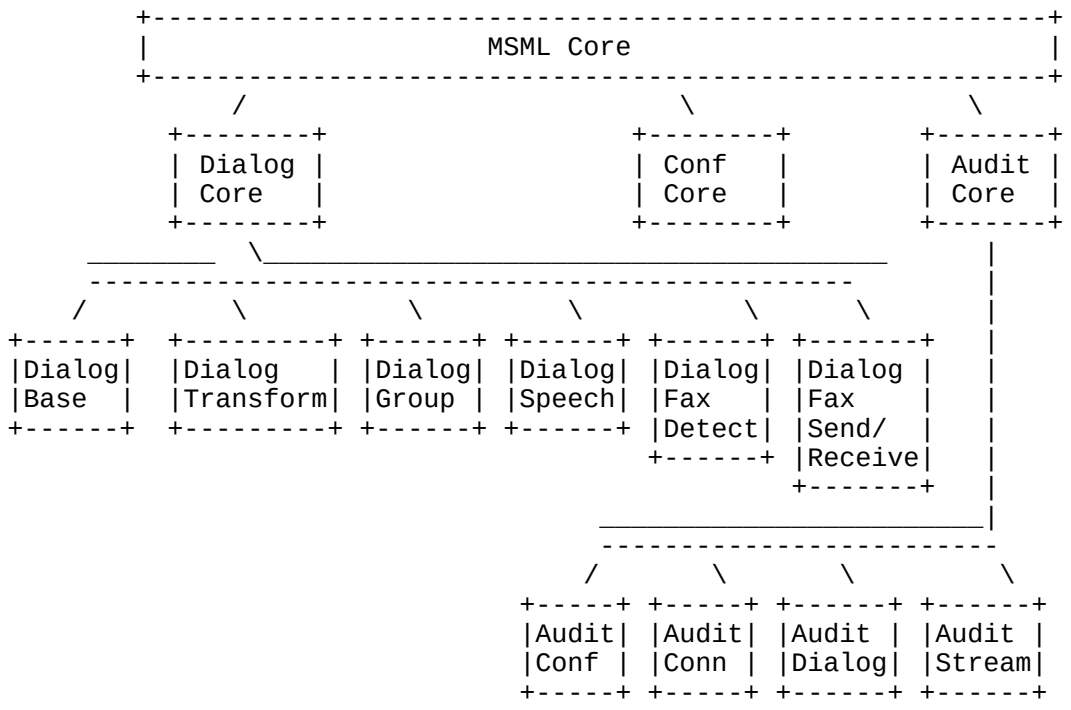
4.1. Package Scheme

The primary mechanism for extending MSML is the "package". A package is an integrated set of one or more XML schemas that define additional features and functions via new or extended use of elements and attributes. Each package, except for those defined in the current document, is defined in a separate standards document, e.g., an Internet Draft or an RFC. All packages that extend the base MSML functionality MUST include references to the MSML base set of schemas provided in the Internet Drafts. A schema in a package MUST only extend MSML; that is, it must not alter the existing specification.

A particular MSML script will include references to all the schemas defining the packages whose elements and attributes it makes use of. A particular script MUST reference MSML base and optionally extension package(s). See the IANA Considerations section.

Each package MUST define its own namespace so that elements or attributes with the same name in different packages do not conflict. A script using a particular element or attribute MUST prefix the namespace name on that element or attribute’s name if it is defined in a package (as opposed to being defined in the base).

MSML consists of a core package that provides structure without support for any specific feature set. Additional packages, relying on the core package, provide functional features. Any combination of additional packages may be used along with the core package. The following describes the set of MSML packages defined in this document.



o MSML Core Package (Mandatory)

Describes the minimum base framework that MUST be implemented to support additional core packages.

o MSML Conference Core Package (Conditionally Mandatory, for Conferencing)

Describes the audio and multimedia basic and advanced conferencing package that MAY be implemented.

o MSML Dialog Core Package (Conditionally Mandatory, for Dialogs)

Describes the dialog core package that MUST be implemented for any dialog services. However, systems supporting conferencing only, MAY omit support for MSML dialogs. The MSML Dialog Core Package specifies the framework within which additional dialog packages are supported. The MSML Dialog Base Package MUST be supported, while all other dialog packages MAY be supported.

o MSML Dialog Base Package (Conditionally Mandatory, for Dialogs)

- o MSML Dialog Group Package (Optional)
- o MSML Dialog Transform Package (Optional)
- o MSML Dialog Fax Detection Package (Optional)
- o MSML Dialog Fax Send/Receive Package (Optional)
- o MSML Dialog Speech Package (Optional)

o MSML Audit Core Package (Conditionally Mandatory, for Auditing)

Describes the audit core package that MUST be implemented to support auditing services. The MSML audit core package specifies the framework within which additional audit packages are supported.

- o MSML Audit Conference Package (Conditionally Mandatory, for Auditing Conference, Conference Dialog, and Conference Stream)
- o MSML Audit Connection Package (Conditionally Mandatory, for Auditing Connection, Connection Dialog, and Connection Stream)
- o MSML Audit Dialog Package (Conditionally Mandatory, for Auditing Dialog, and MUST be used with either MSML Audit Conference Package or MSML Audit Connection Package)
- o MSML Audit Stream Package (Conditionally Mandatory, for Auditing Stream, and MUST be used with either MSML Audit Conference Package or MSML Audit Connection Package)

The formal process for defining extensions to MSML dialogs is to define a new package. The new package MUST provide a text description of what extensions are included and how they work. It MUST also define an XML schema file (if applicable) that defines the new package (which may be through extension, restriction of an existing package, or a specific profile of an existing package). Dependencies upon other packages MUST be stated. For example, a package that extends or restricts has a dependency on the original package specification. Finally, the new package MUST be assigned a unique name and version.

The types of things that can be defined in new packages are:

- o new primitives
- o extensions to existing primitives (events, shadow variables, attributes, content)

- new recognition grammars for existing primitives
- new markup languages for speech generation
- languages for specifying a topology schema
- new predefined topology schemas
- new variables / segment types (sets & languages)
- new control flow elements

MSML packages are assembled together to form a specific MSML profile that is shared between different implementations. The base MSML dialog profiles that are defined in this document consist of the MSML Core Package, MSML Dialog Core Package, MSML Dialog Base Package, MSML Dialog Group Package, MSML Transform Package, MSML Fax Packages, and the MSML Speech Package.

MSML extension packages, which define primitives, MUST define the following for each primitive within the package:

- the function that the primitive performs
- the attributes that may be used to tailor its behavior
- the events that it is capable of understanding
- the shadow variables that provide access to information determined as a result of the primitive's operation

The mechanism used to ensure that a media server and its client share a compatible set of packages is not defined. Currently, it is expected that provisioning will be used, possibly coupled with a future auditing capability. Additionally, when used in SIP networks, packages could be defined using feature tags and the procedures defined for Indicating User Agent Capabilities in SIP [i1] used to allow a media server to describe its capabilities to other user agents.

4.2. Profile Scheme

Not all devices and applications using MSML will need to support the entire MSML schema. For example, a media processing device might support only audio announcements, only audio simple conferencing, or only multimedia IVR. It is highly desirable to have a system for describing what portion of MSML a particular media processing device or control agent supports.

The package scheme described earlier allows MSML functionality to be functionally grouped, relying on the MSML core package. This scheme allows a portion of the complete MSML specification to be implemented, on a per-package basis, and also creates a framework for future extension packages. However, within a given package, in some cases, only a subset of the package functionality may be required. In order to support subsets of packages, with greater degree of granularity than at the package level, a profile scheme is required.

MSML package profiles would identify a subset of a given MSML package with specific definitions of elements and attributes. Each MSML package profile MUST be accompanied by one or more corresponding schemas. To use the examples above, there could be an audio announcements profile of the MSML Dialog Base Package, an audio simple conferencing profile of the MSML Conference Core Package, and a multimedia IVR profile of the MSML Dialog Base Package.

MSML package profiles MUST be published separately from the MSML specification, in one or more standards documents (e.g., Internet Drafts or RFCs) dedicated to MSML package profiles. Profiles would not be registered with IANA and any organization would additionally be free to create its own profile(s) if required.

5. Execution Flow

MSML assumes a model where there is a single control context within a media server for MSML processing. That context may have one or many SIP [n1] dialogs associated with it. It is assumed that any SIP dialogs associated with the MSML control context have been authorized, as appropriate, by mechanisms outside the scope of MSML.

A media server control context maintains information about the state of all media objects and media streams within a media server. It receives and processes all MSML requests from authorized SIP dialogs and receives all events generated internally by media objects and sends them on the appropriate SIP dialog. An MSML request is able to create new media objects and streams, and to modify or destroy any existing media objects and streams.

An MSML request may simply specify a single action for a media server to undertake. In this case, the document is very similar to a simple command request. Often, though, it may be more natural for a client to request multiple actions at one time, or the client would like several actions to be closely coordinated by the media server. Multiple MSML elements received in a single request MUST be processed sequentially in document order.

An example of the first scenario would be to create a conference and join it with an initial participant. An example of the second case would be to unjoin one or more participants from a main conference and join them to a sidebar conference. In the first scenario, network latencies may not be an issue, but it is simpler for the client to combine the requests. In the second case, the added network latency between separate requests could mean perceptible audio loss to the participant.

Each MSML request is processed as a single transaction. A media server MUST ensure that it has the necessary resources available to carry out the complete transaction before executing any elements of the request. If it does not have sufficient resources, it MUST return a 520 response and MUST NOT execute the transaction.

The MSML request MUST be checked for well-formedness and validated against the schema prior to executing any elements. This allows XML [n2] errors to reported immediately and minimizes failures within a transaction and the corresponding execution of only part of the transaction.

Each element is expected to execute immediately. Elements such as <dialogstart>, which take an unpredictable amount of time, are "forked" and executed in a separate thread (see MSML Dialog Packages). Once successfully forked, execution continues with the element following the </dialogstart>. As such, MSML does not provide mechanisms to sequence or coordinate other operations with dialog elements.

Processing within a transaction MUST stop if any errors occur. Elements that were executed prior to the error are not rolled back. It is the responsibility of the client to determine appropriate actions based upon the results indicated in the response. Most elements MAY contain an optional "mark" attribute. The value of that attribute from the last successfully executed element MUST be returned in an error response. Note that errors that occur during the execution of a dialog occur outside the context of an MSML transaction. These errors will be indicated in an asynchronous event.

Transaction results are returned as part of the SIP request response. The transaction results indicate the success or failure of the transaction. The result MUST also include identifiers for any objects created by a media server for which the client did not provide an instance name. Additionally, if the transaction fails, the reason for the failure MUST be returned, as well as an indication of how much of the transaction was executed before the failure occurred SHOULD be returned.

6. Media Server Object Model

Media servers are general-purpose platforms for executing real-time media processing tasks. These tasks range in complexity from simple ones such as serving announcements, to complex ones, such as speech interfaces, centralized multimedia conferencing, and sophisticated gaming applications.

Calls are established to a media server using SIP. Clients will often use SIP third party call control (3PCC) [i4] to establish calls to a media server on behalf of end users. However MSML does not require that 3PCC be used, only that the client and the media server share a common identifier for the call and its associated RTP [i3] sessions.

Objects represent entities that source, sink, or modify media streams. A media streams is a bidirectional or unidirectional media flow between objects on a media server. The following subsections define the classes of objects that exist on a media server and the way these are identified in MSML.

6.1. Objects

A media object is an endpoint of one or more media streams. It may be a connection that terminates RTP sessions from the network or a resource that transforms or manipulates media. MSML defines four classes of media objects. Each class defines the basic properties of how object instances are used within a media server. However, most classes require that the function of specific instances be defined by the client, using MSML or other languages such as VoiceXML.

The following classes of media processing objects are defined. The class names are given in parentheses:

- network connection (conn)
- conference (conf)
- dialog (dialog)

Network connection is an abstraction for the media processing resources involved in terminating the RTP session(s) of a call. For audio services, a connection instance presents a full-duplex audio stream interface within a media server. Multimedia connections have multiple media streams of different media types, each corresponding to an RTP session. Network connections get instantiated through SIP [n1].

A conference represents the media resources and state information required for a single logical mix of each media type in the conference (e.g., audio and video). MSML models multiple mixes/views of the same media type as separate conferences. Each conference has multiple inputs. Inputs may be divided into classes that allow an application to request different media treatment for different participants. For example, the video streams for some participants may be assigned to fixed regions of the screen while those for other participants may only be shown when they are speaking.

A conference has a single logical output per media type. For each participant, it consists of the audio conference mix, less any contributed audio of the participant, and the video mix shared by all conference participants. Video conferences using voice activated switching have an optional ability to show the previous speaker to the current speaker.

Conferences are instantiated using the <createconference> element. The content of the <createconference> element specifies the parameters of the audio and/or video mixes.

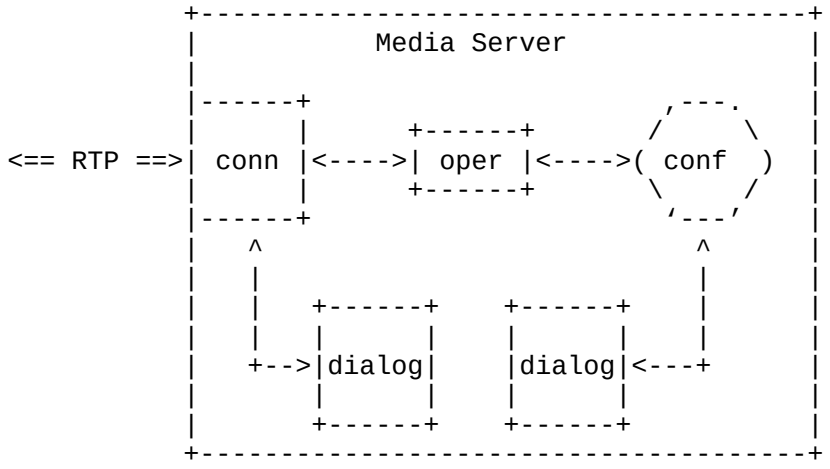
Dialogs are a class of objects that represent automated participants. They are similar to network connections from a media flow perspective and may have one or more media streams as the abstraction for their interface within a media server. Unlike connections, however, dialogs are created and destroyed through MSML, and the media server itself implements the dialog participant. Dialogs are instantiated through the <dialogstart> element. Contents of the <dialogstart> element define the desired or expected dialog behavior. Dialogs may also be invoked by referencing VoiceXML as the dialog description language.

Operators are functions that are used to filter or transform a media stream. The function that an instance of an operator fulfills is defined as a property of the media stream. Operators may be unidirectional or bidirectional and have a media type. Unidirectional operators reflect simple atomic functions such as automatic gain control, filtering tones from conferences, or applying specific gain values to a stream. Unidirectional operators have a single media input, which is connected to the media stream from one object, and a single media output, which is connected to the media stream of a different object.

Bidirectional operators have two media inputs and two media outputs. One media input and output is associated with the stream to one object, and the other input and output is associated with a stream to a different object. Bidirectional objects may treat the media differently in each direction. For example, an operator could be

defined that changed the media sent to a connection based upon recognized speech or dual-tone multi-frequency (DTMF) received from the connection. Operators are implicitly instantiated when streams are created or modified using the elements <join> and <modifystream>, respectively.

The relationships between the different object classes (conf, conn, and dialog) are shown in the figure below.



A single, full-duplex instance of each object class is shown together with common relationships between them. An operator (such as gain) is shown between a connection and a conference and dialogs are shown participating both with an individual connection and with a conference. The figure is not meant to imply only one-to-one relationships. Conferences will often have hundreds of participants, and either connections or conferences may be interacting with more than one dialog. For example, one dialog may be recording a conference while other dialogs announce participants joining or leaving the conference.

6.2. Identifiers

Objects are referenced using identifiers that are composed of one or more terms. Each term specifies an object class and names a specific instance within that class. The object class and instance are separated by a colon ":" in an identifier term.

Identifiers are assigned to objects when they are first created. In general, either the MSML client or a media server may specify the instance name for an object. Objects for which a client does not assign an instance name will be assigned one by a media server.

Media server assigned instance names are returned to the client as a complete object identifier in the response to the request that created the object.

It is meaningful for some classes of objects to exist independently on a media server. Network connections may be created through SIP at any time. MSML can then be used to associate their media with other objects as required to create services. Conferences may be created and have specific resources reserved waiting for participant connections.

Objects from these two classes, connections and conferences, are considered independent objects since they can exist on a standalone basis. Identifiers for independent objects consist of a single term as defined above. For example, identifiers for a conference and connection could be "conf:abc" or "conn:1234" respectively. Clients that choose to assign instance names to independent objects must use globally unique instance names. One way to create globally unique names is to include the domain name of the client as part of the name.

Dialogs are created to provide a service to independent objects. Dialogs may act as a participant in a conference or interact with a connection similar to a two-participant call. Dialogs depend upon the existence of independent objects, and this is reflected in the composition of their identifiers. Operators modify the media flow between other objects, such as application of gain between a connection and a conference. As operators are merely media transform primitives defined as properties of the media stream, they are not represented by identifiers and created implicitly.

Identifiers for dialogs are composed of a structured list of slash ('/') separated terms. The left-most term of the identifier must specify a conference or connection. This serves as the root for the identifier. An example of an identifier for a dialog acting as a conference participant could be:

```
conf:abc/dialog:recorder
```

All objects except connections are created using MSML. Connections are created when media sessions get established through SIP. There are several options clients and media servers can use to establish a shared instance name for a connection and its media streams.

When media servers support multiple media types, the instance name SHOULD be a call identifier that can be used to identify the collection of RTP sessions associated with a call. When MSML is used in conjunction with SIP and third party call control, the call

identifier MUST be the same as the local tag assigned by the media server to identify the SIP dialog. This will be the tag the media server adds to the "To" header in its response to an initial invite transaction. RFC 3261 requires the tag values to be globally unique.

An example of a connection identifier is: conn:74jgd63956ts.

With third party call control, the MSML client acts as a back-to-back user agent (B2BUA) to establish the media sessions. SIP dialogs are established between the client and the media server allowing the use of the media server local tag as a connection identifier. If third party call control is not used, a SIP event package MAY be used to allow a media server to notify new sessions to a client that has subscribed to this information.

Identifiers as described above allow every object in a media server to be uniquely addressed. They can also be used to refer to multiple objects. There are two ways in which this can currently be done:

wildcards

common instance names

An identifier can reference multiple objects when a wildcard is used as an instance name. MSML reserves the instance name composed of a single asterisk ('*') to mean all objects that have the same identifier root and class. Instance names containing an asterisk cannot be created. Wildcards MUST only be used as the right-most term of an identifier and MUST NOT be used as part of the root for dialog identifiers. Wildcards are only allowed where explicitly indicated below.

The following are examples of valid wildcards:

conf:abc/dialog:*

conn:*

An example of illegal wildcard usage is:

conf:*/dialog:73849

Although identifiers share a common syntax, MSML elements restrict the class of objects that are valid in a given context. As an example, although it is valid to join two connections together, it is not valid to join two IVR dialogs.

7. MSML Core Package

This section describes the core MSML package that MUST be supported in order to use any other MSML packages. The core MSML package defines a framework, without explicit functionality, over which functional packages are used.

7.1. <msml>

<msml> is the root element. When received by a media server, it defines the set of operations that form a single MSML request. Operations are requested by the contents of the element. Each operation MAY appear zero or more times as children of <msml>. Specific operations are defined within the conference package and in the set of dialog packages.

The results of a request or the contents of events sent by a media server are also enclosed within the <msml> element. The results of the transaction are included as a body in the response to the SIP request that contained the transaction. This response will contain any identifiers that the media server assigned to newly created objects. All messages that a media server generates are correlated to an object identifier. Objects and identifiers are discussed in section 6 (Media Server Object Model).

Attributes:

version: "1.1" Mandatory

7.2. <send>

Events are used to affect the behavior of different objects within a media server. The <send> element is used to send an event to the specified recipient within the media server.

Attributes:

event: the name of an event. Mandatory.

target: an object identifier. When the identifier is for a dialog, it may optionally be appended with a slash "/" followed by the target to be included in an MSML dialog <send>. Mandatory.

valuelist: a list of zero or more parameters that are included with the event.

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all mark attributes within an MSML document should be unique.

7.3. <result>

The <result> element is used to report the results of an MSML transaction. It is included as a body in the final response to the SIP request that initiated the transaction. An optional child element <description> may include text that expands on the meaning of error responses. Response codes are defined in section 11 (Response Codes).

Attributes:

response: a numeric code indicating the overall success or failure of the transaction, and in the case of failure, an indication of the reason. Mandatory.

mark: in the case of an error, the value of the mark attribute from the last successfully executed element that included the mark attribute.

In the case of failure, a description of the reason SHOULD be provided using the child element <description>.

Three other child elements allow the response to include identifiers for objects created by the request but that did not have instance names specified by the client. Those elements are <confid> and <dialogid>, for objects created through a <createconference> and <dialogstart> respectively.

7.4. <event>

The <event> element is used to notify an event to a media server client. Three types of events are defined by the MSML Core Package: "msml.dialog.exit", "msml.conf.nomedia", and "msml.conf.asn". These correspond to the termination of an executing dialog, a conference being automatically deleted when the last participant has left, and the notification of the current set of active speakers for a conference, respectively. Events may also be generated by an executing dialog. In this case, the event type is specified by the dialog (see MSML Dialog Core Package <send>).

Attributes:

name: the type of event. If the event is generated because of the execution MSML dialog <send>, the value MUST be the value of the "event" attribute from the <send> element within the MSML Dialog Core Package. If the event is generated because of the execution of an <exit>, the value MUST be "moml.exit". If the event is generated because of the execution of a <disconnect>, the value MUST be "moml.disconnect". If the event is generated because of an error, the value must be "moml.error". Mandatory.

id: the identifier of the conference or dialog that generated the event or caused the event to be generated. Mandatory.

<event> has two children, <name> and <value>, which contain the name and value respectively of each namelist item associated with the event.

8. MSML Conference Core Package

8.1. Conferences

A conference has a mixer for each type of media that the conference supports. Each mix has a corresponding description that defines how the media from participants contributes to that mix. A mixer has multiple inputs that are combined in a media specific way to create a single logical output.

The elements that describe the mix for each media type are called mixer description elements. They are:

<audiomix> defines the parameters for mixing audio media.

<videolayout> defines the composition of a video window.

These elements, defined in sections 8.6 (Audio Mix) and 8.7 (Video Layout) respectively, are used as content of the <createconference> element to establish the initial properties of a conference. The elements are used within the <modifyconference> element to change the properties of a conference once it has been created, or within the <destroyconference> element to remove individual mixes from the conference.

Conferences may be terminated by an MSML client using the <destroyconference> element to remove the entire conference or by removing the last mixer(s) associated with the conference. Conferences can also be terminated automatically by a media server based on criteria specified when the conference is created. When the

conference is deleted, any remaining participants will have their associated SIP dialogs left unchanged or deleted based on the value of the "term" attribute specified when the conference was created.

8.2. Media Streams

Objects have at least one media input and output for each type of media that they support. Each object class defines the number of input and output objects of that class support. Media streams are created when objects are joined, either explicitly using <join> or implicitly when dialogs are created using <dialogstart>. Dialog creation has two stages, allocating and configuring the resources required for the dialog instance, and implicitly joining those resources to the dialog target during the dialog execution. Refer to the MSML Dialog Base Package.

A join operation by default creates a bidirectional audio stream between two objects. Video and unidirectional streams may also be created. A media stream is created by connecting the output from one object to the input of another object and vice versa (assuming a bidirectional or full-duplex join).

Many objects may only support a single input for each type of media. Within this specification, only the conference object class supports an arbitrary number of inputs. When a stream is requested to be created to an object that already has a stream of the same type connected to its single input, the result of the request depends upon the type of the media stream.

Audio mixing is done by summing audio signals. Automatically mixing audio streams has common and straightforward applications. For example, the ability to bridge two streams allows for the easy creation of simple three-way calls or to bridge private announcements with a (whispered) conference mix for an individual participant. In the case of general conferences, however, an MSML client SHOULD create an audio conference and then join participants to the conference. Conference mixers SHOULD subtract the audio of each participant from the mix so that they do not hear themselves.

A media server receiving a request that requires joining an audio stream to the single audio input of an object that already has an audio stream connected SHOULD automatically bridge the new stream with the existing stream, creating a mix of the two audio streams. The maximum number of streams that may be bridged in this manner is implementation specific. It is RECOMMENDED that a media server support bridging at least two streams. A media server that cannot bridge a new stream with any existing streams MUST fail the operation requesting the join.

Unlike audio mixing, there are many different ways that two video streams may be combined and presented. For example, they may be presented side by side in separate panes, picture in picture, or in a single pane that displays only a single stream at a time based on a heuristic such as active speaker. Each of these options creates a very different presentation and requires significantly different media resources.

A join operation does not describe how a new stream can be combined with an existing stream. Therefore, automatic bridging of video is not supported. A media server MUST fail requests to join a new video stream to an object that only supports a single video input and already has a video stream connected to that input. For an object to have multiple video streams joined to it, the object itself must be capable in supporting multiple video streams. Conference objects can support multiple video streams and provide a way to specify the mixing presentation for the video streams.

A media server MUST NOT establish any streams unless the media server is able to create all the streams requested by an operation. Streams are only able to be created if both objects support a media type and at least one of the following conditions is true:

1. Each object that is to receive media is not already receiving a stream of that type.

2. Any object that is to receive media and is already receiving a stream of that type supports receiving an additional stream of that type. The only class of objects defined in this specification that directly support receiving multiple streams of the same type are conferences.

3. The media server is able to automatically bridge media streams for an object that is to receive media and that is already receiving a stream of the requested type. The only type of media defined in this specification that MAY be automatically bridged is audio.

The directionality of media streams associated with a connection is modeled independently from what SDP [n9] allows for the corresponding RTP [i3] sessions. Media servers MUST respect the SDP in what they actually transmit but MUST NOT allow the SDP to affect the directionality when joining streams internal to the media server.

8.3. <createconference>

<createconference> is used to allocate and configure the media mixing resources for conferences. A description of the properties for each type of media mix required for the conference is defined within the content of the <createconference> element. Mixer descriptions are described in Audio Mix and Video Layout sections. When no mixer descriptions are specified, the default behavior MUST be equivalent to inclusion of a single <audiomix>.

Clients can request that a media server automatically delete a conference when a specified condition occurs by using the "deletewhen" attribute. A value of "nomedia" indicates that the conference MUST be deleted when no participants remain in the conference. When this occurs, an "msml.conf.nomedia" event MUST be notified to the MSML client. A value of "nocontrol" indicates that the conference MUST be deleted when the SIP [n1] dialog that carries the <createconference> element is terminated. When this occurs, a media server MUST terminate all participant dialogs by sending a BYE for their associated SIP dialog. A value of "never" MUST leave the ability to delete a conference under the control of the MSML client.

Attributes:

name: the instance name of the conference. If the attribute is not present, the media server MUST assign a globally unique name for the conference. If the attribute is present but the name is already in use, an error (432) will result and MSML document execution MUST stop. Events that the conference generates use this name as the value of their "id" attribute (see section 7.4 (<event>)).

deletewhen: defines whether a media server should automatically delete the conference. Possible values are "nomedia", "nocontrol", and "never". Default is "nomedia".

term: when true, the media server MUST send a BYE request on all SIP dialogs still associated with the conference when the conference is deleted. Setting term equal to false allows clients to start dialogs on connections once the conference has completed. Default is "true".

mark: a token that MAY be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all mark attributes within an MSML document should be unique.

An example of creating an audio conference is shown below. This conference allows at most two participants to contend to be heard and reports the set of active speakers no more frequently than every 10 seconds.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <createconference name="example">
      <audiomix>
         <n-loudest n="3"/>
         <asn ri="10s"/>
      </audiomix>
   </createconference>
</msml>
```

8.3.1. <reserve>

Conference resources may be reserved by including the <reserve> element as a child of <createconference>. <reserve> allows the specification of a set of resources that a media server will reserve for the conference. Any requests for resources beyond those that have been reserved should be honored on a best-effort basis by a media server.

Attributes:

required: boolean that specifies whether <createconference> should fail if the requested resources are not available. When set to false, the conference will be created, with no reserved resources, if the complete reservation cannot be honored. Default is "true".

8.3.1.1. <resource>

The resources to be reserved are defined using <resource>. The contents of these elements describe a resource that is to be reserved. Descriptions are implementation dependent. Media servers that support MSML dialogs may use the elements from that package as the basis for resource descriptions. Each resource element may use the attribute "n" to define the quantity of the resource to reserve.

For example, the following creates a conference and reserves two types of resources. One resource element may represent resources that are shared by all participants of the conference, while the other may represent resources that are reserved for each of the expected participants.

Attributes:

n: number of resources to be reserved. Default is 1.

type: specifies whether the resource is to be reserved by each individual participant or reserved as a shared conference resource. Valid values for this attribute are "individual" or "shared". Default is "individual".

```
<createconference>
   <reserve>
      <resource n="20">
        <!--description of resources used by each participant-->
      </resource>
      <resource n="2" type="shared">
        <!--description of the shared conference resources-->
      </resource>
   </reserve>
</createconference>
```

8.4. <modifyconference>

All of the properties of an audio mix or the presentation of a video mix may be changed during the life of a conference using the <modifyconference> element. Changes to an audio mix are requested by including an <audiomix> element as a child of <modifyconference>. This may also be used to add an audio mixer to the conference if none was previously allocated. Changes to a video presentation are requested by including a <videolayout> element as a child of <modifyconference>. Similar to an audio mixer, this may be used to add a video mixer if none was previously allocated.

Mixers are removed by including a mixer description element within <destroyconference/>.

Features and presentation aspects are enabled/added or modified by including the element(s) that define the feature or presentation aspect within a mixer description. The complete specification of the element must be included just as it would be included when the conference is created. The new definition completely replaces any previous definition that existed. Only things that are defined by elements included in the mixer descriptions are affected. Any existing configuration aspects of a conference, which are not specified within the <modifyconference/> element, MUST maintain their current state in the media server.

For example, if an MSML client wanted to change the minimum reporting interval for active speaker notification from that shown in the Conference Examples section (<createconference>) it would send the following to the media server:

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <modifyconference id="conf:example">
      <audiomix>
         <asn ri="4"/>
      </audiomix>
   </modifyconference>
</msml>
```

This would also enable active speaker notification if it had not previously been enabled. The N-loudest mixing is unaffected.

Multiple elements MAY be included in the mixer descriptions similar to when conferences are created. For example, in a video conference, the video mix description (<videolayout>) could specify that the layout of the video being displayed should change such that the regions currently displaying participants get smaller and new region(s) are created to support additional participants. A media server MUST make all of the requested changes or none of the requested changes.

Additional examples of modifying conferences are presented in the Conference Examples section.

Attributes:

id: the identifier for a conference. Wildcards MUST NOT be used. Mandatory.

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all "mark" attributes within an MSML document SHOULD be unique.

8.5. <destroyconference>

Destroy conference is used to delete mixers or to delete the entire conference and all state and shared resources. When a mixer is removed, all of the streams joined to that mixer are unjoined. When a conference is destroyed, SIP dialogs for any remaining participants MUST be maintained or removed based on the value of the "term" attribute when the conference was created.

When there is no element content, <destroyconference/> deletes the entire conference. Individual mixers are removed by including a mixer description element identifying the mix (or mixes) to be removed as content to <destroyconference/>. <audiomix/> is used remove audio mixers and <videolayout/> is used remove video mixers. When one or more mixer descriptions are specified, then media server MUST only delete the specified mixer and MUST NOT affect any other existing mixers. When <audiomix/> or <videolayout/> is identified for individual removal, other feature aspects of the mix MUST NOT be included. If specified, the media server MUST ignore any such elements. When the last mixer is removed from a conference, a media server MUST remove all conference state, leaving or removing any remaining SIP dialogs as described above.

Attributes:

id: the identifier for a conference. Mandatory.

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all "mark" attributes within an MSML document SHOULD be unique.

8.6. <audiomix>

The properties of the overall audio mix are specified using the <audiomix> element.

Attributes:

id: an optional identifier for the audio mix.

samplerate: Integer value specifies the sample rate (in Hz) for the audio mixer. Optional, default value of 8000.

An example of the description for an audio mix is:

```
<audiomix id="mix1">
  <asn ri="10s"/>
  <n-loudest n="3"/>
</audiomix>
```

8.6.1. <n-loudest>

The <n-loudest> element defines that participants contend to be included in the conference mix based upon their audio energy. When the element is not present, all participants are mixed.

Attributes:

n: the number of participants that will be included in the audio mix based upon having the greatest audio energy. Mandatory.

8.6.2. <asn>

The <asn> element enables notification of active speakers. Active speakers MUST be notified using the <event> element with an event name of "msml.conf.asn". The namelist of the event consists of the set of active speakers. The name of each item is the string "speaker" with a value of the connection identifier for the connection.

Attributes:

ri: the minimum reporting interval defines the minimum duration of time that must pass before changes to active speakers will be reported. A value of zero disables active speaker notification.

asth: specifies the active speaker threshold (in unit of dBm0). Valid value range is 0 to -96. Optional, default is -96.

An example of an active speaker notification is:

```
<event name="msml.conf.asn" id="conf:example">
   <name>speaker</name>
   <value>conn:hd93tg5hdf</value>
   <name>speaker</name>
   <value>conn:w8cn59vei7</value>
   <name>speaker</name>
   <value>conn:p78fnh6sek47fg</value> </event>
```

8.7. <videolayout>

A video layout is specified using the <videolayout> element. It is used as a container to hold elements that describe all of the properties of a video mix. The parameters of the window that displays the video mix are defined by the <root> element. When the video mix in composed of multiple panes, the location and characteristics of the panes are defined by one or more <region> elements. A <region> element is not required when only a single video stream is displayed at one time and none of the visual attributes of regions are required.

Some regions may be used to display a video stream based on a selection criteria rather than having a video stream of a single participant continuously presented in the region. One such an

example is a distance learning lecture where the instructor sees each of the students periodically displayed in a region. When a region is used to display one of a number of streams, it is placed as a child of a <selector> element.

Attributes:

type: specifies the language used to define the layout. Layouts defined using MSML MUST use the value "text/msml-basic-layout". This is the same convention as defined for the layout package from the W3C SMIL 2.0 specification [i6]. The default when omitted is "text/msml-basic-layout".

id: an optional identifier for the video layout.

8.7.1. <root>

The <root> element describes the root window or virtual screen in which the conference video mix will be displayed. Simple conferences can display participant video directly within the root window but more complex conferences will use regions for this purpose. Areas of the window which are not used to display video will show the root window background.

All video presentations require a root window. It MUST be present when a video mix is created and it cannot be deleted; however, its attributes MAY be changed using the <modifyconference> element.

Attributes:

size: the size of the root window specified as one of the five standard common intermediate formats (e.g., CIF, QCIF).

backgroundcolor: the color for the root window background defined using the values for the "background-color" property of the CSS2 specification [n10].

backgroundimage: the URI for an image to be displayed as the root window background. Transparent portions of the image allow the background color to show through.

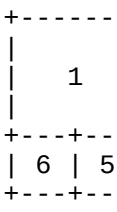
8.7.2. <region>

<region> elements define video panes that are used to display participant video streams. Regions are rendered on top of the root window.

The size of a region is specified relative to the size of the root window using the "relativesize" attribute. Relative sizes are expressed as fractions (e.g., 1/4, 1/3) that preserve the aspect ratio of the original video stream while allowing for efficient scaling implementations.

Regions are located on the root window based on the value of the position attributes "top" and "left". These attributes define the position of the top left corner of the region as an offset from the top left corner of the root window. Their values may be expressed either as a number of pixels or as a percent of the vertical or horizontal dimension of the root window. Percent values are appended with a percent ('%') character. Percent values of "33%" and "67%" should be interpreted as "1/3" and "2/3" to allow easy alignment of regions whose size is expressed relative to the size of the root window.

An example of a video layout with six regions is:




```
<videolayout type="text/msml-basic-layout">
   <root 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"/>
</videolayout>
```

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. A value of zero disables the region, freeing any resources associated with the region, and unjoining any video stream displayed in the region.

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 <videolayout> 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 <videolayout> element.

For example, if a layout was specified as follows:

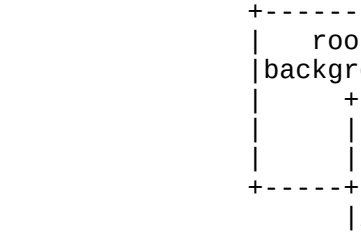
```
<videolayout>
   <root size="CIF"/>
   <region id="a" ... priority=".3" .../>
   <region id="b" ... />
   <region id="c" ... priority=".2" ...>
   <region id="d" ... />
</videolayout>
```

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

Portions of regions that extend beyond the root window will be cropped. For example, a layout specified as:

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

would appear similar to:



Visual attributes are used to define aspects of the visual appearance of individual regions. A border may be defined together with a title and/or logo. Text and logos are displayed as images on top of the region’s video, below all regions with a lower priority. The visual attributes are "title", "titletextcolor", "titlebackgroundcolor", "bordercolor", "borderwidth", and "logo".

Visual attributes can also be defined for individual streams (Video Stream Properties). When visual attributes are specified as part of both a region and a stream, those associated with the stream MUST take precedence. This allows streams that are chosen for display automatically (Stream Selection) to have proper text and logos displayed. The region visual attributes are displayed when no stream is associated with the region.

Two other attributes associated with a region, "blank" and "freeze", define the state of the video displayed in the region. When the blank or freeze attribute is assigned the value "true", then the media server MUST display the region either as a blank region, or the video image frozen at the last received frame.

These attributes are specified for a region and not allowed for streams because that appears to be the common use case. Applying them to streams would allow only that stream to be affected within a selector while other streams continue to display normally. Except for personal mixing scenarios, the same effect can be achieved by having the participant mute their own transmission to the media server.

Attributes: associated with each region:

id: a name that can be used to refer to the region.

left: the position of the region from the left side of the root window.

top: the position of the region from the top of the root window.

relativesize: the size of the region expressed as a fraction of the root window size.

priority: a number between 0 and 1 that is used to define the precedence when rendering overlapping regions. A value of zero disables the region.

title: text to be displayed as the title for the region

titletextcolor: the color of the text

titlebackgroundcolor: the color of the text background

bordercolor: the color of the region border

borderwidth: the width of the region border

logo: the URI of an image file to be displayed

freeze: a boolean value, with a default of "false", that defines whether the video image should be frozen at the currently displayed frame

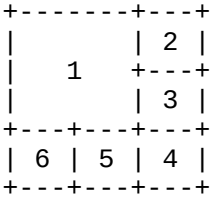
blank: a boolean value, with a default of "false", that defines whether the region should display black instead of the associated video stream

8.7.3. <selector>

It is often desired that one of several video streams be automatically selected to be displayed. The <selector> element is used to define the selection criteria and its associated parameters. The selection algorithm is specified by the "method" attribute. Currently defined selection methods allow for voice activated switching and to iterate sequentially through the set of associated video streams.

The regions that will display the selected video stream are placed as child elements of the <selector> element. Including regions within a <selector> element does not affect their layout with respect to regions not subject to the selection. For simple video conferences that display the video directly in the root window, the <root> element can be placed as a child of <selector>. Region elements MUST NOT be used in this case.

For example, below is a common video layout that allows the video stream from the currently active speaker to be displayed in the large region ("1") at the top left of the layout while the streams from five other participants are displayed in regions located at the layout periphery.



```
<videolayout type="text/msml-basic-layout">
   <root size="CIF"/>
   <selector id="switch" method="vas">
      <region id="1" left="0" top="0" relativesize="2/3"/>
   </selector>
   <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"/>
</videolayout>
```

All selector methods must be defined so that they work if only a single region is a child of the selector. Selector methods that support more than one child region MUST specify how the method works across multiple regions. Media server implementations MAY support only a single region for methods that are defined to allow multiple regions.

The selector or region for a participant's video is defined using the "display" attribute of <stream> during a join operation. Specifying a selector allows the stream to be displayed according to the criteria defined by the selector method. Specifying a region supports continuous presence display of participants. Some streams may be joined with both a selector and a region. In this case, the value of <blankothers> attribute defines whether the streams associated with a continuous presence region should be blanked when the stream is selected for display in one of the selector regions.

Attributes: common to all selector methods are:

id: a name that can be used to refer to the selector.

method: the name of the method used to select the video stream. A value of "vas" (see the following section, Voice Activated Switching) MAY be specified.

status: specifies whether the selector is "active" or "disabled".

blankothers: when "true", video streams that are also displayed in continuous presence regions will have the continuous presence regions blanked when the stream is displayed in a selection region.

8.7.3.1. Voice Activated Switching ("vas")

Voice activated switching (VAS) is used to display the video stream that correlates with the participant who is currently speaking. It is specified using a selector method value of "vas".

If the video stream associated with the active speaker is not currently displayed in a selection region, then it replaces the video in the region that is displaying the video of the speaker that was least recently active. If the video of the active speaker is currently displayed in a selection region, then there is no change to any region. When VAS is applied to a single region, this has the effect that the current speaker is displayed in that region.

Attributes:

si: switching interval is the minimum period of time that must elapse before allowing the video to switch to the active speaker.

speakersees: defines whether the active speaker sees the "current" speaker (themselves) or the "previous" speaker.

8.8. <join>

<join> is used to create one or more streams between two independent objects. Streams may be audio or video and may be bidirectional or unidirectional. A bidirectional stream is implicitly composed of two unidirectional streams that can be manipulated independently. The streams to be established are specified by <stream> elements (section <stream>) as the content of <join>.

Without any content, <join> by default establishes a bidirectional audio stream. When only a stream of a single type has previously been created between two objects, or when only a unidirectional stream exists, <join> can be used to add a stream of another media type or make the stream bidirectional by including the necessary <stream> elements. Bidirectional streams are made unidirectional by using <unjoin> (section <unjoin>) to remove the unidirectional stream for the direction that is no longer required.

In addition to defining the media type and direction of streams, <stream> elements are also used to establish the properties of streams, such as gain, voice masking, or tone clamping of audio streams, or labels and other visual characteristics of video streams. Properties are often defined asymmetrically for a single direction of a stream. Creating a bidirectional stream requires two <stream> elements within the <join>, one for each direction, if one direction is to have different properties from the other direction.

If a media server can provide services using both compressed or uncompressed media, the MSML client may need to distinguish within requests which format is to be used. When compressed streams are created, both objects must use the same media format or an error response (450) is generated.

Attributes:

id1: an identifier of either a connection or conference. Wildcards MUST NOT be used. Mandatory. Any other object class results in a 440 error.

id2: an identifier of either a connection or conference. Wildcards MUST NOT be used. Mandatory. Any other object class results in a 440 error.

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all mark attributes within an MSML document SHOULD be unique.

For example, consider a call center coaching scenario where a supervisor can listen to the conversation between an agent and a customer and provide hints to the agent, which are not heard by the customer. One join establishes a stream between the agent and the customer and another join establishes a stream between the agent and the supervisor. A third join is used to establish a half-duplex stream from the customer to the supervisor. The media server automatically bridges the media streams from the customer and the supervisor for the agent, and from the customer and the agent for the supervisor.

Assuming the following connections, each with a single audio stream:

conn:supervisor

conn:agent

conn:customer

The following would create the media flows previously described:

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <join id1="conn:supervisor" id2="conn:agent"/>
   <join id1="conn:agent" id2="conn:customer"/>
   <join id1="conn:supervisor" id2="conn:customer">
      <stream media="audio" dir="to-id1"/>
   </join>
</msml>
```

The following example shows joining a participant to a multimedia conference. It assumes that the conference has a video presentation region named "topright". The "display" attribute is explained in the section Video Stream Properties.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <join id1="conn:hd83t5hf7g3" id2="conf:example">
      <stream media="audio"/>
      <stream media="video" dir="from-id1" display="topright"/>
      <stream media="video" dir="to-id1"/>
   </join>
</msml>
```

8.9. <modifystream>

Media streams can have different properties such as the gain for an audio stream or a visual label for a video stream. These properties are specified as the content of <stream> elements (section <stream>). <modifystream> is used to change the properties of a stream by including one or more <stream> elements that are to have their properties changed.

Stream properties MUST be set as specified by the element <stream> as a child element of <modifystream> element. Any properties not included in the <stream> element when modifying a stream MUST remain unchanged. Setting a property for only one direction of a bidirectional stream MUST NOT affect the other direction. The directionality of streams can be changed by issuing an <unjoin> followed by a <join>. Any streams that exist between the two objects that are not included within <modifystream> MUST NOT be affected.

Attributes:

id1: an identifier of either a conference or a connection. The instance name MUST NOT contain a wildcard if "id2" contains a wildcard. Mandatory.

id2: an identifier of either a conference or a connection. The instance name MUST NOT contain a wildcard if "id1" contains a wildcard. Mandatory.

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all mark attributes within an MSML document is RECOMMENDED to be unique.

8.10. <unjoin>

Unjoin removes one or more media streams between two objects. In the absence of any content in the <stream> element, all media streams between the objects MUST be removed. Individual streams may be removed by specifying them using <stream> elements, while the unspecified streams MUST NOT be removed. A bidirectional stream is changed to a unidirectional stream by unjoining the direction that is no longer required, using the <unjoin> element. Operator elements MUST NOT be specified within <stream> elements when streams are being unjoined using the <unjoin> element. Any specified stream operators MUST be ignored.

<unjoin> and <join> may be used together to move a media stream, such as from a main conference to a sidebar conference.

Attributes:

id1: an identifier of either a conference or a connection. The instance name MUST NOT contain a wildcard if "id2" contains a wildcard. Mandatory.

id2: an identifier of either a conference or a connection. The instance name MUST NOT contain a wildcard if "id1" contains a wildcard. Mandatory.

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all mark attributes within an MSML document SHOULD be unique.

The following removes a participant from a conference and plays a leave tone for the remaining participants in the conference.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <unjoin id1="conn:jd73ht89sf489f" id2="conf:1"/>
   <dialogstart target="conf:1" type="application/moml+xml">
      <play>
         <audio uri="file://leave_tone.wav"/>
      </play>
   </dialogstart>
</msml>
```

8.11. <monitor>

Monitor is a specialized unidirectional join that copies the media that is destined for a connection object. One example of the use for <monitor> may be quality monitoring within a conference. The media stream may be removed using the <unjoin> element (see the section <unjoin>).

Attributes:

id1: an identifier of the connection to be monitored. Mandatory. Any other object class results in a 440 error. Wildcards MUST NOT be used.

id2: an identifier of the object that is to receive the copy of the media destined to id1. id2 may be a connection or a conference. Mandatory. Any other object class results in a 440 error. Wildcards MUST NOT be used.

compressed: "true" or "false". Specifies whether the join should occur before or after compression. When "true", id2 must be a connection using the same media format as id1 or an error response (450) is generated. Default is "false".

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all mark attributes within an MSML document SHOULD be unique.

8.12. <stream>

Individual streams are specified using the <stream> element. They MAY be included as a child element in any of the stream manipulation elements <join>, <modifystream>, or <unjoin>.

The type of the stream is specified using a "media" attribute that uses values corresponding to the top-level MIME media types as defined in RFC 2046 [i7]. This specification only addresses audio and video media. Other specifications may define procedures for additional types.

A bidirectional stream is identified when no direction attribute "dir" is present. A unidirectional stream is identified when a direction attribute is present. The "dir" attribute MUST have a value of "from-id1" or "to-id1" depending on the required direction. These values are relative to the identifier attributes of the parent element.

The compressed attribute is used to distinguish the compressed nature of the stream when necessary. It is implementation specific what is used when the attribute is not present. Joining compressed streams acts much like an RTP [i3] relay.

The properties of the media streams are specified as the content of <stream> elements when the element is used as a child of <join> or <modifystream>. Stream elements MUST NOT have any content when they are used as a child of <unjoin> to identify specific streams to remove.

Some properties are defined within MSML as additional attributes or child elements of <stream> that are media type specific. Ones for audio streams and video streams are defined in the following two sub-sections. Operators, viewed as properties of the media stream, MAY be specified as child elements of the <stream> element.

Attributes:

media: "audio" or video". Mandatory

dir: "from-id1" or "to-id1".

compressed: "true" or "false". Specifies whether the stream uses compressed media. Default is implementation specific.

8.12.1. Audio Stream Properties

Audio mixes can be specified to only mix the N-loudest participants. However, there may be some "preferred" participants that are always able to contribute. When audio streams are joined to a conference that uses N-loudest audio mixing, preferred streams need to be identified.

A preferred audio stream is identified using the "preferred" attribute. The "preferred" attribute MAY be used for an audio stream that is input to a conference and MUST NOT be used for other streams.

Additional attributes of the <stream> element for audio streams are:

Attributes:

preferred: a boolean value that defines whether the stream does not contend for N-loudest mixing. A value of "true" means that the stream MUST always be mixed while a value of "false" means that the stream MAY contend for mixing into a conference when N-loudest mixing is enabled. Default is "false".

There are two elements that can be used to change the characteristics of an audio stream as defined below.

8.12.1.1. <gain>

The <gain> element may be used to adjust the volume of an audio media stream. It may be set to a specific gain amount, to automatically adjust the gain to a desired target level, or to mute the stream.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the gain primitive.

amt: a specific gain to apply specified in dB or the string "mute" indicating that the stream should be muted. This attribute MUST NOT be used if "agc" is present.

agc: boolean indicating whether automatic gain control is to be used. This attribute MUST NOT be used if "amt" is present.

tgtlvl: the desired target level for AGC specified in dBm0. This attribute MUST be specified if "agc" is set to "true". This attribute MUST NOT be specified if "agc" is not present.

maxgain: the maximum gain that AGC may apply. Maxgain is specified in dB. This attribute MUST be used if "agc" is present and MUST NOT be used when "agc" is not present.

8.12.1.2. <clamp>

The <clamp> element is used to filter tones and/or audio-band dtmf from a media stream.

Attributes:

dtmf: boolean indicating whether DTMF tones should be removed.

tone: boolean indicating whether other tones should be removed.

8.12.2. Video Stream Properties

Video mixes define a presentation that may have multiple regions, such as a quad-split. Each region displays the video from one or more participants. When video streams are joined to such a conference, the region that will display the video needs to be specified as part of the join operation.

The region that will display the video is specified using the "display" attribute. The "display" attribute MUST be used for a video stream that is input to a conference and MUST NOT be used for other streams. The value of the attribute MUST identify a <region> (see the section <region>) or a <selector> (see the section <selector>) that is defined for the conference. A stream MUST NOT be directly joined to a region that is defined within a selector. Changing the value of the "display" attribute can be used to change where in a video presentation layout a video stream is displayed.

Additional attributes of the <stream> element for video streams are:

Attributes:

display: the identifier of a video layout region or selector that is to be used to display the video stream.

override: specifies whether or not the given video stream is the override source in the region defined by "display" attribute. Valid values are "true" or "false". Optional, default value is "false". Only a video stream that is input to a conference can be the override source. A particular region can have at most one override source at a time. The most recently joined video stream with this attribute set to "true" becomes the override source. When there’s an override source in place, its video is always displayed in the region, regardless of what video selection algorithm (either a selector or continuous presence mode) is configured for that region. Once the override source is cleared, the conference MUST revert back to original video selection algorithm.

8.12.2.1. <visual>

Some regions of video conferences may display different streams automatically, such as when voice activated switching is used. Connections MAY also be joined directly without the use of video mixing. In these cases, the <visual> element may be used to define visual display properties for a stream.

The <visual> element MAY use any of the visual attributes defined for regions (see the section <region>). This allows the visual aspects of regions within a <selector> to be tailored to the selected video stream, or for streams that are directly joined to display a name or logo.

9. MSML Dialog Packages

9.1. Overview

MSML Dialog Packages define an XML [n2] language for composing complex media objects from a vocabulary of simple media resource objects called primitives. It is primarily a descriptive or declarative language to describe media processing objects. MSML dialogs operate on a single or multiple streams that are identified by the MSML document outside the scope of the MSML Dialog Package.

MSML dialogs are intended to be used in different environments. As such, the language itself does not define how an MSML dialog is used. Each environment in which an MSML dialog is used must define how it is used, the set of services provided, and the mechanism for passing information between the environment and MSML dialog. The specific mechanisms used to realize the interface between MSML dialog and its environment are platform specific.

MSML Dialog Packages provide two models for access to media resources and service creation building blocks. Both models MAY be used in conjunction with each other in a complementary manner. The first model (referred to as "Media Primitives and Composites", part of the mandatory MSML Dialog Base Package) contains media primitives (such as digit collection and announcements) and composite functions (such as play and collect combined as a single operation). The second model (referred to as "Media Groups", part of the optional MSML Dialog Group Package) allows the ability to define complex customized interactions, via event passing mechanisms, between media primitives, if required.

MSML Dialog Core Package

Defines core framework over which all MSML Dialog Packages operate.

MSML Dialog Base Package

Media Primitives

<dtmf> or <collect>
DTMF digit collection

<play>
Playing of Announcements

<dtmfgen>
Generation of DTMF digits

<tonegen>
Tone genration

<record>
Media recording

Media Composites

<collect>
Supports play and collect operation.
Composite function with inclusion of play.

<record>
Supports play and record operation.
Composite function with inclusion of play.

MSML Dialog Group Package

<group>
Allows grouping of media primitives for parallel execution, with an event exchange mechanism between the media primitives to achieve customized media operations. All the above media primitive elements are accepted within the group.

The following operations MUST be supported using elements described above using either the MSML Dialog Base Package or MSML Dialog Group Package.

Announcement only
<play>

Collection only
<dtmf> or <collect>

Recording only
<record>

```
            Play and Collect
                        <collect>
                           <play/>
                        </collect>

            Play and Record
                        <record>
                           <play/>
                        </record>
```

Additional MSML Dialog Packages are:

- o MSML Dialog Transform Package
- o MSML Dialog Speech Package
- o MSML Fax Detection Package
- o MSML Fax Send/Receive Package

MSML dialogs MAY be used to simply expose primitive media resource objects but will be used more often to describe dialog operations and media transformation objects that can be controlled via user interaction.

MSML dialogs do not contain any computation or flow control constructs. There are no results automatically generated when media operations complete. Results MUST be explicitly requested using a <send> or <exit> element within the definition of the MSML dialog.

9.2. Primitives

Primitives perform a single function on a media stream or multiple streams such as generating audio/video, recognizing speech or DTMF, or adjusting the gain. They may be composed so that primitives execute concurrently. Primitives not composed for concurrent execution MUST simply execute sequentially in the order they occur in an MSML document. All concurrently executing primitives in the same MSML object (defined in one MSML document) MAY interact with each other through events (see MSML Dialog Group Package).

Primitives are categorized into one of the following descriptive categories.

- o Recognizers have a media input but no output. They allow different things within a media stream to be recognized or detected and for events to be generated based upon received media.

- Transformers have one media input and output and may send and receive events.

- Sources and sinks generate or consume media. They have either a media input or a media output but not both. They may receive and generate events.

- Composites combine underlying primitives to provide higher-level user interaction, without the need for specific event-based exchange between the primitives. The composite elements provide a simpler mechanism for more commonly used services, such as play and collect or play and record.

Primitives may define different media processing behavior (states) based upon the events that they receive. Primitives that support different processing states must define their default starting state and should support the "initial" attribute to allow that state to be specified when the primitive is instantiated. All primitives must support the "terminate" event class.

The following types of primitives are defined within this specification:

Recognizers	Transformers	Source/Sink	Composites
dtmf/collect	agc	play	dtmf/collect
faxdetect	clamp	record	record
speech	gain	dtmfgen	
vad	gate	tonegen	
	relay	faxsend	
		faxrcv	

Primitives have shadow variables, similar to those within VoiceXML [n5], which are automatically assigned values when the primitives are used. Upon initialization of an MSML dialog context, all shadow variables have the string value "undefined". Each primitive has its own instance of shadow variables that are global in scope to the entire MSML dialog context.

Names SHOULD be assigned to individual primitives when more than one primitive of the same type is used within one MSML document. Shadow variables are overwritten if the primitive has not been named and is instantiated a second time.

Shadow variables cannot be modified under user control. They may be returned from the MSML dialog context using the <send> element.

9.3. Events

Events provide the mechanism for primitives to interact with each other and for an MSML context to interact with its external environment. The external environment is defined by the way in which an MSML context has been invoked. This will often be through MSML, but other languages and protocols such as SIP may also be used.

Every primitive and group conceptually implements their own event queue. Events sent to them get placed into their associated queue. Events are removed from their queues and processed in order. Primitives within a group conceptually have their own thread of execution. Due to the asynchronous nature of servicing events from multiple queues, it cannot be assumed that several events sent in sequence to different queues will be processed in the order in which they were sent. For example, if recognition of something led to sending events to both a <play> and a <record> in that order, it is possible that the <record> may process its event before the <play>.

Primitives each define the set of events that they support and the behavior associated with their handling of each event. This allows many types of behaviors to be defined. For example, VCR type controls can be constructed by defining primitives that support events corresponding to each control. Media recognition/detection can be used to cause those events to be generated.

Alternatively, events can be originated elsewhere, such as from a control agent, and simply received by the primitive implementing the control. Examples of the use of events include adjusting volume (gain) and pause and resume of both announcement playout and record creation.

Primitives act on events based upon the longest match of an event name. Event names are a period '.' delimited sequence of tokens. The first token, or the root of the name, can be considered an event class. Matching allows a standard meaning to be defined and then extended based upon what triggers an event's generation. For example, a record primitive has different behavior depending upon whether it completed because a user stopped speaking or because it was cancelled. The recording is retained in the first case but not the second.

Longest match allows new recognizers to be created and used without changing how existing primitives are defined. For example, a face recognition capability could be created that generates a terminate.frowning event when a user looks puzzled. Although no primitive directly defines this event, it will still effect a generic terminate action. Primitives that require specialized behavior based

upon frowning may be extended to support this. As well, the event can still be exported from the MSML context without requiring that primitives receiving the event understand facial expressions.

9.4. MSML Dialog Usage with SIP

MSML dialogs MAY be used directly with SIP for dialog interactions (e.g., IVR or fax). It can be initially invoked as part of the "Prompt and Collect" service described in "Basic Network Media Services with SIP" [n7]. That defines service indicators for a small number of well-defined services using the user part of the SIP Request-URI (R-URI).

The prompt and collect service uses "dialog" as the service indicator. URI parameters further refine the specific IVR request. This document defines an additional parameter "msml-param" for the dialog service indicator as follows:

```
dialog-parameters = ";" ( dialog-param [ vxml-parameters ] )
                        | moml-param
dialog-param      = "voicexml=" dialog-url
moml-param        = "moml=" moml-url
```

There are no additional URI parameters when MSML is used as the dialog language.

MSML dialogs define discrete IVR dialog commands. These commands MAY be included directly in the body of the INVITE to the "dialog" service indicator by using the "cid" [n8] URL scheme. This scheme identifies a message body part that in this case would contain the MSML dialog request. Note that a multipart message body, containing a single part, MUST be present even if the INVITE does not contain an SDP offer. Subsequent MSML dialog requests are sent in the body of SIP INFO messages as are all messages from a media server.

An example of SIP URI as described above is:

```
sip:dialog@mediaserver.example.net;\
    moml=cid:14864099865376@appserver.example.net
```

The body part that contained the MSML dialog referenced by the URL would have a Content-Id header of:

```
Content-Id: <14864099865376@appserver.example.net>
```

The results of executing an <exit> or <disconnect>, or of executing a <send> that has a "target" attribute value equal to "source", are notified in SIP INFO messages using the <event> element from MSML Core package. No messages are sent if execution completes normally without executing one of these elements.

If there is an error during validation or execution, then a media server MUST notify the error as described above and must include the namelist items "moml.error.status" and "moml.error.description". The values for these items are defined in section 11.

A restricted subset of MSML dialogs can also be used with the "Announcement" service defined in [n7]. This service uses "annc" as the service indicator and defines parameters that describe an announcement. The "play=" parameter identifies the URL of a prompt or a provisioned announcement sequence. The value of the "play=" parameter can refer to an MSML dialog body part using a "cid" URL as described above. That body part must only contain the <play> primitive.

Using MSML dialogs enhances the announcement service by allowing the client to specify a sequence of audio segments rather than requiring each sequence to be provisioned as well as support for video. Moreover, MSML dialogs define a standard set of variables in contrast to [n7] which defines a parameterization mechanism but does not formally specify any semantics.

If a media server does not understand the "cid" scheme or does not understand MSML dialogs, it must respond with the SIP response code "488 - not acceptable here". If the MSML dialog body contains elements other than the <play> primitive, or there are errors during validation, a media server must respond with a SIP response code "400 - bad request". Finally, if there is a discrepancy between parameters specified in the Request-URI and corresponding attributes defined in the MSML dialog body, the Request-URI parameters must be silently ignored.

MSML dialogs MUST NOT change the operation of the announcement service from that defined in [n7]. When the announcement completes, a media server issues a SIP BYE request. The INFO method MUST NOT used with the announcement service.

9.5. MSML Dialog Structure and Modularity

MSML is structured as a set of packages. Only the core and base packages are required. The Dialog Core Package defines the framework for MSML requests to a media server, without specific functionality. It consists of the "primitive" abstraction, an abstract element for

control flow, the sequential execution model, and the <send> element. That is, the MSML Dialog Core Package allows for the execution of a sequence of one or more media processing primitives with the ability to notify events to the invocation environment.

Primitives are contained within the MSML Dialog Base Package, which defines the basic <play>, <record>, <dtmf>, <dtmfgen>, <tonegen>, and <collect> elements. Another package, the MSML Dialog Transform Package, defines the simple half-duplex filters. More advanced primitives are defined in the speech and fax packages. The MSML speech package depends on the MSML Dialog Base Package as it extends the capability of <play> by adding synthesized speech. Finally, the group execution model, which is currently the only element that changes the flow of control, is defined in a separate MSML Dialog Group Package. All of these packages are optional with the exception that MSML Dialog Core and MSML Dialog Base Packages MUST be implemented to provide the minimal functionality.

9.6. MSML Dialog Core Package

The MSML Dialog Core Package defines the structural framework and abstractions for MSML dialogs (via its schema). It also defines the basic elements that are not part of the core primitive or control abstractions. This package is dependent on the MSML Core Package. Events generated by MSML dialogs, such as prompt completion, digits collected, or dialog termination, are communicated by the media server via the MSML Core Package (see MSML Core Package <event>).

MSML dialogs are executed independently from the MSML core context. When an MSML dialog is started, MSML allocates the dialog control resources, and if successful, starts those resources executing. MSML core execution then continues without waiting for the MSML dialog to complete. This forking of MSML dialog invocation from the MSML core context is done via the <dialogstart> element. Media streams are created between the MSML dialog target and other internal media server resources as part of dialog execution. Stream creation is subject to the requirements defined in the MSML Core Package and media streams as defined by the MSML Conference Core Package.

9.6.1. <dialogstart>

The <dialogstart> element is used to instantiate an MSML media dialog on connections or conferences. The dialog is specified either inline or by a URI [n6]. Inline dialogs MUST be composed of any of the MSML Dialog Packages. MSML dialogs MAY be defined externally as VoiceXML [n5]. The MSML dialog description MUST NOT be inline if the src attribute, containing a URI, is present.

The originator of the MSML dialog is notified using a "msml.dialog.exit" event when the dialog completes. Any results returned by the dialog when it exits are sent as a namelist to the event.

The "msml.dialog.exit" event is also used when dialogs fail due to errors encountered fetching external documents or errors that occur within the dialog execution thread. In this case, a namelist containing the items "dialog.exit.status" and "dialog.exit.description" is returned with the event to inform the client of the failure and the failure reason. The values of these items are defined within this package and the MSML Core Package. Information from the failed dialog may be returned as additional namelist items.

Attributes:

target: an identifier of a connection or a conference that will interact with the dialog. The identifier must not contain wildcards. Mandatory.

src: the URL of the dialog description. MUST NOT be used if the MSML dialog description is inline. Otherwise, an error (422) will result and MSML document execution will stop.

type: a MIME type that identifies the type of language used to describe the dialog. application/moml+xml and application/vxml+xml are used to identify MSML dialogs and VoiceXML [n5] respectively. Mandatory.

name: an instance name for the dialog. If the attribute is not present, the media server will assign an identifier to the dialog. If the attribute is present but the name is already associated with the target, an error (431) will result and MSML document execution will stop. Any results that a dialog generates will be correlated to its identifier.

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML element is returned in an error response. Therefore, the value of all "mark" attributes within an MSML document should be unique.

The following sections show examples of initiating an external MSML dialog, an inline embedded MSML dialog, and an MSML-initiated VoiceXML dialog.

The following example starts an MSML dialog on a connection.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <dialogstart target="conn:abcd1234"
         type="application/moml+xml"
         name="sample"
         src="http://server.example.com/scripts/foo.moml"/>
 </msml>
```

The following example starts an inline embedded MSML dialog on a connection.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
  <dialogstart target="conn:abcd1234" name="sample">
     <play>
        <audio uri="file://clip1.wav"/>
        <audio uri="http://host1/clip2.wav"/>
        <tts uri="http://host2/text.ssml"/>
        <var type="date" subtype="mdy" value="20030601"/>
     </play>
     <send target="source"
           event="done"
           namelist="play.amt play.end"/>
   </dialogstart>
</msml>
```

The following example starts a VoiceXML dialog on a connection.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <dialogstart target="conn:abcd1234"
       type="application/vxml+xml"
       name="sample"
       src="http://server.example.com/scripts/foo.vxml"/>
</msml>
```

If this dialog fails once its execution thread had begun, for example, the fetch of the VoiceXML document failed, an example of the event that would be returned would be:

```
<?xml version="1.0" encoding="UTF-8"?>
<event name="msml.dialog.exit"
       id="conn:abcd1234/dialog:sample">
   <name>dialog.exit.status</name>
   <value>423</value>
   <name>dialog.exit.description</name>
   <value>External document fetch error</value>
</event>
```

9.6.2. <dialogend>

Dialog end is used to terminate an MSML dialog created through <dialogstart> before it completes of its own accord. The operation of <dialogend> depends on the dialog language being used by the executing context. When that context is VoiceXML, a "connection.disconnected" event will be thrown to the VoiceXML application. When that context is MSML dialog, a "terminate" event will be sent to the MSML core context.

<dialogend> allows the executing dialog the opportunity to gracefully complete before generating a "msml.dialog.exit" event. Dialog results may be returned and will be contained as a namelist to that event.

Attributes:

id: the identifier of a dialog. Mandatory.

mark: a token that can be used to identify execution progress in the case of errors. The value of the mark attribute from the last successfully executed MSML dialog element is returned in an error response. Therefore, the value of all "mark" attributes within an MSML document should be unique.

For example, if the dialog from the previous example was still executing, the following would terminate the dialog and generate an "msml.dialog.exit" event.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <dialogend id="conn:abcd1234/dialog:sample"/>
</msml>
```

9.6.3. <send>

The <send> element sends an event and optional namelist to the recipient identified by the target attribute. Event names are defined by the recipient. In the case where the recipient is an MSML dialog group or primitive, the events are defined within this document. Other recipients MAY use names that are suitable for their environment.

The "target" attribute specifies the recipient of the event. Recipients MAY be other MSML dialog primitives or groups executing within the object, the object itself, or the environment that invoked the MSML dialog. Sending events to media primitives or groups is supported by the MSML Dialog Group Package. Any target that is

unknown within the object is assumed to be destined to the external environment. By convention, the string "source" SHOULD used to address that environment, but any target name distinct from the MSML dialog namespace MAY be used.

Attributes:

event: the name of an event. Mandatory.

target: the recipient of the event. The recipient MUST be a MSML dialog primitive, the currently executing group, or the MSML dialog environment. A primitive is specified by a primitive type, optionally appended by a period '.' followed by the identifier of a primitive. Identifiers are only needed when more than one primitive of the same type exists in the object. The executing group is specified using the token "group". The environment is specified using the token "source", optionally appended by a period '.' followed by any environment specific target. Mandatory.

namelist: a list of zero or more shadow variables that are included with the event.

9.6.4. <exit>

The <exit> element causes execution of the MSML dialog to terminate.

Attributes:

namelist: a list of one or more shadow variables that MAY optionally be sent to the context that invoked the MSML Dialog object.

9.6.5. <disconnect>

The <disconnect> element is similar to <exit> but has the additional semantics of indicating to the context that invoked the MSML dialog that it should disconnect from a media server, the media stream associated with the object. The method of disconnection depends upon how the media stream was initially established. If SIP was used, a <disconnect> would cause a media server to issue a BYE request. The request would be sent for the SIP dialog associated with media session on which the MSML dialog was operating.

Attributes:

namelist: a list of one or more shadow variables that MAY optionally be sent to the context that invoked the MSML dialog object.

9.7. MSML Dialog Base Package

The MSML Dialog Base Package defines a required set of base functionality for the media server. It supports individual media primitives, such as playing an announcement or collection digits, as well as composite operations such as play and collect. When this package is used in conjunction with the MSML Dialog Group Package, the event-based mechanism is used to control primitives. This package may also be used in conjunction with the MSML Speech Package to extend the functionality of prompts to include TTS and user input collection to include ASR.

In the following sections, subsections of a primitive define child elements of that primitive and are not themselves considered primitives. They do not receive events or populate shadow variables.

9.7.1. <play>

Play is used to generate an audio or video stream. It MUST play in sequence the media created by the child media elements <audio>, <video>, <media>, <tts>, and <var>. When the play stops, either because the terminate event is received or all media generation has completed, the <playexit> element, if present, is executed. At least one media generation element must be present.

Play supports two states: generate and suspend. Media generation occurs in the generate state and is suspended in the suspend state. Once in the suspend state, media generation continues upon receiving the generate event. The default initial state is generate.

Audio MAY be generated in different languages by specifying the xml:lang attribute for <play> and/or the child elements of <play>. The language is inherited by the child elements, but each child MAY specify its own language. Except for physical audio clips, it is an error if a language is specified but the media server cannot render the audio in the requested language.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the play primitive.

interval: specifies the delay between stopping one iteration and beginning another. The attribute has no effect if iterate is not also specified. Default is no interval.

iterate: specifies the number of times the media specified by the child media elements should be played. Each iteration is a complete play of each of the child media elements in document order. Defaults to once '1'.

initial: defines the initial state for the play element. Default is "generate".

maxtime: defines the maximum allowed time for the <play> to complete.

barge: defines whether or not audio announcements may be interrupted by DTMF detection during play-out. The DTMF digit barging the announcement is stored in the digit buffer. Valid values for barge are "true" or "false", and the attribute is mandatory. When barge is applied to a conference target, DTMF digit detected from any conference participant MUST terminate the announcement.

cleardb: defines whether or not the digit buffer is cleared, prior to starting the announcement. Valid values for cleardb are "true" or "false", and the attribute is mandatory.

offset: defines an offset, measured in units of time, where the <play> is to begin media generation. Offset is only valid when all child media elements are <audio>.

skip: an amount, expressed in time, that will be used to skip through the media when "forward" and "backward" events are received. Default is 3 s (three seconds).

xml:lang: specifies the language to use for content that can be rendered in different languages.

Events:

The following describes input events to the media primitive object. The MSML Dialog Group Package allows an event exchange mechanism between primitives.

pause: causes the play to enter the suspend state.

resume: causes play to enter the generate state.

forward: skips forward through the media. Only has effect when all child media elements are <audio>.

backward: skips backward through the media. Only has effect when all child media elements are <audio>.

restart: skips to the beginning of the media. Only has effect when all child media elements are <audio>.

toggle-state: causes the suspend / generate state to toggle.

terminate: terminates the play and assigns values to the shadow variables.

Shadow Variables:

play.amt: identifies the length of time for which media was generated before the play was stopped. This does not include time that may have elapsed while the play was in the suspend state.

play.end: contains the event that caused the play to stop. When the play stops because all media generation has completed, end is assigned the value "play.complete".

Note: Attributes barge and cleardb provide a simplified mechanism for controlling play operations with implicit DTMF without the use of <group> and event exchange mechanism. When using the <play> element within the group framework and barge is specified, detection of barge condition generates an implicit terminate event to the play primitive.

The following sections describe the child elements of <play>.

9.7.1.1. <audio>

The <audio> element identifies prerecorded audio to play. Local URI references may resolve to a single physical audio clip, a logical clip, or a provisioned sequence of clips (physical or logical). A logical clip is one that can be rendered differently based on the language attribute. Logical clips are provisioned for each of the languages that a media server supports. Remote URI references are resolved according to the capabilities of the remote server.

Attributes:

uri: identifies the location of the audio to be played. The file and http schemes are supported. Mandatory.

format: defines the encoding and file type of the audio resource. The format attribute is defined as a string type of form "audio/<filetype>;codecs=<codec>". The keyword 'audio' identifies an audio content. The codecs field identifies the audio file's codec to be used for decoding the audio content. If format attribute is not specified, the filetype MUST be determined from the URI and the codec information MUST be determined from the media resource.

audiosamplerate: identifies audio sample rate in kHz. If not specified, the sample rate SHOULD be determined from the media resource.

audiosamplesize: identifies audio sample size in bits. If not specified, the sample size SHOULD be determined from the media resource.

iterate: specifies the number of times the audio is to be played. Defaults to once '1'.

xml:lang: specifies the language to use when the URI identifies a logical clip, either directly, or as part of a sequence.

9.7.1.2. <video>

The <video> element identifies prerecorded multimedia to play. Contents identified by the URI attribute may contain audio only, video only, or both audio and video. The media server SHOULD attempt to play both audio and video from the identified URI, if both are available in the content.

Attributes:

uri: identifies the location of the video or multimedia to be played. The file and http schemes are supported. Mandatory.

format: defines the encoding and file type of the video or multimedia resource. The format attribute is defined as a string type of form "video/<filetype>;codecs=<codecx>,<codecy>". The keyword 'video' identifies video-only media or media containing audio and video. The "codecs" field identifies the audio and/or video codecs to be used for decoding the file content, where the order of the codec values is not significant. In the event of audio and video content, using 'video' keyword, the codecs=<codecx>,<codecy> field MAY be used to identify the audio codec and the video codec. If not specified, the codec information SHOULD be determined from the media file.

audiosamplerate: identifies audio sample rate in kHz. If not specified, the sample rate SHOULD be determined from the media file.

audiosamplesize: identifies audio sample size in bits. If not specified, the sample size SHOULD be determined from the media file.

codecconfig: identifies an optional special instruction string for codec configuration. Default is to send no special configuration string to the codec.

profile: identifies a video profile name specific to the codec. If not specified, default video profile of the codec SHOULD be selected.

level: identifies a video profile level to the codec. Default is to send no profile information to the codec and allow the codec to select an internal default.

imagewidth: identifies the width of video image in pixels. Default is to use image width information from media file.

imageheight: identifies the height of video image in pixels. Default is to use image height information from media file.

maxbitrate: identifies the bitrate of the video signal in kbps. Default is to use maximum bitrate information from the media file.

framerate: identifies the video frame rate in frames per second. Default is to use frame rate information from the media file.

iterate: specifies the number of times the media content is to be played. Defaults to once '1'.

9.7.1.3. <media>

The <media> element identifies multimedia content for play. All content of the <media> element MUST start to play concurrently. This element may be used to generate a multimedia stream from two independent media resources, one identifying audio and the other identifying video.

The <media> element MUST contain at least one child element. Valid child elements of <media> are <audio> and <video>, as described earlier. <media> element MUST contain at most one <audio> element or at most one <video> element.

9.7.1.4. <var>

The <var> element specifies the generation of audio from a variable using prerecorded audio segments. A variable represents a semantic concept (such as date or number) and dynamically produces the appropriate speech.

Prerecorded audio allows an application vendor or service provider to choose the exact voice for their audio and therefore completely control the "sound and feel" of the service provided to end users. It provides very high audio quality and allows the variables to blend seamlessly into the surrounding audio segments.

Text to speech (TTS) using Speech Synthesis Markup Language (SSML) [n11] may also be used to render variables, but may not provide as good quality, or allow as complete control of the "sound and feel" or user experience. TTS is normally used for reading text such as emails and for very large vocabularies such as stock names. TTS results in a very clear difference between the variables and the surrounding audio segments. (See MSML Dialog Speech Package.)

Attributes:

type: specifies the type of variable. Mandatory. Variable type must be one of "date", "digits", "duration", "month", "money", "number", "silence", "time", or "weekday".

subtype: specifies an optional clarification of type. Specific values depend upon the type.

value: text that should be rendered appropriate to the type and subtype attributes. Mandatory.

xml:lang: specifies the language to use when rendering the variable.

9.7.1.5. <playexit>

The <playexit> element MUST be invoked when generation of all content of the <play> has come to completion. The contents of this element MAY be used to send events.

Attributes:

none

9.7.2. <dtmfgen>

DTMF generator originates one or more DTMF digits in sequence.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the dtmfgen primitive.

digits: a string of characters from the alphabet "0-9a-d#*" that correspond to a sequence of DTMF tones. Mandatory.

level: used to define the power level for which the tones will be generated. Expressed in dBm0 in a range of 0 to -96 dBm0. Larger negative values express lower power levels. Note that values lower than -55 dBm0 will be rejected by most receivers (TR-TSY-000181, ITU-T Q.24A). Default is -6 dBm0.

dur: the duration in milliseconds for which each tone should be generated. Implementations may round the value if they only support discrete durations. Default is 100 ms.

interval: the duration in milliseconds of a silence interval following each generated tone. Implementations may round the value if they only support discrete durations. Default is 100 ms.

Events:

terminate: terminates DTMF generation and assigns values to the

shadow variables.

Shadow Variables:

dtmfgen.end: contains the event that caused DTMF generation to stop.

The following sections describe the child elements of <dtmfgen>.

9.7.2.1. <dtmfgenexit>

The <dtmfgenexit> element MUST be invoked when the DTMF generation operation completes or is terminated as a result of receiving the terminate event. The <dtmfgenexit> element MAY be used to send events when the DTMF generation has completed.

Attributes:

none

9.7.3. <tonegen>

Tone generator allows customized tone generation. A sequence of varying tones with optional silence intervals can be composed using the <tonegen> element. Child elements of <tonegen>, namely <tone> and <silence>, specify a single tone or sequence of tones.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the tonegen primitive.

iterate: A numeric value specifying the total number of iterations. A value of 'forever' represents infinite repetitions. Optional. Default is 1.

Events:

terminate: terminates tone generation and assigns values to the shadow variables.

Shadow Variables:

tonegen.end: contains the event that caused tone generation to stop.

The following sections describe the child elements of <tonegen>.

9.7.3.1. <tone>

The <tone> element specifies a single tone with an optional silence interval. The tone specification consists of two tone frequencies, their attenuation values, a duration of the tone, and the number of times to repeat the tone.

Attributes:

duration: time duration or length of the individual tone, specified in "ms" or "s" in increments of 10 ms. A value of 0 represents an infinite duration. Mandatory.

iterate: specifies the number of times to execute the contents of <tone> element. A value of 'forever' represents infinite repetitions. Optional. Default is 1.

Events:

none

Child Elements:

The child elements of <tone> element specify a single tone and an optional silence interval to be inserted at the end of tone generation. A tone is defined by <tone1> and <tone2> elements. Each <tone> element MUST contain at least one of <tone1> or <tone2>, or MAY contain <tone1> and <tone2> exactly once.

<tone1>

Attributes:

freq: specifies the frequency of the first tone in "Hz", ranging from 0 to 3999 Hz. Mandatory.

atten: specifies the attenuation level expressed in dBm0, ranging from 0 to -96 dBm0. Mandatory.

<tone2>

Attributes:

freq: specifies the frequency of the second tone in "Hz", ranging from 0 to 3999 Hz. Mandatory.

atten: specifies the attenuation level expressed in dBm0, ranging from 0 to -96 dBm0. Mandatory.

<silence> - Refer to the silence element definition below.

9.7.3.2. <silence>

The <silence> element inserts a silence interval as optional content of <tonegen> or <tone> elements.

Attributes:

duration: specifies the amount of silence interval in "ms" or "s", in increments of 10ms. Mandatory.

Events:

none

9.7.3.3. <tonegenexit>

The <tonegenexit> element MUST be invoked when the tone generation operation completes or is terminated as a result of receiving the terminate event. The <tonegenexit> element MAY be used to send events when the tone generation has completed.

Attributes:

none

9.7.4. <record>

Record creates a recording. Similar to play, <record> supports two states: create and suspend. Received media becomes part of the recording when <record> is in the create state and is discarded when it is in the suspend state.

Recording MUST be terminated when a terminate event is received or when a nospeech event is received and no audio has yet been recorded. <record> differentiates different types of terminate events.

An optional <play> element MAY be specified as a child element of <record>. This mechanism provides a complete play-record operation, where the prompts specified within the <play> element are played in advance of start of recording.

Note: Attributes prespeech, postspeech, and termkey provide a simplified mechanism for controlling record operations using implicit DTMF and VAD, without the use of <group> and event exchange mechanism.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the record primitive.

append: a boolean that defines whether the recording is allowed to be appended to an existing file if dest already exists. Default is "false". The attribute is ignored if the scheme is http.

dest: the destination for the recording, which will contain either audio only, video only, or both audio and video depending on the stream(s) being recorded. Recording MAY be either local or external based upon the attribute value. File and http schemes are supported.

audiodest: the destination for the audio-only recording. Recording MAY be either local or external based upon the attribute value. All combinations of dest, audiodest, and videodest are valid. File and http schemes are supported.

videodest: the destination for the video-only recording. Recording MAY be either local or external based upon the attribute value. All combinations of dest, audiodest, and videodest are valid. File and http schemes are supported.

format: defines the encoding and file type of the recording. The format attribute is defined as a string type of form "audio|video/filetype;codecs=x,y". The keyword 'audio' identifies an audio only recording, while the keyword 'video' identifies video-only recording or an audio plus video recording. The codecs field identifies the audio and/or video codecs to be used for the recording, where the order of the codec values is not significant. In the event of audio and video recording, using 'video' keyword, the codecs=x,y field MAY be used to identify the audio codec and the video codec. Mandatory.

codecconfig: identifies an optional special instruction string for codec configuration. Default is to send no special configuration string to the codec.

audiosamplerate: identifies audio sample rate in kHz. If not specified, the sample rate SHOULD be determined from the media source.

audiosamplesize: identifies audio sample size in bits. If not specified, the sample size SHOULD be determined from the media source.

profile: identifies a video profile name specific to the codec. If not specified, default video profile of the codec SHOULD be selected for the recording.

level: identifies a video profile level to the codec. Default is to send no profile information to the codec and allow the codec to select an internal default.

imagewidth: identifies the width of video image in pixels. Default is to use image width information from the media source.

imageheight: identifies the height of video image in pixels. Default is to use image height information from the media source.

maxbitrate: identifies the bitrate of the video signal in kbps. Default is to use maximum bitrate information from the media source.

framerate: identifies the video frame rate in frames per second. Default is to use frame rate information from the media source.

initial: defines the initial state for the record element. Default is "create", which starts the recording as soon as the <record> element is executed. The "initial" attribute is applicable only when <record> is used within the <group> structure.

maxtime: defines the maximum length of the recording in units of time. Mandatory.

prespeech: defines a timer value, in seconds, for detection of absence of audio energy at the start of the record operation. If no audio energy is detection for the amount of time specified by prespeech, the recording is terminated. Default is 0 s, which does not activate the prespeech timer.

postspeech: defines a timer value, in seconds, for detection of absence of audio energy while the recoding is in progress. During an in progress recording, if absence of audio energy is detected as specified by the postspeech timer, the recording is terminated. Default is 0 s, which disables the ability to terminate a recording due to postspeech silence.

termkey: defines a single DTMF key that, when detected, terminates the recording. Absence of this attribute prevents the recording from being terminated due to detection of DTMF digits. When termkey is specified, the detected DTMF digit terminates the recording and the DTMF digit is not entered in the digit buffer.

Events:

The following describes input events to the media primitive object. The MSML Dialog Group Package allows an event exchange mechanism between primitives.

pause: causes the record to enter the suspend state. Received media is discarded.

resume: causes the record to resume if it was suspended. It has no effect otherwise.

toggle-state: causes the suspend / create state to toggle.

terminate: terminates the recording and assigns values to the shadow variables.

terminate.cancelled: terminates the recording and assigns values to the shadow variables. If the dest attribute used the file scheme, the local recording is deleted. Applications are responsible for removing external files created using the http scheme.

terminate.finalsilence: terminates the recording and assigns values to the shadow variables. If the dest attribute used the file scheme, the final silence is removed from the recording.

nospeech: terminates the recording and assigns values to the shadow variables if it is received and no recording has yet been created. The "nospeech" event is ignored if audio has already been recorded.

Shadow Variables:

record.len: the actual length of the recording measured in units of time. This does not include time that may have elapsed while the record was in the suspend state.

record.end: contains the event that caused the record to terminate. When the record terminates because maxtime is exceeded, end is assigned the value "record.complete.maxlength".

record.recordid: contains the value of the "dest" attribute, if supplied, otherwise contains a media server assigned record identifier.

Record termination due to prespeech silence results in assigned value of "record.failed.prespeech"

Record termination due to postspeech silence results in assigned value of "record.complete.postspeech"

Record termination due to DTMF detection results in assigned value of "record.complete.termkey"

The following sections describe the child elements of <record>.

9.7.4.1. <play>

The optional <play> element as a child element of <record> allows a prompt to be played prior to start of recording. The record operation starts at the end of the play sequence or if the play is barged by DTMF, assuming that barge=true is specified for <play>. For a complete description, refer to <play> element.

9.7.4.2. <tonegen>

The optional <tonegen> element as a child element of <record> allows a tone or sequence of tones to be played prior to start of recording. The record operation starts at the end of the tone generation. For a complete description, refer to <tonegen> element.

9.7.4.3. <recordexit>

The <recordexit> element MUST be invoked when the record operation completes or when the recording is terminated as a result of receiving the terminate event. The <recordexit> element MAY be used to send events when the recording has completed.

Attributes:

none

9.7.5. <dtmf> or <collect>

DTMF input fulfills several roles within MSML dialogs. It is used to trigger events that will affect the media processing operation of other primitives. It is also used to collect DTMF digits from a media stream that are to be reported back to the user of MSML dialog. Often DTMF detection is used for both purposes. Barge is the most common example, where a prompt is stopped based upon DTMF input but more digits may remain to be collected.

DTMF detection supports multiple simultaneous recognition patterns. Different patterns can be used to trigger sending different events in order to implement DTMF controls. Alternatively, one pattern may be

used to represent a collection and another pattern, a substring of the first, used as a barge indication.

An optional <play> element MAY be specified as a child element of <dtmf> or <collect>. This mechanism provides a complete play-collect operation, where the prompt(s) specified within the <play> element are played in advance of DTMF digit collection.

Note that all patterns share the same digit collection buffer, inter-digit timing, a single <nomatch> element, and a single <noinput> element. As such, multiple patterns may not be suitable to support simultaneous collections for different purposes. When this is required, separate <dtmf> elements should be used instead.

<dtmf> terminates if any of the <pattern>, <noinput>, or <nomatch> elements are matched the maximum number of times that they are allowed. The number of times they may match may be specified as an attribute of <dtmf> or of the individual child elements.

Element identifier <dtmf> is equivalent to <collect>. However, <collect> is the preferred name. MSML clients SHOULD use <collect>, while MSML servers SHOULD support both.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to this primitive.

cleardb: a boolean indication of whether the buffer for digit collection should be cleared of any collected digits when the element is instantiated. If set to false, any digits currently in the buffer MUST be immediately compared against the pattern elements.

fdt: defines the first-digit timer value. The first-digit timer is started when DTMF detection is initially invoked. If no DTMF digits are detected during this initial interval, the <noinput> element MUST be invoked. Optional, default is 0 s (wait forever for the first digit).

idt: defines the inter-digit timer to be used when digits are being collected. When specified, the timer is started when the first digit is detected and restarted on each subsequent digit. Timer expiration is applied to all patterns. After that, if any patterns remain active and a nomatch element is specified, the nomatch is executed and DTMF input MUST terminate. The idt attribute should only be used when digit collection is being performed. Optional, default is 4 s.

edt: defines the extra-digit timer value. Specifies the length of time the media server MUST wait after a match to detect a termination key, if one is specified by the <pattern> element. Optional, default is 4 s.

starttimer: boolean value that defines whether the first digit timer (fdt) is started initially. When set to false, the starttimer event must be received for it to start. Default is "false".

iterate: specifies the number of times the <pattern>, <noinput>, and <nomatch> elements may be executed unless those elements specify differently. The value "forever" MAY be used to indicate that these may be executed any number of times. Default is once '1'.

ldd: defines the minimum duration for a digit to be held in order for it to be detected as a long DTMF digit. A long DTMF digit event MUST be treated as a single DTMF event, and MUST contain an extra character 'L' at the end to be distinguished from the other regular digit events. For example, "#L" and "#" are different DTMF events. Optional, default of 0 s. A value of 0 s disables long DTMF digit detection and reporting. Attribute value is an integer with a valid range from 100 ms to 100 s (units MUST be supplied).

Events:

The following describes input events to the media primitive object. The MSML Dialog Group Package allows an event exchange mechanism between primitives.

starttimer: starts the first digit timer (fdt) if it has not already been started. Has no effect otherwise.

terminate: terminates the DTMF input and assigns values to the shadow variables.

Shadow Variables:

dtmf.digits: the string of DTMF digits that have been received (the contents of the digit buffer).

dtmf.len: the number of digits in the digit buffer.

dtmf.last: the last digit in the digit buffer.

dtmf.end: contains the event that caused the <dtmf> to terminate or is assigned one of "dtmf.match", "dtmf.noinput", or "dtmf.nomatch" depending upon which of the corresponding elements reached its maximum.

The following sections describe the child elements of <dtmf> or <collect>.

9.7.5.1. <play>

The optional <play> element as a child element of <dtmf> or <collect> allows a prompt to be played prior to DTMF digit collection. DTMF digit collection starts at the end of the play sequence or if the play is barged by DTMF, assuming that barge=true is specified for <play>. For a complete description, refer to <play> element.

9.7.5.2. <pattern>

The <pattern> element describes one or more DTMF digits that are to be recognized. When the pattern is matched, the child elements MUST be executed.

Attributes:

digits: the digit pattern that should be matched. Mandatory.

format: an enumerated value that defines the format used to express the digit pattern. The format may be "mgcp" or "megaco" for patterns expressed as a digit map from those specifications, or as one of the simple built-in formats defined within this specification. Currently, a single built-in format "moml+digits" is defined that allows a match based on either one or more specific digits, or based upon a specific length specification with an optional return key. "moml+digits" is the default.

iterate: specifies the number of times the <pattern> may be matched. The value "forever" may be used to indicate that <pattern> may be matched any number of times. This value overrides any specified in <dtmf>. Default is once ’1’.

9.7.5.3. <detect>

The contents of the <detect> element MUST be executed whenever any DTMF is first detected. It MUST be matched at most once.

Attributes:

none

9.7.5.4. <noinput>

The <noinput> element is used when DTMF is being collected. Children of the <noinput> element MUST be executed when DTMF has not been detected and the first digit timeout occurs.

Attributes:

iterate: specifies the number of times the <noinput> may be triggered. The value "forever" may be used to indicate that <noinput> may be triggered any number of times. This value overrides any specified in <dtmf>. Default is once '1'.

9.7.5.5. <nomatch>

The <nomatch> element is used when DTMF is being collected. Children of the <nomatch> element MUST be executed when it is determined that none of the individual patterns can be matched.

Attributes:

iterate: specifies the number of times the <nomatch> may be triggered. The value "forever" may be used to indicate that <nomatch> may be triggered any number of times. This value overrides any specified in <dtmf>. Default is once '1'.

9.7.5.6. <dtmfexit>

The <dtmfexit> element MUST be invoked when the dtmf input completes because one of <pattern>, <noinput>, or <nomatch> occurred its maximum number of times.

Attributes:

None

9.7.6. <moml>

The root element <moml> MUST be used when the document is a stand-alone MSML dialog, where the invoking application media type indicates 'application/moml+xml'. Additionally, for backwards compatibility, the <moml> element MUST be used within <dialogstart>, which contains an inline embedded MSML dialog.

Valid contents of <moml> are all elements described within this MSML Dialog Base Package.

Attributes:

version: "1.0" Mandatory.

id: an identifier unique to this object. Events returned from MSML dialog (the "target" attribute of a <send> is equal to "source") will be correlated with this identifier. Mandatory.

Events:

terminate: terminates the MOML context. A terminate event gets sent to the currently executing <group> or primitive.

9.8. MSML Dialog Group Package

The group package defines a single control flow construct that specifies concurrent execution. Primitives are composed for concurrent execution by placing them within a <group> element. Groups define how media flows between multiple concurrently executing primitives. They have one or more inputs and one or more outputs. A <group> represents the declaration of a complex media processing operation. The event interaction between primitives (see the following subsection) is defined within the context of one or more groups. However groups themselves do not scope events, they simply define that primitives are concurrently executing and a primitive must be executing in order to receive an event.

Placing primitives within a group structure is an optional feature of this specification. It allows for complex services to created using the event exchange mechanism between the primitives. For simpler services, such as play/collect or play/record, the use of group mechanism is not necessary. MSML Dialog Group Package is dependent on the MSML Dialog Base Package.

Groups may also be used to describe media objects that transform a media stream while optionally allowing application or user control of the transformation. For example, a gain control could be defined that responds to user speech or DTMF input. In this case, a recognition primitive would send events to a gain control primitive.

Groups have one attribute that defines the media flow within them. They also have a dimension that defines how many media inputs and outputs they have. Currently, dimensions of 1 and 2 are supported based upon the group topology. These correspond to a group with one input and one output and a group with two inputs and two outputs.

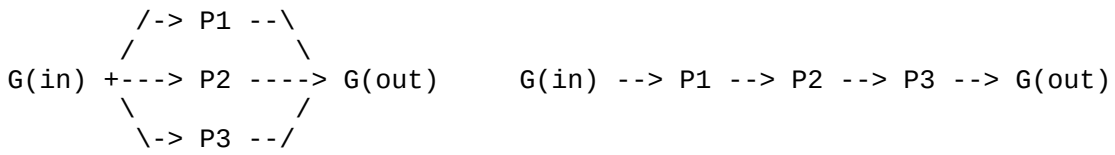
Media flow to and from the primitives within the group is based upon a topology attribute of the <group> element. The topology attribute defines a topology schema and implies the group dimension.

There are several common ways in which primitives are often connected together. A schema provides a convenient template that can be applied to multiple primitives without having to define all of the individual media relationships. The following two schemas are initially defined for one-dimensional groups:

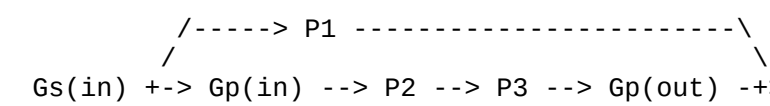
- parallel: specifies that media sent to the group is sent to every primitive that has an input. The group bridges the output from every primitive that has an output into a single common group output.

- serial: specifies that the first primitive listed in the group receives the media sent to the group. Its output is to be connected to the input of the next primitive defined within the group and so on until the last primitive within the group becomes the group output.

Groups with these topologies are shown in the two diagrams below. The group on the left has a parallel topology and that on the right has a serial topology.



More complex media flows MAY be created by nesting groups of serial and parallel topologies within each other. For example, the diagram below has a group with a serial topology nested within a star topology.



This combination could be used to create record operation where DTMF was to be clamped from the recording itself, but a DTMF key press is still used to stop the recording. In this case, P1 would be a DTMF recognizer, P2 would be a clamp primitive, and P3 a recorder as shown by the following example. This example omits child elements and attributes not concerned with the core concept. The following section discusses sending events, and the details of each of the primitives are found in section 4.

```
<group topology="parallel">
   <dtmf/>
   <group topology="serial">
      <clamp/>
      <record/>
   </group>
</group>
```

A single schema, "fullduplex", is defined for a two-dimensional group. A full-duplex two-dimensional group has exactly two immediate children. Those children may be primitives or other one-dimensional groups. A "fullduplex" group must only be used as the top-most group and must not be nested. Each primitive (P1) and group (G2) becomes half of the full-duplex group as shown in the diagram below.

```
G-A(in1)  +-> G2 --> G-B(out1)

G-A(out2) <-- P1 <-+ G-B(in2)
```

Full-duplex groups are symmetrical when both halves are the same. They are asymmetrical when they differ. Asymmetric groups need to have a name associated with each side. The left side is defined as the input of the first child of the full-duplex group combined with the output of the second child. The right side is reverse. These sides were labeled A and B respectively in the preceding diagram.

An example of a full-duplex group is the user operated gain control mentioned at the beginning of this subsection. The gain should operate on the audio that a user hears, but the gain is controlled by recognizing things such as DTMF or spoken commands in media that the user originates. The following shows the XML tag grouping that would accomplish this and corresponds to the media flow shown in the diagram above. If the user’s audio is not required for anything other than control of the gain, then the <relay> is not required and the internal group could be omitted. A complete XML description for this is included in the examples section.

```
<group topology="fullduplex">
   <group topology="parallel">
      <dtmf/>
      <relay/>
   </group>
   <gain/>
</group>
```

Primitives within a group MUST begin concurrently but MAY finish asynchronously based upon events that they receive or their task completes. A group MUST terminate when all of the primitives within

it have completed. If the group contains a <groupexit> element, then the contents of that element MUST be executed as part of group termination.

A group itself MAY receive a terminate event requesting termination. A terminate event sent to the group causes a terminate event to be sent to each of its currently active primitives. The <groupexit> element is not executed until all primitives have processed their respective terminate events.

9.8.1. <group>

The <group> element allows the contained primitives to be executed concurrently.

Attributes:

topology: specifies a schema that defines the flow of media within the group. Three schemas are initially defined. "fullduplex" is specified for use with two-dimensional groups. "parallel" and "serial" are for use with one-dimensional groups. The definitions of these topologies are in section 9.8. Mandatory.

id: identifies the name of the group. Mandatory when groups are nested.

Events:

terminate: causes a terminate event to be sent to each element contained within the group.

9.8.2. <groupexit>

The <groupexit> element allows events to be sent when group processing completes. Group processing completes when all contained primitives terminate.

Attributes:

none

Events:

none

9.9. MSML Dialog Transform Package

The MSML Dialog Transform Package gathers together the simple primitives that work as filters on half-duplex media streams.

9.9.1. <vad>

Voice activity detection (VAD) is used to detect voice and silence when speech recognition is not required. Similar to both speech and DTMF, a VAD has different media conditions that it can match. Those conditions can be qualified by a minimum length of time that is required for them to be considered recognized.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the vad primitive.

starttimer: boolean value that defines whether the timer is started to allow recognition of the initial condition (voice, silence). When set to false, the starttimer event must be received in order for the initial condition to be recognized. The timer does not affect recognition of the transition conditions. Default is "false".

Events:

starttimer: starts the timer to allow recognition of the initial condition if it has not already been started. Has no effect otherwise.

terminate: terminates voice activity detection.

Shadow Variables:

none

The following sections describe the child elements of <vad>.

9.9.1.1. <voice>, <silence>, <tvoice>, <tsilence>

Each child element corresponds to a condition that a VAD can detect. The first two detect when voice or silence has been initially present for a minimum length of time since the VAD was started. The second two require that a transition to the voice or silence condition first occur.

Attributes:

len: the length of time the condition must persist in order to be recognized. Mandatory. In the case of <tvoice> and <tsilence>, the length of time applies only to the final recognized condition.

sen: the maximum length of time the condition not being detected may occur without causing the detector to begin measuring that condition.

9.9.2. <gain>

Gain MAY be used to adjust of the gain of a media stream by a specific amount. Application of <gain> removes any previous connection AGC setting used by the <agc> element.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the gain primitive.

incr: an increment, expressed in dB, that will be used to adjust the gain when "louder" and "softer" events are received. Default is 3 dB.

amt: a specific gain to apply specified in dB. Mandatory.

Events:

mute: self-explanatory.

unmute: self-explanatory.

reset: sets the gain to zero dB.

louder: makes the audio on a stream louder.

softer: makes the audio on a stream quieter.

amt: sets the gain to the specified value between -96 dB and 96 dB.

9.9.3. <agc>

Automatic gain control MAY be used to have a media server automatically adjust the gain of a media stream. Application of <agc> removes any previous connection gain setting used by the <gain> element.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the gain primitive.

tgtlvl: the desired target level for AGC, specified in dBm0 with a valid range of -40 to 0. Mandatory.

maxgain: an optional attribute used to specify the maximum gain that AGC will apply, specified in dBm0 with a valid range of 0 to 40, with a default of 10.

Events:

mute: self-explanatory.

unmute: self-explanatory.

9.9.4. <gate>

The <gate> element is a simple filter that will pass or halt media, regardless of the format of the media stream, based on the events it receives. <gate> shares the same mute and unmute events for compatibility with the gain primitives <gain> and <agc>.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the gate primitive.

initial: the values "pass" and "halt" define whether media is initially allowed to pass. Default is to pass.

Events:

mute: halts media flow through the primitive.

unmute: allows media to pass through the primitive.

9.9.5. <clamp>

This element MAY be used to filter DTMF tones from a media stream. Media other than DTMF tones is passed unchanged.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the clamp primitive.

Events:

none.

9.9.6. <relay>

This element is a simple primitive that copies its input to its output.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the relay primitive.

Events:

none.

9.10. MSML Dialog Speech Package

The MSML speech package defines functionality that MAY be used for automatic speech recognition <speech> and extends the <play> primitive defined in the MSML Dialog Base Package to include speech synthesis. As such, this package depends on the MSML Dialog Base Package.

9.10.1. <speech>

The <speech> element activates grammars or user input rules associated with speech recognition. If multiple grammars are specified, all are activated. All active grammars share the same timers, recognition attributes, and <noinput> and <nomatch> elements. Each grammar may have its own <match> element.

<speech> terminates if any of the <grammar>, <noinput>, or <nomatch> elements are matched the maximum number of times that they are allowed. The number of times they may match may be specified as an attribute of <speech> or of the individual child elements.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the speech primitive.

noint: specifies a time period during which speech input must be started; otherwise, the associated <noinput> element is invoked.

norect: specifies a maximum time period during which speech must begin to be matched; otherwise, the associated <nomatch> element is invoked.

spcmplt: specifies the length of silence necessary after speech before a result will be finalized in the case where there is a complete match of an active grammar. Following the silence, the appropriate <match> element will be triggered if the result is above the confidence level. Otherwise, a <nomatch> element will be triggered.

spincmplt: specifies the length of silence necessary after speech before a result will be finalized in the case where there is a incomplete match of all active grammars. Following the silence, the <nomatch> element will be triggered.

confidence: the minimum confidence level that the recognizer must have to consider a recognition result as matching a grammar. Expressed as an integer between 1-100.

sens: specifies the sensitivity of the recognizer to determine whether speech is present. Lower sensitivity may be required for the recognizer to work well in the presence of high background noise or line echo.

starttimer: boolean value that defines whether the no input (noint) and no recognition (norect) are started initially. When set to false, the starttimer event must be received in order to start them. Default is "false".

iterate: specifies the number of times the <grammar>, <noinput>, and <nomatch> elements may be executed unless those elements specify differently. The value "forever" may be used to indicate that these may be executed any number of times. Default is once '1'.

Events:

sens: sets the sensitivity of the recognizer as described above.

starttimer: starts the no input (noint) and no recognition (norect) timers if they have not already been started. Has no effect otherwise.

terminate: terminates the speech input and assigns values to the shadow variables.

Shadow Variables:

speech.end: contains the event that caused the <speech> to terminate or is assigned one of "speech.match", "speech.noinput", or "speech.nomatch" depending upon which of the corresponding elements reached its maximum.

speech.results: contains the results of a matched grammar. The results are formatted using the Natural Language Semantics Markup Language (NLSML) [n4]. When this variable is referenced to return results, the results are returned as a separate MIME entity.

The following sections describe the child elements of <speech>.

9.10.1.1. <grammar>

The <grammar> element specifies and activates a speech grammar based on Speech Recognition Grammar Specification (SRGS) [n3] XML notation. Grammars may be referenced by a URI or defined inline. Child elements of <match> MUST be executed when the specified speech grammar is matched.

Attributes:

uri: specifies the location of an SRGS grammar when the grammar is not defined inline.

iterate: specifies the number of times the <grammar> may be matched. The value "forever" MAY be used to indicate that <grammar> may be matched any number of times. This value overrides any specified in <speech>. Default is once '1'.

9.10.1.2. <match>

<match> is a child of <grammar> and specifies the actions to take when the corresponding grammar is matched.

9.10.1.3. <noinput>

The <noinput> element is used when speech is being recognized. Children of the <noinput> element MUST be executed when speech has not been detected and the no input timeout (noint) occurs.

Attributes:

iterate: specifies the number of times the <noinput> may be triggered. The value "forever" may be used to indicate that <noinput> may be triggered any number of times. This value overrides any specified in <speech>. Default is once '1'.

9.10.1.4. <nomatch>

The <nomatch> element is used when speech is being recognized. Children of the <nomatch> element MUST be executed when it is determined that none of the active grammars will match.

Attributes:

iterate: specifies the maximum number of times the <nomatch> may be triggered. The value "forever" MAY be used to indicate that <nomatch> may be triggered any number of times. This value overrides any specified in <speech>. Default is once '1'.

9.10.1.5. <speechexit>

The <speechexit> element MUST be invoked when the speech input completes because one of <grammar>, <noinput>, or <nomatch> occurred its maximum number of times.

Attributes:

none

9.10.2. <play>

The <play> element, as defined in the MSML Dialog Base Package, is extended with a new child element for synthesizing speech. From an XML perspective, <tts> is a member of a media substitution group. See the schema at the end of this document for details.

The following sections describe the child elements of <play>.

9.10.2.1. <tts>

Contents of the <tts> element are rendered using text-to-speech services and must be compliant to the SSML specification [n11]. Element content MAY be plain text, contain the SSML <speak> element, or the uri attribute should identify the location of text to be rendered.

Attributes:

uri: identifies the location of the text to be rendered. The file and http schemes are supported.

iterate: specifies the number of times the text-to-speech block is to be rendered. Defaults to once '1'.

xml:lang: specifies the language to use when it is not explicitly specified as an attribute for <speak>.

9.11. MSML Dialog Fax Detection Package

The Fax Detection Package defines primitives that allow a media server to provide facsimile detection services.

9.11.1. <faxdetect>

Fax tone detection is used to detect the presence of the T.30 Calling Tone (CNG) or Called Station Identification (CED) tone in a media stream. Child elements of <faxdetectexit> MUST be executed when a CNG tone is detected.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the faxdetect primitive.

Events:

terminate: terminates fax tone detection and assigns values to the associated shadow variables.

Shadow Variables:

faxdetect.tone: A string that specifies the fax tone type detected by the media server. Values supported SHOULD include "CED", "CNG", or empty string. The empty string MUST be used if fax tone detection terminated before detection of a fax tone, resulting in execution of the <faxdetectexit> element.

faxdetect.end: A string value that specifies the reason for termination of <faxdetect>. Values supported SHOULD include "faxdetect.complete" (due to detection of CED or CNG tone), "faxdetect.failed.noresource" (failed due to lack of resources on the media server), "faxdetect.failed" (failed due to any other reason) "faxdetect.terminated" (terminated by <dialogend>), or undefined.

9.11.2. <faxdetectexit>

The <faxdetectexit> element MUST be invoked when fax detection, invoked via <faxdetect>, terminates. Child elements of <faxdetectexit>, <send> and <exit>, allow events to be reported by the media server.

Attributes:

none

9.12. MSML Dialog Fax Send/Receive Package

9.12.1. <faxsend>

The <faxsend> primitive provides the functionality of a calling fax terminal. This typically means sending a set of pages. However, it can also mean requesting the called terminal to send pages instead of, or in addition to, receiving pages. The fax images to send are defined by the <sendobj> elements, described below.

Requesting the called terminal to send pages happens when the <rxpoll> element is included as part of <faxsend>. This element may be included in addition to, or instead of, the <sendobj> element. One <sendobj> (at a minimum) or <rxpoll> element must be present. When both are present, a media server will first send pages and will then poll the other terminal, requesting pages.

Because fax is a distinct media type, the <faxsend> primitive is not expected to interact with other primitives. Rather, it will interact using fax protocols with a remote fax terminal (or gateway) and will send requested status events to its invoking environment. During fax operation, shadow variables are used to record the progress and parameters of the varying stages of fax operation.

Status events are requested by including one or more status request elements. These elements correspond to different stages or events in fax operation and cause predefined events to be sent to the invoking environment when they occur. Since the only recipient of these events is expected to be a fax control agent, requests are simplified by associating a predefined namelist of shadow variables with each event. This decision may be revisited to allowed tailored namelists based on further implementation experience. Status requests apply both to sending and polling operation.

Attributes:

lclid: the identifier that a media server uses to identify itself.

minspeed: the minimum acceptable speed to negotiate for the operation.

maxspeed: the maximum speed to negotiate for the operation. This attribute is primarily for testing purposes.

ecm: specifies whether Error Correction Mode (ECM) is allowed to be used if supported by the remote terminal. Defaults to "true".

Events:

terminate: terminates the fax send operation.

Shadow Variables:

fax.rmtid: the identifier of the remote fax terminal.

fax.rate: the negotiated speed for the operation.

fax.resolution: identifies the resolution of the image. Both metric- and inch-based resolutions are defined. Metric-based resolutions are 75x75, 150x150, 204x98, 204x196, 204x391, and 408x391. Inch-based resolutions are 200x200, 300x300, 400x400, and 600x600.

fax.pagesize: identifies the negotiated page size. Metric sizes are "A3", "A4", "A5", "A6", and "B4". Inch-based page sizes are "Letter" and "Legal".

fax.encoding: identifies the image encoding utilized. Valid values are "MH", "R", "MMR", and "JPEG".

fax.ecm: identifies whether ECM operation was used.

fax.pagebadlines: the number of bad lines in a page.

fax.objbadlines: the number of bad lines in an object.

fax.opbadlines: the number of bad lines in an operation.

fax.objuri: the objuri of the current object.

fax.resendcount: the number of pages resent due to errors.

fax.totalpages: the number of pages processed or stored.

fax.totalobjects: the count of the objects used in the operation.

fax.duration: the duration of the operation expressed as a duration in seconds and milliseconds (e.g., "23s250ms").

fax.result: contains the reason that caused the fax operation to complete. When the operation completes successfully, the value will be assigned "fax.success". Other values include "fax.partial", "fax.nofax", "fax.remotedisconnect", "fax.uri.access.error", and "fax.invalid.startpage".

The following sections describe the child elements of <faxsend>.

9.12.1.1. <sendobj>

<sendobj> is used to define a fax transmission. There MAY be multiple instances of the element, which will be transmitted in order.

Attributes:

objuri: a URI that points to the fax image that will be transmitted. Mandatory.

startpage: the first page of a multi-page objuri to send.

pagecount: page count.

9.12.1.2. <hdrfooter>

<hdrfooter> describes the header/footer that a media server MAY put on pages. The header or footer may be defined as the content of the <format> child element. The <format> element is only allowed if the type attribute has a value of "header" or "footer".

Attributes:

type: specifies whether a header or a footer should be put on pages and identifies the source of the header or footer. The following enumerated values may be used:

"header" indicates that the media server should put a header on pages using the contents of the <format> element.

"nohdr" indicates that there should be no header or footer.

"footer" indicates that the media server should put a footer on pages using the contents of the <format> element.

style: defines the style of insertion onto a fax page that a media server should use for the header or footer. Valid styles are "append", "overlay", or "replace".

<format> is a child of the <hdrfooter> element that defines the style format to be used for the header or footer. It uses a "C" language style format statement (as shown below) to define the contents and layout of the header or footer.

code	length	name	format
%a	3	day of week	3-character abbreviation
%d	2	date	01-31
%m	2	month	01-12
%y	2	year	00-99
%Y	4	year	0000-9999
%I	2	12 hour	01-12
%H	2	24 hour	00-23
%M	2	minute	00-59
%S	2	seconds	00-59
%p	2	AM/PM	AM or PM
%P	2	page number	01-99
%T	2	total pages	01-99
%l	20	local ID (sender)	0-9, + or spaces
%r	20	remote ID (rcvr)	0-9, + or spaces
%%	1	percent	display % in header/ftr

9.12.1.3. <rxpoll>

<rxpoll> provides the information necessary for a receive polling operation to occur. The object(s) to be received are defined by one or more <rcvobj> elements. The <rcvobj> is defined further under the child elements of <faxrcv>. The <rxpoll> element MAY also include a description of the header/footer that a media server SHOULD put on received pages. The <hdrfooter> element and its usage is described above.

Attributes:

rmtid: specifies the identifier of the remote fax terminal that is to be associated with a polling operation. A media server MUST NOT execute a polling operation unless the value of rmtid matches that of the connected remote machine. Mandatory.

9.12.1.4. <faxstart>

The <faxstart> element requests that an event be sent when fax operation has begun. When triggered, the following will be executed:

```
<send target="source" event="fax.start"/>
```

9.12.1.5. <faxnegotiate>

The <faxnegotiate> element requests that an event be sent when a negotiation has been completed. Multiple events MAY be sent each time a Digital Command Signal (DCS) frame is sent or received. When triggered, the following will be executed:

```
<send target="source" event="fax.negotiate"
   namelist="fax.rmtid
      fax.rate
      fax.resolution
      fax.pagesize
      fax.encoding
      fax.ecm"/>
```

9.12.1.6. <faxpagedone>

The <faxpagedone> element requests that an event be sent when a page has been sent or received. When triggered, the following will be executed:

```
<send target="source" event="fax.pagedone"
      namelist="fax.resolution
      fax.pagesize
      fax.encoding
      fax.pagebadlines
      fax.resendcount"/>
```

9.12.1.7. <faxobjectdone>

The <faxobjectdone> element requests that an event be sent when an objuri has been completed. When triggered, the following will be executed:

```
<send target="source" event="fax.objectdone"
      namelist="fax.objuri
      fax.objbadlines
      fax.resendcount
      fax.totalpages
      fax.result"/>
```

9.12.1.8. <faxopcomplete>

The <faxopcomplete> element requests that an event be sent when an operation has been completed. When triggered, the following will be executed:

```
<send target="source" event="fax.opcomplete"
      namelist="fax.totalpages
      fax.opbadlines
      fax.resendcount
      fax.totalobjects
      fax.duration
      fax.result"/>
```

9.12.1.9. <faxpollstarted>

The <faxpollstarted> element requests that an event be sent when a polling operation has started. When triggered, the following will be executed:

```
<send target="source" event="fax.opcomplete"
      namelist="fax.rmtid
      fax.rate
      fax.resolution
      fax.pagesize
      fax.encoding
      fax.ecm"/>
```

9.12.2. <faxrcv>

The <faxrcv> primitive provides the functionality of a called fax terminal. Typically this type of operation is to receive pages. However, it can include sending pages instead of, or in addition to, receiving them. The fax objects to receive are defined by the <rcvobj> elements, described below.

A media server SHOULD send pages as a polled terminal when the <txpoll> element is included as part of <faxrcv>. This element may be included in addition to, or instead of, the <rcvobj> element. One <rcvobj> or <txpoll> element must be present. When both are present, a media server SHOULD first receive pages and will then allow the other terminal to poll the media server, requesting pages.

Because fax is a distinct media type, the <faxrcv> primitive is not expected to interact with other primitives. Rather, it will interact using fax protocols with a remote fax terminal and will send

requested status events to its invoking environment. During fax operation, shadow variables are used to record the progress and parameters of the varying stages of fax operation.

Status events are requested by including one or more status request elements. These elements correspond to different stages or events in fax operation and cause predefined events to be sent to the invoking environment when they occur. Since the only recipient of these events is expected to be a fax control agent, requests are simplified by associating a predefined namelist of shadow variables with each event. This decision may be revisited to allowed tailored namelists based on further implementation experience. Status requests apply both to receiving and polling operation.

Attributes:

id: an optional identifier that may be referenced elsewhere for sending events to the faxrecv primitive.

lclid: the identifier that a media server uses to identify itself.

ecm: specifies whether ECM mode is allowed to be used if supported by the remote terminal. Defaults to "true".

Events:

terminate: terminates the fax reception operation.

Shadow Variables:

<faxrcv> supports the same set of shadow variables as <faxsend>

The following sections describe the child elements of <faxrcv>.

In addition to the elements defined below, <faxrcv> MAY also have the following child elements, which were defined under <faxsend>:

- <hdrfooter>
- <faxstart>
- <faxnegotiate>
- <faxpagedone>
- <faxobjectdone>
- <faxopcomplete>
- <faxpollstarted>

Their meaning and usage are the same as previously defined.

9.12.2.1. <rcvobj>

<rcvobj> is used to define fax objects that a media server will receive. There may be multiple instances of the element, which will be used in order.

Attributes:

objuri: a URI that points to the location that a received image is to be stored. Mandatory.

maxpages: the maximum number of pages that will be stored in objuri.

9.12.2.2. <txpoll>

<txpoll> provides the information for a polling operation to occur as part of a fax receive operation. An object or multiple objects to be sent may be supplied by one or more <sendobj> elements. In the event of multiple occurrences, a media server MUST select the <sendobj> element whose rmtid attribute matches that of the remote terminal.

The <sendobj> element was defined previously as a child element of <faxsend>. The <txpoll> element is extended with an rmtid attribute that specifies the identifier of the remote fax terminal and is used to select the specific <sendobj> to send.

A media server SHOULD put a header/footer on transmitted pages based on any <hdrfooter> element included as part of <txpoll>.

Attributes:

rmtid: specifies the identifier of the remote fax terminal that is to be associated with a polling operation. A media server MUST NOT execute a polling operation unless the value of rmtid matches that of the connected remote machine. Mandatory.

10. MSML Audit Package

10.1. MSML Audit Core Package

This section describes the MSML Audit Core Package that MAY be implemented to support auditing services.

Audit requests and results may vary based on the information being audited. The MSML Audit Core Package specifies the framework to send audit request, defines a state list, and builds audit results. The

additional audit packages define package specific state lists and associated audit result content. The additional audit packages MUST be defined within the framework specified by the Audit Core Package.

10.1.1. <audit>

The <audit> element is an optional child element of <msml>, which MAY be used by MSML clients to perform state auditing of current media resources allocated and in use by the media server. The requested state information is returned in an MSML response.

Attributes:

queryid: the identifier of the MSML object being queried by the MSML client. Mandatory. Supported object types: conference or connection. Wildcards are allowed.

statelist: a list of one or more state parameters that are being queried. Optional. If not present, the media server SHOULD return the id of audited object only. Each object type may contain a set of states. If the "statelist" contains any state that does not match the audited object type, the request MUST be rejected.

mark: in the case of an error, the value of the mark attribute from the last successfully executed element that included the mark attribute.

State Parameters:

The state parameter MUST be named using a dot-notation format "audit.X.a.b.c...", where X is the mandatory field that indicates the class name of the object (e.g., "conf" or "conn") and the "a.b.c..." is the optional field used to describe the actual name of the state parameter in a hierarchical manner. The wildcard "*" MAY be used as part of a state name; however, it MUST only be used in the last field of the dot-notation (e.g., "audit.conf.*" is valid, but "audit.conf.*.a" is invalid). When a wildcard is used, it is equivalent to querying all the states below the specified level. Each field (e.g., within "a.b.c...") will result in individual element names <a>, <b>, and <c> in the audit result to contain corresponding state value. The parent/child relationship between these elements follows the hierarchy of the state name (i.e., <c> is child element of <b>, and <b> is child element of <a>).

10.1.2. <auditresult>

The <auditresult> element is an optional child element of <result>, which MUST be used by the media server to return the audit result. A specific instance of the <auditresult> element contains the state information of a single active object. Therefore, if multiple objects are within the scope of the audit request, then one <auditresult> element per object MUST be present. A zero occurrence of <auditresult> element indicates that there are no active resources within the scope of the audit request.

Attributes:

targetid: the identifier of a conference or connection. Mandatory. Wildcard is not allowed.

The <auditresult> may contain child element(s) that return additional state information, corresponding to the "statelist" attribute in the <audit> request. The child element names correspond to the fields of the state parameter name (e.g., "a.b.c..."), following the same hierarchical structure.

10.2. MSML Audit Conference Package

This section describes the MSML Audit Conference Package that MUST be implemented to support auditing of conference services. The MSML Audit Conference Package follows the framework specified by the MSML Audit Core Package. This package defines the state parameter list and audit result for conference auditing.

10.2.1. State Parameters

All conference state parameter names MUST be prefixed by "audit.conf".

confconfig: query the conferences general configuration.

confconfig.audiomix: query the audio mixer’s general configuration in the conference.

confconfig.audiomix.asn: query the current ASN setting in the audio mixer.

confconfig.audiomix.n-loudest: query the current n-loudest setting in the audio mixer.

confconfig.videolayout: query the video layout’s general configuration in the conference.

confconfig.videolayout.root: query the root window setting of the video layout.

confconfig.videolayout.selector: query the video stream selector setting of the video layout.

confconfig.controller: query who is the conference controller.

dialog: query the active dialog information on the conference. See MSML Audit Dialog Package for details.

stream: query the active stream information on the conference. See MSML Audit Stream Package for details.

10.2.2. <auditresult>

The <auditresult> attribute of "targetid" is required to indicate results for auditing a conference.

The <auditresult> element may optionally contain the following child elements, returning additional conference state information, if corresponding states are queried and available.

10.2.2.1. confconfig

The <confconfig> element is used to return the general configuration state(s) of a conference, using the following attributes.

Attributes:

deletewhen: as defined by <createconference> element in MSML Conference Core Package.

term: as defined by <createconference> element in MSML Conference Core Package.

10.2.2.2. confconfig.audiomix

The <audiomix> element contains the general audio mixer configuration using the following attributes.

Attributes:

id: as defined by <audiomix> element in MSML Conference Core Package.

samplerate: as defined by <audiomix> element in MSML Conference Core Package.

10.2.2.3. confconfig.audiomix.asn

The <asn> element contains the current ASN setting of an audio mixer, if ASN is enabled. The state values are included in the following attributes.

Attributes:

ri: as defined by <asn> element in MSML Conference Core Package.

asth: as defined by <asn> element in MSML Conference Core Package.

10.2.2.4. confconfig.audiomix.n-loudest

The <n-loudest> element contains the current n-loudest setting of the audio mixer. The state values are included in the following attributes.

Attributes:

n: as defined by <n-loudest> element in MSML Conference Core Package.

10.2.2.5. confconfig.videolayout

The <videolayout> element contains the general video layout configuration using the following attributes.

Attributes:

id: as defined by <videolayout> in MSML Conference Core Package.

type: as defined by <videolayout> in MSML Conference Core Package.

10.2.2.6. confconfig.videolayout.root

The <root> element is used to contain root window settings.

Attributes:

size: as defined by <root> element in MSML Conference Core Package.

backgroundcolor: as defined by <root> element in MSML Conference Core Package.

Backgroundimage: as defined by <root> element in MSML Conference Core Package.

10.2.2.7. confconfig.videolayout.selector

The <selector> element is used to contain selector settings.

Attributes:

id: as defined by <selector> element in MSML Conference Core Package.

method: as defined by <selector> element in MSML Conference Core Package.

status: as defined by <selector> element in MSML Conference Core Package.

blankothers: as defined by <selector> element in MSML Conference Core Package.

si: as defined by <selector> element in MSML Conference Core Package when selector method is "vas".

speakersees: as defined by <selector> element in MSML Conference Core Package when selector method is "vas".

10.2.2.8. confconfig.controller

The <controller> element is used to return the conference controller id in its content. The conference controller is the SIP dialog that carries the <createconference> request. The return value is the MSML connection id.

10.2.2.9. dialog

If conference dialog state is queried, the audit result is returned using the <dialog> element as specified in the MSML Audit Dialog Package.

10.2.2.10. stream

If conference stream state is queried, the audit result is returned using the <stream> element as specified in the MSML Audit Stream Package.

10.3. MSML Audit Connection Package

This section describes the MSML Audit Connection Package that MAY be implemented to support auditing connection services. The MSML Audit Connection Package follows the framework specified by the MSML Audit Core Package. This package defines the state parameter list and audit result for connection auditing.

10.3.1. State Parameters

Connection state parameter names are prefixed by "audit.conn".

sipdialog: queries the identifier of the SIP dialog with which the connection is associated.

sipdialog.localseq: queries one of the SIP dialog states - local sequence number.

sipdialog.remoteseq: queries one of the SIP dialog states - remote sequence number.

sipdialog.localURI: queries one of the SIP dialog states - local URI.

sipdialog.remoteURI: queries one of the SIP dialog states - remote URI.

sipdialog.remotetarget: queries one of the SIP dialog states - remote target.

sipdialog.routeset: queries one of the SIP dialog states - route set.

localsdp: queries the local SDP body of the connection.

remotesdp: queries the remote SDP body of the connection.

dialog: queries the active dialog information on the connection. See MSML Audit Dialog Package for details.

stream: queries the active stream information on the connection. See MSML Audit Stream Package for details.

10.3.2. <auditresult>

The <auditresult> attribute "targetid" MUST specify a connection identifier for a connection result.

The <auditresult> element MAY contain the following child elements optionally to return additional connection state information if the corresponding states are queried and are available.

10.3.2.1. sipdialog

The <sipdialog> element contains the associated SIP dialog information. The SIP dialog ID information is returned using the following attributes.

Attributes:

callid: call-ID value as defined in [n1]. Mandatory.

localtag: local-tag value as defined in [n1]. Mandatory.

remotetag: remote-tag value as defined in [n1]. Mandatory.

This element can contain the following child elements optionally to return additional SIP dialog state information to the client if the corresponding states are queried and available.

10.3.2.2. sipdialog.localseq

The <localseq> element contains the local sequence number. The local sequence number is one of the SIP dialog states as defined in [n1].

10.3.2.3. sipdialog.remoteseq

The <remoteseq> element contains the remote sequence number. The remote sequence number is one of the SIP dialog states as defined in [n1].

10.3.2.4. sipdialog.localuri

The <localuri> element contains the local URI value. The local URI is one of the SIP dialog states as defined in [n1].

10.3.2.5. sipdialog.remoteuri

The <remoteuri> element contains the remote URI value. The remote URI is one of the SIP dialog states as defined in [n1].

10.3.2.6. sipdialog.remotetarget

The <remotetarget> element contains the remote target value. The remote target is one of the SIP dialog states as defined in [n1].

10.3.2.7. sipdialog.routeset

The <routeset> element contains the route-set value (an ordered list of URIs separated by comma). The route set is one of the SIP dialog states as defined in [n1].

10.3.2.8. localsdp

The <localsdp> element contains the local SDP body.

10.3.2.9. remotesdp

The <remotesdp> element contains the remote SDP body.

10.3.2.10. dialog

If the connection dialog state is queried, the audit result returns the queried information using the <dialog> element, as specified in the MSML Audit Dialog Package.

10.3.2.11. stream

If the connection stream state is queried, the audit result returns the queried information using the <stream> element, as specified in the MSML Audit Stream Package.

10.4. MSML Audit Dialog Package

This section describes the MSML Audit Dialog Package that MAY be implemented to support auditing dialogs. The MSML Audit Dialog Package follows the framework specified by the MSML Audit Core Package.

The MSML Audit Dialog Package must be used together with either the MSML Audit Conference Package or MSML Audit Connection Package, since the dialogs are applicable to conferences or connections.

10.4.1. State Parameters

Dialog state parameter names are prefixed by "dialog". Since this package must be used together with the MSML Audit Conference Package or MSML Audit Connection Package, the complete dialog state name must be prefixed by "audit.conf.dialog" or "audit.conn.dialog", depending on the context within which the dialog state is queried.

dialog: queries the number of active dialog(s) running on the target (a conference or connection); basic dialog information will be returned.

dialog.duration: queries the amount of time a dialog has been running.

dialog.primitive: queries the media primitive currently being executed by the dialog.

dialog.controller: queries the dialog controller.

10.4.2. <dialog>

The <dialog> element is a child element of <auditresult>, which contains the active dialog information on the target identified by the attribute "targetid" of the <audioresult> element.

Basic dialog information is returned using the following attributes.

Attributes:

src: as defined by the <dialogstart> element in the MSML Dialog Core Package.

type: as defined by the <dialogstart> element in the MSML Dialog Core Package. Mandatory.

name: as defined by the <dialogstart> element in the MSML Dialog Core Package. Mandatory.

This element may contain the following child elements optionally to return additional dialog information if the corresponding state parameter has been queried and the state value is available.

10.4.2.1. <duration>

The <duration> element returns the duration that a dialog has been running on the specified target. The duration value is included in the element content. It is a positive integer value (in unit of seconds).

10.4.2.2. <primitive>

The <primitive> element returns the currently active media primitive in its content. The active media primitive is the primitive that is currently being executed. Possible return values are play, dtmf, collect, dtmfgen, tonegen, record, or none.

10.4.2.3. <controller>

The <controller> element returns the dialog controller id in its content. The dialog controller is the SIP dialog that carries the <dialogstart> request. The returned value is the MSML connection id.

10.5. MSML Audit Stream Package

This section describes the MSML Audit Stream Package that MAY be implemented to support auditing stream. The MSML Audit Stream Package follows the framework specified by the MSML Audit Core Package.

The MSML Audit Stream Package MUST be used together with either the MSML Audit Conference Package or the MSML Audit Connection Package, since the stream is applicable between conferences, between connections, or between conferences and connections.

10.5.1. State Parameters

Stream state parameter names are prefixed by "stream". Since this package must be used together with the MSML Audit Conference Package or MSML Audit Connection Package, the complete stream state name must be prefixed by "audit.conf.stream" or "audit.conn.stream", depending on the context within which the stream state is queried.

stream: queries the number of active streams created on the audited object; basic stream information will be returned.

stream.clamp: queries the clamping status.

stream.gain: queries the gain control information.

stream.visual: queries the visual setting.

10.5.2. <stream>

The <stream> element is a child element of <auditresult> and contains the active stream information on the target identified by the attribute "targetid" of the <audioresult> element.

Basic stream information is returned using the following attributes.

Attributes:

joinwith: an identifier of either a connection or a conference with which the audited object is joined. Mandatory. Wildcard is not allowed.

media: as defined by the <stream> element in the MSML Conference Core Package. Mandatory.

dir: direction of stream, from audited target perspective, "from" or "to". Mandatory.

compressed: as defined by the <stream> element in the MSML Conference Core Package.

display: as defined by the <stream> element in the MSML Conference Core Package.

override: as defined by the <stream> element in the MSML Conference Core Package.

preferred: as defined by the <stream> element in the MSML Conference Core Package.

This element MAY contain the following child elements that optionally return additional stream information, if the corresponding state parameter is queried and the state value is available.

10.5.2.1. <clamp>

The <clamp> element is included if stream clamping is active. The currently active clamping state values are returned using the attributes as defined by the <clamp> element in the MSML Conference Core Package.

10.5.2.2. <gain>

The <gain> element is included if stream gain is active. The current gain control state values are returned using the attributes as defined by the <gain> element in the MSML Conference Core Package.

10.5.2.3. <visual>

The <visual> element is included if stream visual display is active. The current visual display settings are returned using the attributes as defined by the <visual> element in the MSML Conference Core Package.

11. Response Codes

Response codes are used to indicate reasons for failures as well as completion status. The appropriate code and description must be passed to the invoking environment on failure.

The response codes defined in this section are returned as the value of the response attribute to the <result> element. Some values may also be returned as part of a namelist to an "msml.dialog.exit" event generated when an executing MSML dialog fails.

Informational (1xx)

Reserved for future use

Success (200)

200 OK

Request Error (4xx)

400 Bad Request
401 Unknown Element
402 Unsupported Element
403 Missing mandatory element content
404 Forbidden element content
405 Invalid element content
406 Unknown attribute
407 Attribute not supported
408 Missing mandatory attribute
409 Forbidden attribute is present

410 Invalid attribute value

420 Unsupported media description language
421 Unknown media description language
422 Ambiguous request (both URI and inline description)
423 External document fetch error
424 Syntax error in foreign language
425 Semantic error in foreign language
426 Unknown error executing foreign language

430 Object does not exist
431 Object instance name already used
432 Conference name already in use
433 reserved
434 External document fetch error

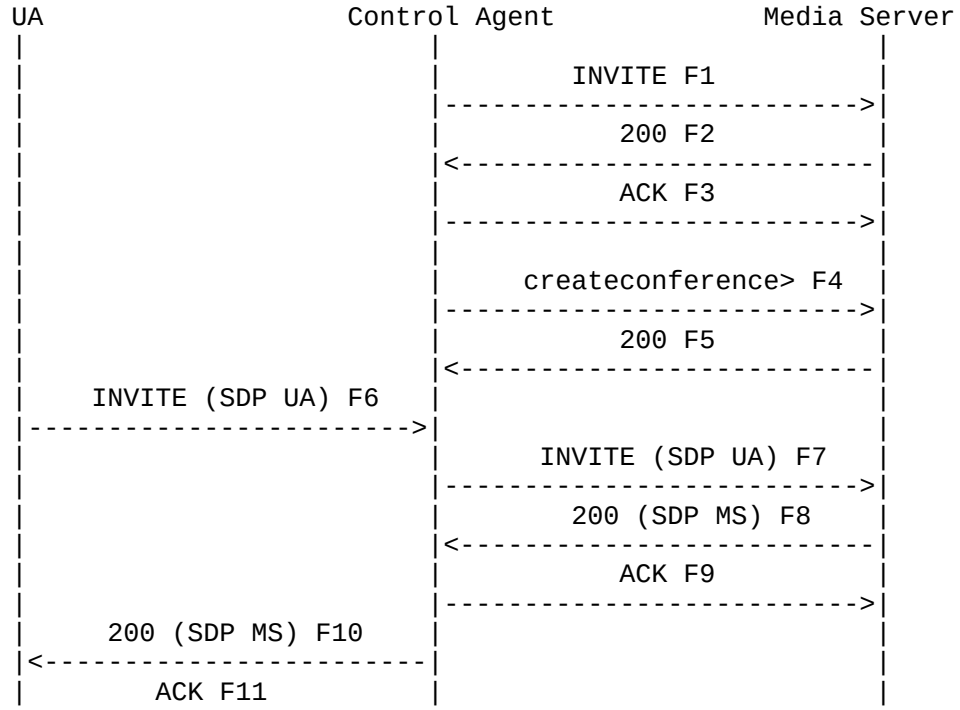
440 Cannot join objects of the specified class
441 Objects have incompatible media types
442 reserved
443 reserved
444 Number of media inputs exceeded

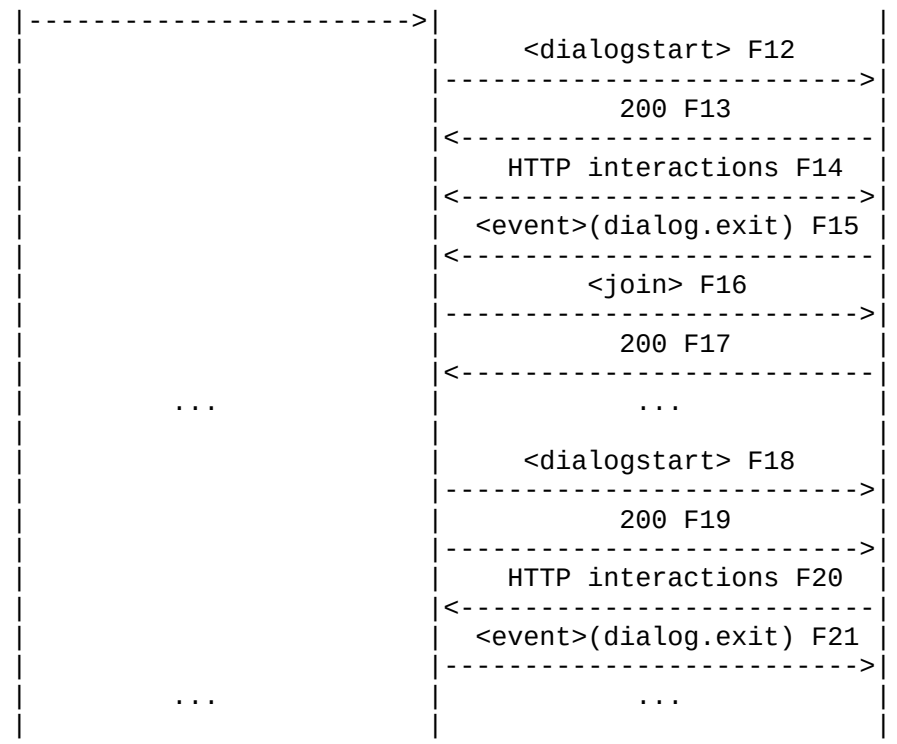
```
      450  Objects have incompatible media formats
      451  Incompatible media stream format

   Server Error (5xx)

      500  Internal media server error
      503  Service Unavailable
      510  Not in service
      511  Service Unavailable
      520  No resource to fulfill request
      521  Internal limit exceeded
```

12. MSML Conference Examples

These examples focus on the MSML Conference Core Package used by a control agent (CA) to control services on a media server (MS). They show the relationship between SIP signaling to establish media sessions and MSML service control commands. For brevity, only the content of MSML messages is shown. The examples assumes that the CA and MS use the IPv4 address and UDP port number of the audio stream (on the MS) to identify the MSML connection.

12.1. Establishing a Dial-In Conference





Steps 1-3: establish an MSML control channel for the conference. Alternatively, a control channel could already have been established that was used for all CA/MS interactions. A control channel per conference is only one possible model. Currently, MSML uses SIP INFO requests and responses on this SIP dialog. There is a proposal to use this message exchange to establish a TCP channel for MSML similar to the approach used for the Media Resource Control Protocol v2 (MRCPv2). This approach would require that a request identifier be added to the <msml> element to correlate requests and responses. This currently relies on the SIP INFO request and response for this property. MSML messages are shown without specifying the transport in this example, but it assumes a request/response correlation based on transport messages.

Step 4: create a conference that will mix the loudest two speakers and report those speakers to the control agent every 10 seconds. The media server will automatically terminate remaining media sessions and delete the conference and associated resources and when the control channel is terminated.

```
   <msml version="1.1">
      <createconference name="exampleConf" deletewhen="nocontrol">
          <audiomix>
             <n-loudest n="3"/>
             <asn ri="10s"/>
          </audiomix>
      </createconference>
   </msml>
```

Step 5: conference created successfully

```
   <msml version="1.1">
      <result response="200"/>
   </msml>
```

Steps 6-11: standard 3PCC establishment of a user-initiated media session to a media server. This is the equivalent of a dial-in conference participant. The "To:" header returned by the MS in the 200 response of Step F8 was:

```
To: <sip:msml@ms.example.com>;tag=jd87dfg4h
```

Step 12: request an initial dialog with the participant to prompt for their name, desired conference, etc. The dialog completes by informing the participant that they are joining the conference. If this was not the first participant, the dialog could also announce the other participants.

```
   <msml version="1.1">
      <dialogstart target="conn:jd87dfg4h" name="12345"
          type="application/vxml+xml"
          src="http://server.example.com/scripts/initial.vxml"/>
   </msml>
```

Step 13: dialog started successfully. The dialog identifier is returned.

```
   <msml version="1.1">
      <result response="200"/>
      <dialogid>conn:jd87dfg4h/dialog:12345</dialogid>
   </msml>
```

Step 14: sequence of HTTP VoiceXML dialog interactions.

Step 15: the VoiceXML browser exits (but does not disconnect). If a namelist had been specified within the VoiceXML <exit> element, it would have been included in the <event> sent to the CA.

```
   <msml version="1.1">
      <event name="msml.dialog.exit"
             id="conn:jd87dfg4h/dialog:12345"/>
   </msml>
```

Step 16: join the participant to the conference and have the volume of their contributing audio automatically adjusted to a target level of -20 dBm0.

```
   <msml version="1.1">
      <join id1="conn:jd87dfg4h" id2="conf:exampleConf">
         <stream media="audio" dir="from-id1">
             <gain agc="true" tgtlvl="-20"/>
         </stream>
         <stream media="audio" dir="to-id1"/>
   </msml>
```

Step 17: successfully joined to conference

```
   <msml version="1.1">
      <result response="200"/>
   </msml>
```

Steps 6 through 17 are repeated for the second participant.

Step 18: play a join tone or message announcing the new participant to the conference.

```
   <msml version="1.1">
      <dialogstart target="conf:exampleConf"
          type="application/vxml+xml"
          src="http://server.example.com/scripts/joinmsg.vxml"/>
   </msml>
```

Step 19: dialog started successfully. The dialog identifier is returned. The media server assigned a unique identifier since name attribute was not specified in <dialogstart>.

```
   <msml version="1.1">
      <result response="200"/>
      <dialogid>conf:ExampleConf/dialog:j6fs8745</dialogid>
   </msml>
```

Step 20: HTTP VoiceXML dialog interaction(s).

Step 21: the VoiceXML browser exits.

```
<msml version="1.1">
   <event name="msml.dialog.exit"
          id="conf:ExampleConf/dialog:j6fs8745"/>
</msml>
```

Steps 6 through 21 are repeated for the third and subsequent participants.

12.2. Example of a Sidebar Audio Conference

This example assumes that a conference has already been established as in the previous example. It creates a sidebar conference that hears the main conference as a whisper. Three participants are moved to the sidebar. After some period of time, the sidebar participants are returned to the main conference and the sidebar is deleted.

Step 1: the sidebar conference is created. It is joined half-duplex to the main conference and a manual gain object is inserted in the media stream. Three participants are then moved from the main conference to the sidebar. Although not shown, a CA could include the "mark" attribute in each element to allow recovery in the event of a mid- transaction error.

```
<msml version="1.1">
   <createconference name="sidebarConf"
                     deletewhen="nomedia">
       <audiomix/>
   </createconference>
   <join id1="conf:sidebarConf" id2="conf:exampleConf">
       <stream media="audio" dir="to-id1">
           <gain amt="-20"/>
       </stream>
   </join>
   <unjoin id1="conn:gs5s4-1" id2="conf:exampleConf"/>
   <join id1="conn:gs5s4-1" id2="conf:sidebarConf"/>
   <unjoin id1="conn:hd764gr9-2" id2="conf:exampleConf"/>
   <join id1="conn:hd764gr9-2" id2="conf:sidebarConf"/>
   <unjoin id1="conn:h37frdvgs65-3" id2="conf:exampleConf"/>
   <join id1="conn:h37frdvgs65-3" id2="conf:sidebarConf"/>
</msml>
```

Step 2: sidebar conference created successfully and participants joined.

```
<msml version="1.1">
   <result response="200"/>
</msml>
```

Step 3: once the sidebar conference has completed, the participants are rejoined to the main conference. The sidebar is destroyed automatically by the MS when the last media stream is removed as specified when the sidebar conference was created.

```
<msml version="1.1">
   <unjoin id1="conn:gs5s4-1" id2="conf:sidebarConf"/>
   <join id1="conn:gs5s4-1" id2="conf:exampleConf"/>
   <unjoin id1="conn:hd764gr9-2" id2="conf:sidebarConf"/>
   <join id1="conn:hd764gr9-2" id2="conf:exampleConf"/>
   <unjoin id1="conn:h37frdvgs65-3" id2="conf:sidebarConf"/>
   <join id1="conn:h37frdvgs65-3" id2="conf:exampleConf"/>
</msml>
```

Step 4: participants successfully moved to main conference and sidebar destroyed.

```
<msml version="1.1">
   <result response="200"/>
</msml>
```

12.3. Example of Removing a Conference

This example assumes a conference created similar to the first example where there is an MSML control channel specific to the conference and the conference has been configured to be deleted when that channel is removed (using SIP).

Steps 1-2: the CA signals BYE for the SIP dialog used to establish the conference control channel.

Steps 3-6: the MS initiates terminating the media sessions for each participant remaining in the conference.

The MS deletes the conference and removes all resources when the last participant has been removed.

12.4. Example of Modifying Video Layout

Assume that a conference named "example" is created using the following mixer descriptions.

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

```
<createconference name="quad-split">
   <audiomix>
      <n-loudest n="3"/>
      <asn ri="10s"/>
   </audiomix>
   <videolayout>
      <root size="CIF" background="white" />
      <selector id="default" method="vas" si="500ms">
         <region id="1" left="0" top="0" relativesize="1/4"/>
      </selector>
      <region id="2" left="50%" top="0" relativesize="1/4"/>
      <region id="3" left="0%" top="50%" relativesize="1/4">
      <region id="4" left="50%" top="50%" relativesize="1/4"/>
   </videolayout>
</createconference>
```

The following would change the size of the video window to QCIF and the background color to the default "black".

```
<modifyconference id="conf:example">
   <videolayout>
      <root size="4CIF"/>
   </videolayout>
</modifyconference>
```

The relative location of the regions does not change. However, the sizes of the regions do change because they are relative to the size of the root window. The result is a layout that looks identical but half the size.

The following would freeze the video displayed in region "2" without affecting any other attributes of that region.

```
<modifyconference id="conf:example">
   <videolayout>
      <region id="2" left="50%" top="0" relativesize="1/4"
                     freeze="true"/>
   </videolayout>
</modifyconference>
```

13. MSML Dialog Examples

These examples focus on the MSML Dialog Base Package and the MSML Dialog Group Package.

13.1. Announcement

The following is a simple announcement scenario. Two recorded audio files are played in sequence followed by generated speech followed by a variable. The results are reported once media generation completes.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <dialogstart target="conn:12345" name="12345">
      <play>
         <audio uri="file://clip1.wav"/>
         <audio uri="http://host1/clip2.wav"/>
         <tts uri="http://host2/text.ssml"/>
         <var type="date" subtype="mdy" value="20030601"/>
      </play>
      <send target="source" event="done" namelist="play.amt
                                          play.end"/>
   </dialogstart>
</msml>
```

13.2. Voice Mail Retrieval

Below is an example that shows a simple voice mail retrieval operation consisting of playing a message and allowing the user to pause and resume play using '5' to toggle the state. The operation would terminate when the play completed or the user entered '#'.

During the play, the user can advance forward and backward through the message as well as rewinding to the beginning.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
  <dialogstart target="conn:12345" name="12345">
   <group topology="parallel">
      <play>
         <audio uri="file://message.wav"/>
         <playexit>
            <send target="group" event="terminate"/>
         </playexit>
      </play>
      <dtmf iterate="forever">
         <pattern digits="5">
            <send target="play" event="toggle-state"/>
         </pattern>
         <pattern digits="6">
            <send target="play" event="forward"/>
         </pattern>
         <pattern digits="7">
            <send target="play" event="backward"/>
         </pattern>
         <pattern digits="8">
            <send target="play" event="restart"/>
         </pattern>
         <pattern digits="#">
            <send target="play" event="terminate"/>
         </pattern>
      </dtmf>
   </group>
 </dialogstart>
</msml>
```

13.3. Play and Record

A more complex example is a play and record operation. This sources and sinks media and uses voice activity DTMF detection and recognition to influence behavior. Any DTMF input or voice activity will barge the play and cause the record to begin. However, if the prompt was barged with a DTMF digit of '#', the record terminates without starting. When the play terminates, it send a starttimer event to the VAD to allow it to recognize an initial silence condition. The recording will be terminated (without starting) when the VAD detects an initial 3 seconds of silence.

Once resumed (based upon voice detection), the recording may be terminated under several conditions. It will terminate after 5 seconds of silence or after 60 seconds elapses. It will also terminate if a '#' key is recognized. Every aspect of this behavior can be modified by changing what is recognized and the events that are sent. The following example uses the MSML Dialog Group Package.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml  version="1.1">
 <dialogstart target="conn:12345" name="12345">
   <group topology="parallel">
      <play>
         <audio uri="file://prompt.wav"/>
         <playexit>
            <send target="vad" event="starttimer"/>
         </playexit>
      </play>
      <dtmf>
         <pattern digits="#">
            <send target="record" event="terminate.termkey"/>
         </pattern>
         <detect>
            <send target="play" event="terminate"/>
         </detect>
      </dtmf>
      <vad>
         <voice len="10ms">
            <send target="play" event="terminate"/>
            <send target="record" event="resume"/>
         </voice>
         <silence len="3s">
            <send target="record" event="nospeech"/>
         </silence>
         <tsilence len="5s">
            <send target="record" event="terminate.finalsilence"/>
         </tsilence>
      </vad>
      <record initial="suspend" maxtime="60s"
              dest="file://record.wav" format="g729">
         <recordexit>
            <send target="group" event="terminate"/>
         </recordexit>
      </record>
      <groupexit>
         <send target="source" event="done"
               namelist="record.len record.end"/>
      </groupexit>
   </group>
 </dialogstart>
</msml>
```

The following implements the same functionality, as described above, in using the MSML Dialog Base Package, using the <record> composite mechanism for the play and record operation.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml  version="1.1">
 <dialogstart target="conn:12345" name="12345">
   <record prespeech="3s" postspeech="5s" maxtime="60s" termkey="#"
              dest="file://record.wav" format="g729">
      <play barge="true">
         <audio uri="file://prompt.wav"/>
      </play>
      <recordexit>
         <send target="source" event="done"
               namelist="record.len record.end"/>
      </recordexit>
   </record>
 </dialogstart>
</msml>
```

13.4. Speech Recognition

The following simple example requests that a user speak the name of a city and returns the result.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
 <dialogstart target="conn:12345" name="12345">
   <group topology="parallel">
      <play>
         <audio uri="file://prompt.wav"/>
      </play>
      <speech>
         <grammar version="1.0">
            <rule id="city" scope="public">
               <item>
                  <one-of>
                     <item>vancouver</item>
                     <item>new york</item>
                     <item>london</item>
                  </one-of>
               </item>
            </rule>
            <match>
               <send target="group" event="terminate"/>
            </match>
         </grammar>
         <noinput>
            <send target="group" event="terminate"/>
         </noinput>
         <nomatch>
            <send target="group" event="terminate"/>
         </nomatch>
      </speech>
      <groupexit>
         <send target="source" event="done"
                        namelist="speech.end speech.results"/>
      </groupexit>
   </group>
 </dialogstart>
</msml>
```

13.5. Play and Collect

This example prompts a user to enter 4 DTMF digits terminated by the '#' key (represented by "xxxx#" below). The prompt will be barged and the user has 10 seconds to begin entering input or no input will be indicated.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
 <dialogstart target="conn:12345" name="12345">
   <group topology="parallel">
      <play>
         <audio uri="file://prompt.wav"/>
         <playexit>
            <send target="dtmf" event="starttimer"/>
         </playexit>
      </play>
      <dtmf fdt="10s" idt="16s">
         <pattern digits="xxxx#">
            <send target="group" event="terminate"/>
         </pattern>
         <detect>
            <send target="play" event="terminate"/>
         </detect>
         <noinput>
            <send target="group" event="terminate"/>
         </noinput>
         <nomatch>
            <send target="group" event="terminate"/>
         </nomatch>
      </dtmf>
      <groupexit>
         <send target="source" event="done"
               namelist="dtmf.digits dtmf.end"/>
      </groupexit>
   </group>
 </dialogstart>
</msml>
```

The following implements the same functionality, as described above, using the MSML Dialog Base Package, using the <collect> composite mechanism for the play and collect operation.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
 <dialogstart target="conn:12345" name="12345">

   <collect fdt="10s" idt="16s">
      <play barge="true">
         <audio uri="file://prompt.wav"/>
      </play>
      <pattern digits="xxxx#">
         <send target="source" event="done"
               namelist="dtmf.digits dtmf.end"/>
      </pattern>
      <noinput>
         <send target="source" event="done"
               namelist="dtmf.end"/>
      </noinput>
      <nomatch>
         <send target="source" event="done"
               namelist="dtmf.end"/>
      </nomatch>
   </collect>
 </dialogstart>
</msml>
```

13.6. User Controlled Gain

This shows an example of nesting groups to create an arbitrary full-duplex media control. DTMF is detected on media flowing in one direction and used to adjust the gain applied to media flowing in the opposite direction. Additionally, the stream that is used to detect DTMF has DTMF removed and its gain automatically adjusted before leaving the group. This widget could be used between a conference participant and a conference mixer.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml  version="1.0">
 <dialogstart target="conn:12345" name="12345">
   <group topology="fullduplex">
      <group topology="parallel">
         <dtmf>
            <pattern digits="1" iterate="forever">
               <send target="gain" event="louder"/>
            </pattern>
            <pattern digits="2" iterate="forever">
               <send target="gain" event="softer"/>
            </pattern>
         </dtmf>
         <group topology="serial">
            <clamp/>
            <agc tgtlvl="0"/>
         </group>
      </group>
      <gain amt="0" incr="5"/>
   </group>
 </dialogstart>
</msml>
```

14. MSML Audit Examples

The following examples describe the MSML Audit Conference Package and the MSML Audit Connection Package, and their use together with the MSML Audit Dialog Package or/and the MSML Audit Stream Package.

14.1. Audit All Conferences

This example describes an audit of all active conferences on the media server, querying the conference configurations.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <audit queryid="conf:*" statelist="audit.conf.confconfig.*"/>
</msml>
```

The following result assumes two conferences currently allocated by the media server. Conference "conf:1" contains both an audio mixer (with ASN enabled) and a video layout (vas) created, while conference "conf:2" contains only an audio mixer created with ASN disabled.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <result response="200">
      <auditresult targetid="conf:1">
         <confconfig deletewhen="nocontrol" term="true">
            <audiomix id="audiomix1">
               <asn ri="5s"/>
               <n-loudest n="16"/>
            </audiomix>
              <videolayout id="videolayout1"
                           type="text/msml-basic-layout">
                 <selector id="selector1" method="vas" si="5s"
                           speakersees="current">
                    <root size="CIF"/>
                 </selector>
              </videolayout>
            <controller>conn:1234</controller>
         </confconfig>
      </auditresult>
      <auditresult targetid="conf:2">
         <confconfig deletewhen="nomedia" term="true">
            <audiomix id="audiomix2">
               <n-loudest n="1"/>
            </audiomix>
            <controller>conn:1234</controller>
         </confconfig>
      </auditresult>
   </result>
</msml>
```

14.2. Audit Conference Dialogs

This example describes an audit of active dialogs on a specific conference. The request queries all available dialog states.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <audit queryid="conf:1" statelist="audit.conf.dialog.*"/>
</msml>
```

The example result assumes a single dialog running on conference "conf:1", which has been running for 60 seconds, and the dialog is currently executing a record operation.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <result response="200">
      <auditresult targetid="conf:1">
         <dialog name="sample">
            <duration>60</duration>
            <primitive>record</primitive>
            <controller>conn:1234</controller>
         </dialog>
      </auditresult>
   </result>
</msml>
```

14.3. Audit Conference Streams

This example request describes an audit of active streams on a specific conference. The request queries all available stream states.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <audit queryid="conf:1" statelist="audit.conf.stream.*"/>
</msml>
```

The example result assumes three audio participants in the conference. Connection "conn:1234" is a talk-listen participant with both clamp and gain control enabled. Connection "conn:1235" is a talk-only participant. Connection "conn:1236" is a listen-only participant with automatic gain control enabled.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <result response="200">
      <auditresult targetid="conf:1">
         <stream joinwith="conn:1234" media="audio" dir="to">
            <clamp dtmf="true" tone="false"/>
            <gain amt="-10"/>
         </stream>
         <stream joinwith="conn:1234" media="audio" dir="from">
            <gain amt="10"/>
         </stream>
         <stream joinwith="conn:1235" media="audio" dir="to">
         </stream>
         <stream joinwith="conn:1236" media="audio" dir="from">
            <gain agc="true" tgtlvl="0" maxgain="10"/>
         </stream>
      </auditresult>
   </result>
</msml>
```

14.4. Audit All Connections

This example request describes an audit of all active connections on the media server. No additional state is queried.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <audit queryid="conn:*"/>
</msml>
```

The example result assumes five connections currently allocated by the media server.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <result response="200">
      <auditresult targetid="conn:1230"/>
      <auditresult targetid="conn:1231"/>
      <auditresult targetid="conn:1232"/>
      <auditresult targetid="conn:1233"/>
      <auditresult targetid="conn:1234"/>
   </result>
</msml>
```

14.5. Audit Connection Dialogs

This example request describes an audit of active dialogs on a specific connection. No additional dialog state is queried.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <audit queryid="conn:1234" statelist="audit.conn.dialog"/>
</msml>
```

The example result assumes three dialogs running on the connection.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <result response="200">
      <auditresult targetid="conn:1234">
         <dialog name="sample1"/>
         <dialog name="sample2"/>
         <dialog name="sample3"/>
      </auditresult>
   </result>
</msml>
```

14.6. Audit Connection Streams

This example request describes an audit of active streams on a specific connection. No additional stream state is queried.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <audit queryid="conn:1234" statelist="audit.conn.stream"/>
</msml>
```

The example result assumes three audio streams created between target connection and other MSML objects, one of which is a bidirectional stream between target connection and a conference, and two are unidirectional streams between two other connections.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <result response="200">
      <auditresult targetid="conn:1234">
         <stream joinwith="conf:1" media="audio" dir="to"/>
         <stream joinwith="conf:1" media="audio" dir="from"/>
         <stream joinwith="conn:1235" media="audio" dir="to"/>
         <stream joinwith="conn:1236" media="audio" dir="from"/>
      </auditresult>
   </result>
</msml>
```

14.7. Audit Connection with Selective States

This example describes an audit of a specific connection, querying associated SIP dialog ID and SDP info.

```
<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <audit queryid="conn:1234" statelist="audit.conn.sipdialog
      audit.conn.localsdp audit.conn.remotesdp"/>
</msml>

<?xml version="1.0" encoding="UTF-8"?>
<msml version="1.1">
   <result response="200">
      <auditresult targetid="conn:1234">
         <sipdialog callid="ABCD@10.0.0.10:5060"
                    localtag="sdfjsiodf"
                    remotetag="zvnmviuhd8"/>
         <localsdp>
            v=0
            o=- 31691 31691 IN IP4 ms5mpc11.lab.radisys.com
            s=media server session
            t=0 0
            m=audio 33794 RTP/AVP 0
            c=IN IP4 10.3.5.111
            a=rtpmap:0 PCMU/8000
            a=sendrecv
            m=video 32770 RTP/AVP 34
            c=IN IP4 10.3.5.11
            b=AS:48
            a=rtpmap:34 H263/90000
            a=fmtp:34 CIF=1
            a=sendrecv
         </localsdp>
         <remotesdp>
            v=0
            o=- 12345 12345 IN IP4 10.0.0.88
            s=RadiSys SIP Media Server session
            t=0 0
            c=IN IP4 10.0.0.126
            b=AS:128
            m=audio 10000 RTP/AVP 0
            a=rtpmap:0 PCMU/8000
            a=ptime:20
            a=sendrecv
            m=video 10002 RTP/AVP 34
            a=rtpmap:34 H263/90000
            a=fmtp:34 CIF=1
```

```
                a=sendrecv
              </remotesdp>
            </auditresult>
          </result>
       </msml>
```

15. Future Work

The following capabilities may be added in future versions of this document:

- Ability for MSML clients to audit or query the media server for supported set of MSML packages and profiles.
- Ability to version MSML packages and profiles and naming scheme for MSML extension packages.

16. XML Schema

MSML specification consists of a set of XML schemas, all of which may be used together or any sub-set of the schemas may be used for each MSML package. The following sections define a complete set of schemas covering all MSML packages.

Each package contains a single schema file, <package-name>-datatypes.xsd. This schema file can be included by its extended package(s). Every package optionally contains another schema file, <package_name>.xsd, which can be used directly to build or validate MSML scripts for a given package.

The complete MSML schema (msml.xsd) includes all the individual MSML packages.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-conf-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-base-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-transform-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-group-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-speech-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-fax-detect-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-fax-sendrecv-
            datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-core-datatypes.xsd"/>
```

```
    <xs:include schemaLocation="msml-audit-conf-datatypes.xsd"/>
    <xs:include schemaLocation="msml-audit-conn-datatypes.xsd"/>
    <xs:include schemaLocation="msml-audit-dialog-datatypes.xsd"/>
    <xs:include schemaLocation="msml-audit-stream-datatypes.xsd"/>
    <xs:element name="msml">
     <xs:complexType>
      <xs:choice>
       <xs:group ref="msmlRequestType" maxOccurs="unbounded"/>
       <xs:element name="event">
        <xs:complexType>
         <xs:choice maxOccurs="unbounded">
          <xs:sequence>
           <xs:element name="name" type="msmlEventNameValue.datatype"/>
           <xs:element name="value">
            <xs:simpleType>
             <xs:restriction base="xs:string">
              <xs:pattern value="[a-zA-Z0-9.]+"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:element>
          </xs:sequence>
         </xs:choice>
         <xs:attribute name="name" type="msmlEventName.datatype"
                       use="required"/>
         <xs:attribute name="id" type="msmlEventSource.datatype"
                       use="required"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="result">
        <xs:complexType>
         <xs:choice>
          <xs:element ref="description" minOccurs="0"/>
          <xs:sequence>
           <xs:element ref="msmlResultSimple" minOccurs="0"
                       maxOccurs="unbounded"/>
           <xs:element ref="msmlResultComplex" minOccurs="0"
                       maxOccurs="unbounded"/>
          </xs:sequence>
         </xs:choice>
         <xs:attribute name="response">
          <xs:simpleType>
           <xs:restriction base="xs:string">
            <xs:pattern value="\d{3}"/>
           </xs:restriction>
          </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="mark" type="mark.datatype"/>
        </xs:complexType>
```

```
       </xs:element>
      </xs:choice>
      <xs:attribute name="version" type="xs:string" use="required"
                    fixed="1.1"/>
     </xs:complexType>
    </xs:element>
   </xs:schema>
```

16.1. MSML Core

16.1.1. msml-core.xsd

```
   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation="msml-core-datatypes.xsd"/>
    <xs:element name="msml">
     <xs:complexType>
      <xs:choice>
       <xs:group ref="msmlRequestType" maxOccurs="unbounded"/>
       <xs:element name="event">
        <xs:complexType>
         <xs:choice maxOccurs="unbounded">
          <xs:sequence>
           <xs:element name="name" type="msmlEventNameValue.datatype"/>
           <xs:element name="value">
            <xs:simpleType>
             <xs:restriction base="xs:string">
              <xs:pattern value="[a-zA-Z0-9.]+"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:element>
          </xs:sequence>
         </xs:choice>
         <xs:attribute name="name" type="msmlEventName.datatype"
                       use="required"/>
         <xs:attribute name="id" type="msmlEventSource.datatype"
                       use="required"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="result">
        <xs:complexType>
         <xs:choice>
          <xs:element ref="description" minOccurs="0"/>
          <xs:sequence>
           <xs:element ref="msmlResultSimple" minOccurs="0"
                       maxOccurs="unbounded"/>
           <xs:element ref="msmlResultComplex" minOccurs="0"
```

```
                    maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:choice>
        <xs:attribute name="response">
         <xs:simpleType>
          <xs:restriction base="xs:string">
           <xs:pattern value="\d{3}"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="mark" type="mark.datatype"/>
       </xs:complexType>
      </xs:element>
     </xs:choice>
     <xs:attribute name="version" type="xs:string" use="required"
                   fixed="1.1"/>
    </xs:complexType>
   </xs:element>
  </xs:schema>
```

16.1.2. msml-core-datatypes.xsd

```
  <?xml version="1.0" encoding="UTF-8"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
             elementFormDefault="qualified"
             attributeFormDefault="unqualified">
   <xs:group name="msmlRequestType">
    <xs:choice>
     <xs:element ref="msmlRequest"/>
     <xs:element name="send">
      <xs:complexType>
       <xs:complexContent>
        <xs:extension base="msmlRequestType">
         <xs:attribute name="event" type="msmlEvent.datatype"
                       use="required"/>
         <xs:attribute name="target" type="msmlTarget.datatype"
                      use="required"/>
         <xs:attribute name="valuelist" type="xs:string"/>
        </xs:extension>
       </xs:complexContent>
      </xs:complexType>
     </xs:element>
    </xs:choice>
   </xs:group>
   <xs:element name="msmlRequest" type="msmlRequestType"
               abstract="true"/>
   <xs:complexType name="msmlRequestType">
    <xs:attribute ref="mark"/>
```

```
    </xs:complexType>
    <xs:element name="msmlResultSimple" type="msmlResultSimpleType"
                abstract="true"/>
    <xs:element name="msmlResultComplex" type="msmlResultComplexType"
                abstract="true"/>
    <xs:simpleType name="msmlResultSimpleType">
     <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:complexType name="msmlResultComplexType"/>
    <xs:element name="description" type="xs:string"/>
    <xs:attribute name="mark" type="mark.datatype"/>
    <xs:simpleType name="msmlInstanceID.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9.:\-_]+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="connID.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern value="conn:[a-zA-Z0-9.:\-_]+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="confID.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern value="conf:[a-zA-Z0-9.:\-_]+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="dialogID.datatype">
     <xs:restriction base="xs:string">
   <xs:pattern value="conf:[a-zA-Z0-9.:\-_]+/dialog:[a-zA-Z0-9.:\-_]+"/>
   <xs:pattern value="conn:[a-zA-Z0-9.:\-_]+/dialog:[a-zA-Z0-9.:\-_]+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="independentID.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern value="conf:[a-zA-Z0-9.:\-_]+"/>
      <xs:pattern value="conn:[a-zA-Z0-9.:\-_]+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="dialogLanguage.datatype">
     <xs:restriction base="xs:string">
      <xs:enumeration value="application/moml+xml"/>
      <xs:enumeration value="application/voicexml+xml"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="msmlEvent.datatype">
     <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="msmlSend.datatype">
```

```
     <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="msmlEventName.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern value="msml.dialog.exit"/>
      <xs:pattern value="msml.conf.asn"/>
      <xs:pattern value="msml.conf.nomedia"/>
      <xs:pattern value="msml.dialog.exit"/>
      <xs:pattern value="[a-zA-Z0-9.:_\-]+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="msmlTarget.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern
   value="conf:[a-zA-Z0-9.:_\-]+(/oper:[a-zA-Z0-9.:_\-]+|\*)*"/>
      <xs:pattern
   value="conn:[a-zA-Z0-9.:_\-]+(/oper:[a-zA-Z0-9.:_\-]+|\*)+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="msmlEventSource.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern value="conf:[a-zA-Z0-9.:_\-]+"/>
      <xs:pattern value="(conf:[a-zA-Z0-9.:_\-]+|conn:[a-zA-Z0-9.:_\-
      ]+)/dialog:[a-zA-Z0-9.:_\-]+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="msmlEventNameValue.datatype">
     <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="mark.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9.:\-_]+"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="boolean.datatype">
     <xs:restriction base="xs:string">
      <xs:enumeration value="true"/>
      <xs:enumeration value="false"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="posDuration.datatype">
     <xs:restriction base="xs:string">
      <xs:pattern value="(\+)?([0-9]*\.)?[0-9]+(ms|s)"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:schema>
```

16.2. MSML Conference Core Package

16.2.1. msml-conf-core.xsd

```
   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
              elementFormDefault="qualified"
              attributeFormDefault="unqualified">
      <xs:include schemaLocation="msml-core.xsd"/>
      <xs:include schemaLocation="msml-conf-core-datatypes.xsd"/>
   </xs:schema>
```

16.2.2. msml-conf-core-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core-datatypes.xsd"/>
 <xs:element name="createconference" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:all>
      <xs:element name="audiomix" type="audioMixType" minOccurs="0"/>
      <xs:element name="videolayout" type="videoLayoutType"
                  minOccurs="0"/>
      <xs:element name="reserve" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="resource" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded"/>
           </xs:sequence>
           <xs:attribute name="n" type="xs:positiveInteger"
                         default="1"/>
           <xs:anyAttribute namespace="##any"/>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
        <xs:attribute name="required" type="boolean.datatype"
                      default="true"/>
       </xs:complexType>
      </xs:element>
     </xs:all>
     <xs:attribute name="name" type="msmlInstanceID.datatype"/>
```

```
     <xs:attribute name="deletewhen" default="never">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="nomedia"/>
        <xs:enumeration value="nocontrol"/>
        <xs:enumeration value="never"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="term" type="boolean.datatype" default="true"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="modifyconference" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:all>
      <xs:element name="audiomix" type="audioMixType" minOccurs="0"/>
      <xs:element name="videolayout" type="videoLayoutType"
                  minOccurs="0"/>
     </xs:all>
     <xs:attribute name="id" type="confID.datatype" use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="destroyconference" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:all>
      <xs:element name="audiomix" type="basicAudioMixType"
                  minOccurs="0"/>
      <xs:element name="videolayout" type="basicVideoLayoutType"
                  minOccurs="0"/>
     </xs:all>
     <xs:attribute name="id" type="confID.datatype" use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="join" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:sequence>
```

```
      <xs:element name="stream" type="streamType" minOccurs="0"
                  maxOccurs="4"/>
     </xs:sequence>
     <xs:attribute name="id1" type="independentID.datatype"
                   use="required"/>
     <xs:attribute name="id2" type="independentID.datatype"
                   use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="modifystream" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:sequence>
      <xs:element name="stream" type="streamType" maxOccurs="4"/>
     </xs:sequence>
     <xs:attribute name="id1" type="independentID.datatype"
                   use="required"/>
     <xs:attribute name="id2" type="independentID.datatype"
                   use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="unjoin" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:sequence>
      <xs:element name="stream" type="basicStreamType" minOccurs="0"
                  maxOccurs="4"/>
     </xs:sequence>
     <xs:attribute name="id1" type="independentID.datatype"
                   use="required"/>
     <xs:attribute name="id2" type="independentID.datatype"
                   use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="monitor" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:attribute name="id1" type="connID.datatype" use="required"/>
     <xs:attribute name="id2" type="independentID.datatype"
```

```
                  use="required"/>
     <xs:attribute name="compressed" type="boolean.datatype"
                  default="false"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="confid" type="msmlResultSimpleType"
             substitutionGroup="msmlResultSimple"/>
 <xs:complexType name="basicStreamType">
  <xs:attribute name="dir">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="to-id1"/>
     <xs:enumeration value="from-id1"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="media">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="audio"/>
     <xs:enumeration value="video"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="compressed" type="boolean.datatype"/>
 </xs:complexType>
 <xs:complexType name="streamType">
  <xs:complexContent>
   <xs:extension base="basicStreamType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
     <xs:element name="gain">
      <xs:complexType>
       <xs:attribute name="amt" use="optional">
        <xs:simpleType>
         <xs:restriction base="xs:integer">
          <xs:minInclusive value="-96"/>
          <xs:maxInclusive value="96"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="agc" type="boolean.datatype"/>
       <xs:attribute name="tgtlvl" use="optional">
        <xs:simpleType>
         <xs:restriction base="xs:nonPositiveInteger">
          <xs:minInclusive value="-40"/>
          <xs:maxInclusive value="0"/>
```

```
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="maxgain" default="10">
       <xs:simpleType>
        <xs:restriction base="xs:nonNegativeInteger">
         <xs:minInclusive value="0"/>
         <xs:maxInclusive value="40"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
     </xs:complexType>
    </xs:element>
    <xs:element name="clamp">
     <xs:complexType>
      <xs:attribute name="dtmf" type="boolean.datatype"/>
      <xs:attribute name="tones" type="boolean.datatype"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="visual"/>
   </xs:choice>
   <xs:attribute name="preferred" type="boolean.datatype"
                 default="false"/>
   <xs:attribute name="display" type="xs:string"/>
   <xs:attribute name="override" type="boolean.datatype"
                 default="false"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="basicAudioMixType">
 <xs:attribute name="id" type="xs:string" use="optional"/>
 <xs:attribute name="samplerate" type="xs:positiveInteger"
               use="optional" default="8000"/>
</xs:complexType>
<xs:complexType name="audioMixType">
 <xs:complexContent>
  <xs:extension base="basicAudioMixType">
   <xs:all>
    <xs:element name="asn" minOccurs="0">
     <xs:complexType>
      <xs:attribute name="ri" type="posDuration.datatype"/>
      <xs:attribute name="asth" default="-96">
       <xs:simpleType>
        <xs:restriction base="xs:nonPositiveInteger">
         <xs:minInclusive value="-96"/>
         <xs:maxInclusive value="0"/>
        </xs:restriction>
       </xs:simpleType>
```

```
      </xs:attribute>
     </xs:complexType>
    </xs:element>
    <xs:element name="n-loudest" minOccurs="0">
     <xs:complexType>
      <xs:attribute name="n" type="xs:positiveInteger" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:all>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="basicVideoLayoutType">
 <xs:attribute name="id" type="xs:string" use="required"/>
 <xs:attribute name="type" type="xs:string" use="required"
               fixed="text/msml-basic-layout"/>
</xs:complexType>
<xs:complexType name="videoLayoutType">
 <xs:complexContent>
  <xs:extension base="basicVideoLayoutType">
   <xs:choice>
    <xs:element name="selector">
     <xs:complexType>
      <xs:complexContent>
       <xs:extension base="selectorType">
        <xs:choice>
         <xs:element name="root" type="rootType" minOccurs="0"/>
         <xs:element name="region" minOccurs="0">
          <xs:complexType>
           <xs:attribute name="id" type="xs:string" use="required"/>
           <xs:attribute name="left" type="xs:positiveInteger"/>
           <xs:attribute name="top" type="xs:positiveInteger"/>
           <xs:attribute name="relativeSize">
            <xs:simpleType>
             <xs:restriction base="xs:string">
              <xs:enumeration value="1/4"/>
              <xs:enumeration value="1/3"/>
              <xs:enumeration value="2/3"/>
              <xs:enumeration value="3/4"/>
              <xs:enumeration value="1"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:attribute>
           <xs:attribute name="priority">
            <xs:simpleType>
             <xs:restriction base="xs:float">
              <xs:minInclusive value="0"/>
              <xs:maxExclusive value="1"/>
```

```
              </xs:restriction>
             </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="title" type="xs:string"/>
            <xs:attribute name="titleTextColor" type="xs:string"/>
            <xs:attribute name="titleBackgroundColor" type="xs:string"/>
            <xs:attribute name="borderColor" type="xs:string"/>
            <xs:attribute name="borderWidth" type="xs:positiveInteger"/>
            <xs:attribute name="logo" type="xs:anyURI"/>
           </xs:complexType>
          </xs:element>
         </xs:choice>
        </xs:extension>
       </xs:complexContent>
      </xs:complexType>
     </xs:element>
     <xs:element name="root" type="rootType"/>
     <xs:element name="region" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
       <xs:complexContent>
        <xs:extension base="regionType"/>
       </xs:complexContent>
      </xs:complexType>
     </xs:element>
    </xs:choice>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
 <xs:complexType name="regionType">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="left" type="xs:positiveInteger"/>
  <xs:attribute name="top" type="xs:positiveInteger"/>
  <xs:attribute name="relativeSize">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="1/4"/>
     <xs:enumeration value="1/3"/>
     <xs:enumeration value="2/3"/>
     <xs:enumeration value="3/4"/>
     <xs:enumeration value="1"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="priority">
   <xs:simpleType>
    <xs:restriction base="xs:float">
     <xs:minInclusive value="0"/>
     <xs:maxExclusive value="1"/>
```

```
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="title" type="xs:string"/>
  <xs:attribute name="titleTextColor" type="xs:string"/>
  <xs:attribute name="titleBackgroundColor" type="xs:string"/>
  <xs:attribute name="borderColor" type="xs:string"/>
  <xs:attribute name="borderWidth" type="xs:positiveInteger"/>
  <xs:attribute name="logo" type="xs:anyURI"/>
 </xs:complexType>
 <xs:complexType name="selectorType">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="method" use="required">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="vas"/>
     <xs:enumeration value="sequence"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="status" default="active">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="active"/>
     <xs:enumeration value="disabled"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="si" type="posDuration.datatype" default="1s"/>
  <xs:attribute name="blankothers" type="xs:boolean" default="false"/>
  <xs:attribute name="speakersees" default="current">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="current"/>
     <xs:enumeration value="previous"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>
 <xs:complexType name="rootType">
  <xs:attribute name="size" default="CIF">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="16CIF"/>
     <xs:enumeration value="4CIF"/>
     <xs:enumeration value="CIF"/>
     <xs:enumeration value="QCIF"/>
    </xs:restriction>
```

```
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="backgroundcolor" type="xs:string"
                default="black"/>
  <xs:attribute name="backgroundimage" type="xs:anyURI"/>
 </xs:complexType>
 <xs:simpleType name="confclass.datatype">
  <xs:restriction base="xs:string">
   <xs:enumeration value="standard"/>
   <xs:enumeration value="preferred"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="conferenceType.datatype">
  <xs:restriction base="xs:string">
   <xs:enumeration value="audio.basic"/>
   <xs:enumeration value="audio.advanced"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="duplex.datatype">
  <xs:restriction base="xs:string">
   <xs:enumeration value="half"/>
   <xs:enumeration value="full"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

16.3. MSML Dialog Packages

16.3.1. msml-dialog-core.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
</xs:schema>
```

16.3.2. msml-dialog-core-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core-datatypes.xsd"/>
 <xs:group name="momlRequest">
  <xs:choice>
   <xs:group ref="executeType"/>
```

```
   <xs:group ref="sendType"/>
  </xs:choice>
 </xs:group>
 <xs:element name="dialogstart" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:choice>
      <xs:group ref="momlRequest" minOccurs="0"/>
     </xs:choice>
     <xs:attribute name="target" type="independentID.datatype"
                   use="required"/>
     <xs:attribute name="type" type="dialogLanguage.datatype"
                   use="required"/>
     <xs:attribute name="name" type="msmlInstanceID.datatype"/>
     <xs:attribute name="src" type="xs:anyURI" use="optional"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="dialogend" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:attribute name="id" type="dialogID.datatype" use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="dialogid" type="msmlResultSimpleType"
             substitutionGroup="msmlResultSimple"/>
 <xs:group name="executeType">
  <xs:choice>
   <xs:element ref="primitive" maxOccurs="unbounded"/>
   <xs:element ref="control" maxOccurs="unbounded"/>
  </xs:choice>
 </xs:group>
 <xs:element name="primitive" type="primitiveType" abstract="true"/>
 <xs:complexType name="primitiveType">
  <xs:attribute name="id" type="momlID.datatype"/>
 </xs:complexType>
 <xs:element name="control" abstract="true"/>
 <xs:group name="sendType">
  <xs:choice>
   <xs:choice>
    <xs:element name="exit" type="exitType"/>
    <xs:element name="disconnect" type="exitType"/>
   </xs:choice>
```

```
   <xs:sequence>
    <xs:element ref="send" maxOccurs="unbounded"/>
    <xs:choice minOccurs="0">
     <xs:element name="exit" type="exitType"/>
     <xs:element name="disconnect" type="exitType"/>
    </xs:choice>
   </xs:sequence>
  </xs:choice>
 </xs:group>
 <xs:element name="send">
  <xs:complexType>
   <xs:attribute name="event" type="momlEvent.datatype" use="required"/>
   <xs:attribute name="target" type="momlTarget.datatype"
                 use="required"/>
   <xs:attribute name="namelist" type="momlNamelist.datatype"/>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="exitType">
  <xs:attribute name="namelist" type="momlNamelist.datatype"/>
 </xs:complexType>
 <xs:simpleType name="momlID.datatype">
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z0-9][a-zA-Z0-9._\-]*"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="momlEvent.datatype">
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z0-9][a-zA-Z0-9._\-]*"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="momlNamelist.datatype">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
 <xs:simpleType name="dtmfDigits.datatype">
  <xs:restriction base="xs:string">
   <xs:pattern value="[0-9#*]+"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="iterate.datatype">
  <xs:union memberTypes="xs:positiveInteger">
   <xs:simpleType>
    <xs:restriction base="xs:negativeInteger">
     <xs:minInclusive value="-1"/>
    </xs:restriction>
   </xs:simpleType>
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="forever"/>
```

```
    </xs:restriction>
   </xs:simpleType>
  </xs:union>
 </xs:simpleType>
 <xs:simpleType name="momlTarget.datatype">
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-zA-Z0-9][a-zA-Z0-9._\-]*"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="duration.datatype">
  <xs:restriction base="xs:string">
   <xs:pattern value="(\+|\-)?([0-9]*\.)?[0-9]+(ms|s)"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

16.3.3. msml-dialog-base.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-base-datatypes.xsd"/>
</xs:schema>
```

16.3.4. msml-dialog-base-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:import namespace="http://www.w3.org/XML/1998/namespace"
            schemaLocation="http://www.w3.org/2001/xml.xsd"/>
 <xs:element name="play" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:sequence>
      <xs:choice maxOccurs="unbounded">
       <xs:element name="audio" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
         <xs:attribute name="uri" type="xs:anyURI" use="required"/>
         <xs:attribute name="iterate" type="iterate.datatype"
                       default="1"/>
         <xs:attribute name="format" type="xs:string" use="optional"/>
```

```
        <xs:attribute name="audiosamplerate" type="xs:positiveInteger"
                      use="optional"/>
        <xs:attribute name="audiosamplesize" type="xs:positiveInteger"
                      use="optional"/>
        <xs:attribute ref="xml:lang"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="video" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:attribute name="uri" type="xs:anyURI" use="required"/>
        <xs:attribute name="iterate" type="iterate.datatype"
                      use="optional" default="1"/>
        <xs:attribute name="format" type="xs:string" use="optional"/>
        <xs:attribute name="audiosamplerate" type="xs:positiveInteger"
                      use="optional"/>
        <xs:attribute name="audiosamplesize" type="xs:positiveInteger"
                      use="optional"/>
        <xs:attribute name="codecconfig" type="xs:string"
                      use="optional"/>
        <xs:attribute name="profile" type="xs:string" use="optional"/>
        <xs:attribute name="level" type="xs:string" use="optional"/>
        <xs:attribute name="imagewidth" type="xs:positiveInteger"
                      use="optional"/>
        <xs:attribute name="imageheight" type="xs:positiveInteger"
                      use="optional"/>
        <xs:attribute name="maxbitrate" type="xs:positiveInteger"
                      use="optional"/>
        <xs:attribute name="framerate" type="xs:positiveInteger"
                      use="optional"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="media" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="audio" minOccurs="0">
          <xs:complexType>
           <xs:attribute name="uri" type="xs:anyURI" use="required"/>
           <xs:attribute name="iterate" type="iterate.datatype"
                         default="1"/>
           <xs:attribute name="format" type="xs:string"
                         use="optional"/>
           <xs:attribute name="audiosamplerate"
                         type="xs:positiveInteger" use="optional"/>
           <xs:attribute name="audiosamplesize"
                         type="xs:positiveInteger" use="optional"/>
           <xs:attribute ref="xml:lang"/>
          </xs:complexType>
         </xs:element>
```

```
          <xs:element name="video" minOccurs="0">
           <xs:complexType>
            <xs:attribute name="uri" type="xs:anyURI" use="required"/>
            <xs:attribute name="iterate" type="iterate.datatype"
                          use="optional" default="1"/>
            <xs:attribute name="format" type="xs:string"
                          use="optional"/>
            <xs:attribute name="audiosamplerate"
                          type="xs:positiveInteger" use="optional"/>
            <xs:attribute name="audiosamplesize"
                          type="xs:positiveInteger" use="optional"/>
            <xs:attribute name="codecconfig" type="xs:string"
                          use="optional"/>
            <xs:attribute name="profile" type="xs:string"
                          use="optional"/>
            <xs:attribute name="level" type="xs:string" use="optional"/>
            <xs:attribute name="imagewidth" type="xs:positiveInteger"
                          use="optional"/>
            <xs:attribute name="imageheight" type="xs:positiveInteger"
                          use="optional"/>
            <xs:attribute name="maxbitrate" type="xs:positiveInteger"
                          use="optional"/>
            <xs:attribute name="framerate" type="xs:positiveInteger"
                          use="optional"/>
           </xs:complexType>
          </xs:element>
         </xs:choice>
        </xs:complexType>
       </xs:element>
       <xs:element ref="smedia" minOccurs="0" maxOccurs="unbounded"/>
      </xs:choice>
      <xs:choice minOccurs="0">
       <xs:element name="playexit">
        <xs:complexType>
         <xs:group ref="sendType"/>
        </xs:complexType>
       </xs:element>
      </xs:choice>
     </xs:sequence>
     <xs:attribute name="interval" type="posDuration.datatype"
                   use="optional"/>
     <xs:attribute name="iterate" type="iterate.datatype" use="optional"
                   default="1"/>
     <xs:attribute name="offset" type="duration.datatype"
                   use="optional"/>
     <xs:attribute name="initial" use="optional" default="generate">
      <xs:simpleType>
       <xs:restriction base="xs:string">
```

```
        <xs:enumeration value="generate"/>
        <xs:enumeration value="suspend"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="maxtime" type="posDuration.datatype"
                   use="optional"/>
     <xs:attribute name="skip" type="duration.datatype" use="optional"
                   default="3s"/>
     <xs:attribute name="barge" type="boolean.datatype" use="optional"
                   default="false"/>
     <xs:attribute name="cleardb" type="boolean.datatype" use="optional"
                   default="false"/>
     <xs:attribute ref="xml:lang"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="record" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:choice minOccurs="0">
      <xs:element ref="play" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="tonegen" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="recordexit">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
     </xs:choice>
     <xs:attribute name="append" type="boolean.datatype" use="optional"
                   default="false"/>
     <xs:attribute name="dest" type="xs:anyURI" use="optional"/>
     <xs:attribute name="audiodest" type="xs:anyURI" use="optional"/>
     <xs:attribute name="videodest" type="xs:anyURI" use="optional"/>
     <xs:attribute name="format" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:string"/>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="codecconfig" use="optional">
      <xs:simpleType>
       <xs:restriction base="xs:string"/>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="audiosamplerate" type="xs:positiveInteger"
                   use="optional"/>
```

```
     <xs:attribute name="audiosamplesize" type="xs:positiveInteger"
                   use="optional"/>
     <xs:attribute name="profile" use="optional">
      <xs:simpleType>
       <xs:restriction base="xs:string"/>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="level" use="optional">
      <xs:simpleType>
       <xs:restriction base="xs:string"/>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="imagewidth" type="xs:positiveInteger"
                   use="optional"/>
     <xs:attribute name="imageheight" type="xs:positiveInteger"
                   use="optional"/>
     <xs:attribute name="maxbitrate" type="xs:positiveInteger"
                   use="optional"/>
     <xs:attribute name="framerate" type="xs:positiveInteger"
                   use="optional"/>
     <xs:attribute name="maxtime" type="posDuration.datatype"
                   use="required"/>
     <xs:attribute name="initial" use="optional" default="create">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="create"/>
        <xs:enumeration value="suspend"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="prespeech" type="posDuration.datatype"
                   use="optional" default="0s"/>
     <xs:attribute name="postspeech" type="posDuration.datatype"
                   use="optional" default="0s"/>
     <xs:attribute name="termkey" use="optional">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:pattern value="[0-9#*ABCD]"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="dtmf" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
```

```
    <xs:extension base="primitiveType">
     <xs:sequence>
      <xs:element name="pattern" maxOccurs="unbounded">
       <xs:complexType>
        <xs:group ref="sendType"/>
        <xs:attribute name="digits" type="xs:string" use="required"/>
        <xs:attribute name="format">
         <xs:simpleType>
          <xs:restriction base="xs:string">
           <xs:enumeration value="mgcp"/>
           <xs:enumeration value="megaco"/>
           <xs:enumeration value="moml+digits"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="iterate" type="iterate.datatype"
                      default="1"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="detect" minOccurs="0">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="noinput" type="iterateSendType" minOccurs="0"/>
      <xs:element name="nomatch" type="iterateSendType" minOccurs="0"/>
      <xs:element name="dtmfexit" minOccurs="0">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
      <xs:element ref="play" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="cleardb" type="boolean.datatype"
                   default="true"/>
     <xs:attribute name="fdt" type="posDuration.datatype" default="0s"/>
     <xs:attribute name="idt" type="posDuration.datatype" default="4s"/>
     <xs:attribute name="edt" type="posDuration.datatype" default="4s"/>
     <xs:attribute name="starttimer" type="boolean.datatype"
                   default="false"/>
     <xs:attribute name="iterate" type="iterate.datatype" default="1"/>
     <xs:attribute name="ldd" type="posDuration.datatype" default="0s"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="collect" substitutionGroup="primitive">
  <xs:complexType>
```

```
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:sequence>
      <xs:element name="pattern" maxOccurs="unbounded">
       <xs:complexType>
        <xs:group ref="sendType"/>
        <xs:attribute name="digits" type="xs:string" use="required"/>
        <xs:attribute name="format">
         <xs:simpleType>
          <xs:restriction base="xs:string">
           <xs:enumeration value="mgcp"/>
           <xs:enumeration value="megaco"/>
           <xs:enumeration value="moml+digits"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="iterate" type="iterate.datatype"
                      default="1"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="detect" minOccurs="0">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="noinput" type="iterateSendType" minOccurs="0"/>
      <xs:element name="nomatch" type="iterateSendType" minOccurs="0"/>
      <xs:element name="dtmfexit" minOccurs="0">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
      <xs:element ref="play" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="cleardb" type="boolean.datatype"
                   default="true"/>
     <xs:attribute name="fdt" type="posDuration.datatype" default="0s"/>
     <xs:attribute name="idt" type="posDuration.datatype" default="4s"/>
     <xs:attribute name="edt" type="posDuration.datatype" default="4s"/>
     <xs:attribute name="starttimer" type="boolean.datatype"
                   default="false"/>
     <xs:attribute name="iterate" type="iterate.datatype" default="1"/>
     <xs:attribute name="ldd" type="posDuration.datatype"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="dtmfgen" substitutionGroup="primitive">
```

```
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:choice minOccurs="0">
      <xs:element name="dtmfgenexit">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
     </xs:choice>
     <xs:attribute name="level" use="optional" default="-6">
      <xs:simpleType>
       <xs:restriction base="xs:nonPositiveInteger">
        <xs:maxInclusive value="0"/>
        <xs:minInclusive value="-96"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="digits" type="dtmfDigits.datatype"
                   use="required"/>
     <xs:attribute name="dur" type="posDuration.datatype" use="optional"
                   default="100ms"/>
     <xs:attribute name="interval" type="posDuration.datatype"
                   use="optional" default="100ms"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="tonegen" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:choice minOccurs="0">
      <xs:element name="tonegenexit" minOccurs="0">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="tone" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="tone1">
          <xs:complexType>
           <xs:attribute name="freq" use="required">
            <xs:simpleType>
             <xs:restriction base="xs:unsignedInt">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="3999"/>
```

```
            </xs:restriction>
           </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="atten" use="required">
           <xs:simpleType>
            <xs:restriction base="xs:nonPositiveInteger">
             <xs:minInclusive value="-96"/>
             <xs:maxInclusive value="0"/>
            </xs:restriction>
           </xs:simpleType>
          </xs:attribute>
         </xs:complexType>
        </xs:element>
        <xs:element name="tone2">
         <xs:complexType>
          <xs:attribute name="freq" use="required">
           <xs:simpleType>
            <xs:restriction base="xs:unsignedInt">
             <xs:minInclusive value="0"/>
             <xs:maxInclusive value="3999"/>
            </xs:restriction>
           </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="atten" use="required">
           <xs:simpleType>
            <xs:restriction base="xs:nonPositiveInteger">
             <xs:minInclusive value="-96"/>
             <xs:maxInclusive value="0"/>
            </xs:restriction>
           </xs:simpleType>
          </xs:attribute>
         </xs:complexType>
        </xs:element>
        <xs:element name="silence" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
          <xs:attribute name="duration" type="duration.datatype"
                        use="required"/>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
       <xs:attribute name="duration" use="required">
        <xs:simpleType>
         <xs:restriction base="duration.datatype"/>
        </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="iterate" type="iterate.datatype"
                     use="optional" default="1"/>
      </xs:complexType>
```

```
      </xs:element>
      <xs:element name="silence" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:attribute name="duration" type="duration.datatype"
                      use="required"/>
       </xs:complexType>
      </xs:element>
     </xs:choice>
     <xs:attribute name="iterate" type="iterate.datatype" use="optional"
                   default="1"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="iterateSendType">
  <xs:group ref="sendType"/>
  <xs:attribute name="iterate" type="iterate.datatype" default="1"/>
 </xs:complexType>
 <xs:element name="smedia" type="smediaType" abstract="true"/>
 <xs:complexType name="smediaType">
  <xs:attribute ref="xml:lang"/>
  <xs:attribute name="iterate" type="iterate.datatype"/>
 </xs:complexType>
 <xs:element name="var" substitutionGroup="smedia">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="smediaType">
     <xs:attribute name="type" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="date"/>
        <xs:enumeration value="digits"/>
        <xs:enumeration value="duration"/>
        <xs:enumeration value="month"/>
        <xs:enumeration value="money"/>
        <xs:enumeration value="number"/>
        <xs:enumeration value="silence"/>
        <xs:enumeration value="time"/>
        <xs:enumeration value="weekday"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="subtype" type="xs:string" use="optional"/>
     <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:extension>
   </xs:complexContent>
```

```
  </xs:complexType>
 </xs:element>
</xs:schema>
```

16.3.5. msml-dialog-transform.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-transform-datatypes.xsd"/>
</xs:schema>
```

16.3.6. msml-dialog-transform-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:import namespace="http://www.w3.org/XML/1998/namespace"
            schemaLocation="http://www.w3.org/2001/xml.xsd"/>
 <xs:element name="vad" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:all>
      <xs:element name="voice" type="vadPatternType" minOccurs="0"/>
      <xs:element name="silence" type="vadPatternType" minOccurs="0"/>
      <xs:element name="tvoice" type="vadPatternType" minOccurs="0"/>
      <xs:element name="tsilence" type="vadPatternType" minOccurs="0"/>
     </xs:all>
     <xs:attribute name="starttimer" type="boolean.datatype"
                   default="false"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="gain" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:attribute name="incr" default="3">
      <xs:simpleType>
       <xs:restriction base="xs:positiveInteger">
        <xs:maxInclusive value="96"/>
```

```
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="amt" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:integer">
        <xs:minInclusive value="-96"/>
        <xs:maxInclusive value="96"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="agc" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:attribute name="tgtlvl" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:nonPositiveInteger">
        <xs:minInclusive value="-40"/>
        <xs:maxInclusive value="0"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="maxgain" default="10">
      <xs:simpleType>
       <xs:restriction base="xs:nonNegativeInteger">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="40"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="gate" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:attribute name="initial" default="pass">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="pass"/>
        <xs:enumeration value="halt"/>
```

```
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="clamp" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType"/>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="relay" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType"/>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="vadPatternType">
  <xs:group ref="sendType"/>
  <xs:attribute name="iterate" type="iterate.datatype" default="1"/>
  <xs:attribute name="len" type="posDuration.datatype" use="required"/>
  <xs:attribute name="sen" type="posDuration.datatype" use="optional"/>
 </xs:complexType>
</xs:schema>
```

16.3.7. msml-dialog-group.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-base-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-group-datatypes.xsd"/>
</xs:schema>
```

16.3.8. msml-dialog-group-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="unqualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core-datatypes.xsd"/>
```

```
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-base-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-transform-datatypes.xsd"/>
 <xs:element name="group" substitutionGroup="control">
  <xs:complexType>
   <xs:sequence>
    <xs:group ref="executeType"/>
    <xs:element name="groupexit" minOccurs="0">
     <xs:complexType>
      <xs:group ref="sendType"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="id" type="momlID.datatype"/>
   <xs:attribute name="topology" use="required">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="serial"/>
      <xs:enumeration value="parallel"/>
      <xs:enumeration value="fullduplex"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

16.3.9. msml-dialog-speech.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-base-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-speech-datatypes.xsd"/>
</xs:schema>
```

16.3.10. msml-dialog-speech-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-base-datatypes.xsd"/>
 <xs:include schemaLocation="http://www.w3.org/TR/2002/WD-speech-
```

```
            synthesis-20020405/synthesis-core.xsd"/>
 <xs:include schemaLocation="http://www.w3.org/TR/speech-
            grammar/grammar-core.xsd"/>
 <xs:element name="speech" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:sequence>
      <xs:element name="grammar" maxOccurs="unbounded">
       <xs:complexType>
        <xs:complexContent>
         <xs:extension base="grammar">
          <xs:choice>
           <xs:element name="match" type="iterateSendType"
                       minOccurs="0"/>
          </xs:choice>
          <xs:attribute name="uri" type="xs:anyURI"/>
          <xs:attribute name="iterate" type="iterate.datatype"
                        default="1"/>
         </xs:extension>
        </xs:complexContent>
       </xs:complexType>
      </xs:element>
      <xs:element name="noinput" type="iterateSendType" minOccurs="0"/>
      <xs:element name="nomatch" type="iterateSendType" minOccurs="0"/>
      <xs:element name="speechexit" minOccurs="0">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
     <xs:attribute name="noint" type="posDuration.datatype"/>
     <xs:attribute name="norect" type="posDuration.datatype"/>
     <xs:attribute name="spcmplt" type="posDuration.datatype"/>
     <xs:attribute name="confidence">
      <xs:simpleType>
       <xs:restriction base="xs:positiveInteger">
        <xs:maxInclusive value="100"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="sens" type="xs:positiveInteger"/>
     <xs:attribute name="starttimer" type="boolean.datatype"
                   default="false"/>
     <xs:attribute name="iterate" type="iterate.datatype" default="1"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
```

```
 </xs:element>
 <xs:element name="tts" type="smediaType" substitutionGroup="smedia"/>
</xs:schema>
```

16.3.11. msml-dialog-fax-detect.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-fax-detect-datatypes.xsd"/>
</xs:schema>
```

16.3.12. msml-dialog-fax-detect-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:element name="faxdetect" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:choice minOccurs="0">
      <xs:element name="faxdetectexit">
       <xs:complexType>
        <xs:group ref="sendType"/>
       </xs:complexType>
      </xs:element>
     </xs:choice>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

16.3.13. msml-dialog-fax-sendrecv.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-dialog-fax-sendrecv-datatypes.xsd"/>
```

```
</xs:schema>
```

16.3.14. msml-dialog-fax-sendrecv-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-dialog-core-datatypes.xsd"/>
 <xs:element name="faxsend" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:sequence>
      <xs:element name="sendobj" type="sendobjType" minOccurs="0"
          maxOccurs="unbounded"/>
      <xs:element name="hdrfooter" type="hdrfooterType" minOccurs="0"/>
      <xs:element name="rxpoll" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="rcvobj" type="rcvobjType"
                     maxOccurs="unbounded"/>
         <xs:element name="hdrfooter" type="hdrfooterType"
                     minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="rmtid" type="faxid.datatype"
                      use="required"/>
       </xs:complexType>
      </xs:element>
      <xs:group ref="faxstatusrequest"/>
     </xs:sequence>
     <xs:attribute name="lclid" type="faxid.datatype" use="optional"/>
     <xs:attribute name="minspeed" type="faxspeed.datatype"
                   use="optional"/>
     <xs:attribute name="maxspeed" type="faxspeed.datatype"
                   use="optional"/>
     <xs:attribute name="ecm" type="boolean.datatype" use="optional"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="faxrecv" substitutionGroup="primitive">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="primitiveType">
     <xs:sequence>
      <xs:element name="rcvobj" type="rcvobjType" minOccurs="0"
                  maxOccurs="unbounded"/>
```

```
      <xs:element name="hdrfooter" type="hdrfooterType" minOccurs="0"/>
      <xs:element name="txpoll" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="sendobj" type="sendobjType"
                     maxOccurs="unbounded"/>
         <xs:element name="hdrfooter" type="hdrfooterType"
                     minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="rmtid" type="faxid.datatype"/>
       </xs:complexType>
      </xs:element>
      <xs:group ref="faxstatusrequest"/>
     </xs:sequence>
     <xs:attribute name="lclid" type="faxid.datatype" use="optional"/>
     <xs:attribute name="ecm" type="boolean.datatype" default="true"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:group name="faxstatusrequest">
  <xs:sequence>
   <xs:element name="faxstart" minOccurs="0"/>
   <xs:element name="faxnegotiate" minOccurs="0"/>
   <xs:element name="faxpagedone" minOccurs="0"/>
   <xs:element name="faxobjectdone" minOccurs="0"/>
   <xs:element name="faxopcomplete" minOccurs="0"/>
   <xs:element name="faxpollstart" minOccurs="0"/>
  </xs:sequence>
 </xs:group>
 <xs:complexType name="hdrfooterType">
  <xs:choice>
   <xs:element name="format" type="xs:string" minOccurs="0"
               maxOccurs="unbounded"/>
  </xs:choice>
  <xs:attribute name="type" type="hdrfooter.datatype"/>
  <xs:attribute name="style" type="hdrfooterstyle.datatype"/>
 </xs:complexType>
 <xs:complexType name="formatType">
  <xs:simpleContent>
   <xs:extension base="xs:string">
    <xs:attribute name="style">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="append"/>
       <xs:enumeration value="overlay"/>
       <xs:enumeration value="replace"/>
      </xs:restriction>
```

```
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
 <xs:complexType name="rcvobjType">
  <xs:attribute name="objuri" type="xs:anyURI" use="required"/>
  <xs:attribute name="maxpages" type="xs:positiveInteger"/>
 </xs:complexType>
 <xs:complexType name="sendobjType">
  <xs:attribute name="objuri" type="xs:anyURI" use="required"/>
  <xs:attribute name="startpage" type="xs:positiveInteger"/>
  <xs:attribute name="pagecount" type="xs:positiveInteger"/>
 </xs:complexType>
 <xs:simpleType name="faxid.datatype">
  <xs:restriction base="xs:string">
   <xs:pattern value="[0-9+*- ]{20}"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="faxspeed.datatype">
  <xs:restriction base="xs:string">
   <xs:enumeration value="2400"/>
   <xs:enumeration value="4800"/>
   <xs:enumeration value="7200"/>
   <xs:enumeration value="9600"/>
   <xs:enumeration value="12000"/>
   <xs:enumeration value="14400"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="hdrfooter.datatype">
  <xs:restriction base="xs:string">
   <xs:enumeration value="header"/>
   <xs:enumeration value="footer"/>
   <xs:enumeration value="autohdr"/>
   <xs:enumeration value="nohdr"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="hdrfooterstyle.datatype">
  <xs:restriction base="xs:string">
   <xs:enumeration value="append"/>
   <xs:enumeration value="overlay"/>
   <xs:enumeration value="replace"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

16.4. MSML Audit Packages

16.4.1. msml-audit-core.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-core-datatypes.xsd"/>
</xs:schema>
```

16.4.2. msml-audit-core-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core-datatypes.xsd"/>
 <xs:element name="audit" substitutionGroup="msmlRequest">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlRequestType">
     <xs:attribute name="queryid" type="auditQueryId.datatype"
                   use="required"/>
     <xs:attribute name="statelist" type="auditStateList.datatype"
                   use="optional"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="auditresult" substitutionGroup="msmlResultComplex">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="msmlResultComplexType">
    <xs:choice maxOccurs="unbounded">
     <xs:element ref="stateParameter"/>
     <xs:element ref="stateParameterSimple"/>
     </xs:choice>
     <xs:attribute name="targetid" type="independentID.datatype"
                   use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="stateParameter" type="stateParameterType"
             abstract="true"/>
```

```
 <xs:element name="stateParameterSimple" type="stateParameterSimpleType"
             abstract="true"/>
 <xs:complexType name="stateParameterType"/>
 <xs:simpleType name="stateParameterSimpleType">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
 <xs:simpleType name="auditQueryId.datatype">
  <xs:restriction base="xs:string">
   <xs:pattern value="conf:[a-zA-Z0-9.:\-_]+"/>
   <xs:pattern value="conn:[a-zA-Z0-9.:\-_]+"/>
   <xs:pattern value="conf:\*"/>
   <xs:pattern value="conn:\*"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="auditStateList.datatype">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
</xs:schema>
```

16.4.3. msml-audit-conf.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-dialog-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-stream-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-conf-datatypes.xsd"/>
</xs:schema>
```

16.4.4. msml-audit-conf-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-conf-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-core-datatypes.xsd"/>
 <xs:element name="confconfig" substitutionGroup="stateParameter">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="stateParameterType">
     <xs:sequence>
      <xs:element name="audiomix" type="audioMixType" minOccurs="0"
                  maxOccurs="unbounded"/>
```

```
      <xs:element name="videolayout" type="videoLayoutType"
                  minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="controller" type="connID.datatype"
                  minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="deletewhen" use="optional" default="never">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="nomedia"/>
        <xs:enumeration value="nocontrol"/>
        <xs:enumeration value="never"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="term" type="boolean.datatype" use="optional"
                   default="true"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

16.4.5. msml-audit-conn.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-core.xsd"/>
 <xs:include schemaLocation="msml-audit-core-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-dialog-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-stream-datatypes.xsd"/>
 <xs:include schemaLocation="msml-audit-conn-datatypes.xsd"/>
</xs:schema>
```

16.4.6. msml-audit-conn-datatypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-audit-core-datatypes.xsd"/>
 <xs:element name="sipdialog" substitutionGroup="stateParameter">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="stateParameterType">
     <xs:sequence>
      <xs:element name="localseq" type="xs:integer" minOccurs="0"/>
```

```
      <xs:element name="remoteseq" type="xs:int" minOccurs="0"/>
      <xs:element name="localuri" type="xs:string" minOccurs="0"/>
      <xs:element name="remoteuri" type="xs:string" minOccurs="0"/>
      <xs:element name="remotetarget" type="xs:string" minOccurs="0"/>
      <xs:element name="routeset" type="xs:string" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="callid" type="xs:string" use="required"/>
     <xs:attribute name="localtag" type="xs:string" use="required"/>
     <xs:attribute name="remotetag" type="xs:string" use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="localsdp" type="stateParameterSimpleType"
             substitutionGroup="stateParameterSimple"/>
 <xs:element name="remotesdp" type="stateParameterSimpleType"
             substitutionGroup="stateParameterSimple"/>
</xs:schema>
```

16.4.7. msml-audit-dialog-datatypes.xsd

Audit Dialog functionality requires use of either the Audit Conf Package or the Audit Conn Package.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-audit-core-datatypes.xsd"/>
 <xs:element name="dialog" substitutionGroup="stateParameter">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="stateParameterType">
     <xs:sequence>
      <xs:element name="duration" type="xs:positiveInteger"
                  minOccurs="0"/>
      <xs:element name="primitive" minOccurs="0">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:pattern value="play"/>
         <xs:pattern value="dtmf"/>
         <xs:pattern value="collect"/>
         <xs:pattern value="dtmfgen"/>
         <xs:pattern value="tonegen"/>
         <xs:pattern value="record"/>
         <xs:pattern value="none"/>
        </xs:restriction>
       </xs:simpleType>
```

```
        </xs:element>
        <xs:element name="controller" type="connID.datatype"
                    minOccurs="0"/>
       </xs:sequence>
       <xs:attribute name="name" type="msmlInstanceID.datatype"
                     use="required"/>
       <xs:attribute name="src" type="xs:anyURI" use="optional"/>
       <xs:attribute name="type" type="dialogLanguage.datatype"
                     use="required"/>
      </xs:extension>
     </xs:complexContent>
    </xs:complexType>
   </xs:element>
  </xs:schema>
```

16.4.8. msml-audit-stream-datatypes.xsd

Audit Stream functionality requires use of either the Audit Conf Package or the Audit Conn Package.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
 <xs:include schemaLocation="msml-audit-core-datatypes.xsd"/>
 <xs:element name="stream" substitutionGroup="stateParameter">
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="stateParameterType">
     <xs:all>
      <xs:element name="clamp" minOccurs="0">
       <xs:complexType>
        <xs:attribute name="dtmf" type="boolean.datatype"/>
        <xs:attribute name="tones" type="boolean.datatype"/>
       </xs:complexType>
      </xs:element>
      <xs:element name="gain" minOccurs="0">
       <xs:complexType>
        <xs:attribute name="amt" use="optional">
         <xs:simpleType>
          <xs:restriction base="xs:integer">
           <xs:minInclusive value="-96"/>
           <xs:maxInclusive value="96"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="agc" type="boolean.datatype"/>
        <xs:attribute name="tgtlvl" use="optional">
```

```
            <xs:simpleType>
             <xs:restriction base="xs:nonPositiveInteger">
              <xs:minInclusive value="-40"/>
              <xs:maxInclusive value="0"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:attribute>
           <xs:attribute name="maxgain" default="10">
            <xs:simpleType>
             <xs:restriction base="xs:nonNegativeInteger">
              <xs:minInclusive value="0"/>
              <xs:maxInclusive value="40"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:attribute>
          </xs:complexType>
         </xs:element>
         <xs:element name="visual" minOccurs="0"/>
        </xs:all>
        <xs:attribute name="joinwith" type="independentID.datatype"
                      use="required"/>
        <xs:attribute name="media" use="required">
         <xs:simpleType>
          <xs:restriction base="xs:string">
           <xs:pattern value="audio"/>
           <xs:pattern value="video"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="dir" use="required">
         <xs:simpleType>
          <xs:restriction base="xs:string">
           <xs:pattern value="from"/>
           <xs:pattern value="to"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="compressed" type="boolean.datatype"/>
        <xs:attribute name="preferred" type="boolean.datatype"
                      default="false"/>
        <xs:attribute name="display" type="xs:string"/>
        <xs:attribute name="override" type="boolean.datatype"
                      default="false"/>
       </xs:extension>
      </xs:complexContent>
     </xs:complexType>
    </xs:element>
   </xs:schema>
```

17. Security Considerations

MSML being an XML-based language, security considerations as defined by RFC 3023 [i2] are applicable.

Media server interfaces driven using MSML are under the explicit control of a SIP application server. SIP call legs are used to deliver XML-based MSML transactions to the media server. The security and integrity of MSML transactions, whenever required, SHOULD use sips: and TLS for encryption and authentication of the SIP control channel used to carry MSML payloads. Further information related to security, privacy, and integrity of MSML media types is described in the IANA Considerations section.

Media streams, such as audio/video, MAY optionally be protected, encrypted/decrypted, and authenticated, utilizing Secure Real Time Protocol (SRTP), wherever media stream security is required. Media negotiation establishes the required level of security and is initiated by the clients, which is outside the scope of the control interface specified by MSML.

18. IANA Considerations

18.1. IANA Registrations for 'application' MIME Media Type

The following registrations have been made:

Type Name: "application"

Subtype names:

'application/vnd.radisys.msml+xml',

'application/vnd.radisys.moml+xml',

'application/vnd.radisys.msml-conf+xml',

'application/vnd.radisys.msml-dialog+xml',

'application/vnd.radisys.msml-dialog-base+xml',

'application/vnd.radisys.msml-dialog-group+xml',

'application/vnd.radisys.msml-dialog-speech+xml',

'application/vnd.radisys.msml-dialog-transform+xml',

'application/vnd.radisys.msml-dialog-fax-detect+xml',

'application/vnd.radisys.msml-dialog-fax-sendrecv+xml',

'application/vnd.radisys.msml-audit+xml',

'application/vnd.radisys.msml-audit-conf+xml',

'application/vnd.radisys.msml-audit-conn+xml',

'application/vnd.radisys.msml-audit-dialog+xml',

'application/vnd.radisys.msml-audit-stream+xml'

Required parameters: none

Optional parameters: charset

charset semantics as specified in RFC 3023 [i2] for "application/xml" media type.

Encoding considerations:

As specified in RFC 3023 [i2].

Security Considerations:

Media types included in this section are XML based, and therefore security considerations as defined by RFC 3023 [i10] are applicable.

These media types do not contain active or executable content as the content itself merely provides control of the underlying media streams.

Secure exchange of content associated with these media types for purposes of authentication and privacy, whenever applicable, shall require the establishment of a secure control channel using sips: and TLS.

Privacy and integrity of media content associated with these media types shall be considered when applications using these media types are exchanging personal information such as personal identification codes or conference access codes. Whenever such content is deemed to require secure transport and authentication, a secure channel using sips: and TLS MUST be used, as these media types themselves provide no such inherent mechanisms for security.

Interoperability considerations:

As specified in RFC 3023 [i2] and as specified within this document.

Published specification: RFC 5707

Intended applications for these media types:

Multimedia Conferencing, Interactive Voice Response systems

Additional information:

Magic number(s): None

File extension(s): None

Macintosh file type code(s): None

Person & email address to contact for further information:

Adnan Saleem <adnan.saleem@radisys.com>

Intended usage: COMMON

18.2. IANA Registrations for 'text' MIME Media Type

The following registrations are planned:

'text/vnd.radisys.msml-basic-layout'

Required parameters: none

Optional parameters: charset

charset semantics as specified in RFC 3023 [i2] for "text/xml" media type.

Encoding considerations: As specified in RFC 3023 [i2].

Security Considerations:

Media types included in this section are XML based, and therefore security considerations as defined by RFC 3023 [i10] are applicable.

The media type defined in this section does not contain active or executable content. The media type defines only a visual layout scheme of a video conference. Establishment of active connections associated with the video conference are outside the scope of this media type.

Since this media type only defines a visual layout scheme, with no reference or information about client connections or participants within the conference, privacy and integrity concerns are not applicable to this media type.

Interoperability considerations:

As specified in RFC 3023 [i2] and as specified within this document.

Published specification: RFC 5707

Intended applications for these media types:

Multimedia Conferencing, Interactive Voice Response systems

Additional information:

Magic number(s): None

File extension(s): None

Macintosh file type code(s): None

Person & email address to contact for further information:

Adnan Saleem <adnan.saleem@radisys.com>

Intended usage: COMMON

18.3. URN Sub-Namespace Registration

The namespace URI for elements defined within this specification is a URN [i8]. It uses the namespace identifier 'ietf' defined by [i9] and extended by RFC 3688 [i10].

The following registrations of URN Sub-Namespaces are planned:

XML namespace: urn:ietf:params:xml:ns:msml

```
   XML:

   BEGIN

   <?xml version="1.0"?>

   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"

   "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">

   <html xmlns="http://www.w3.org/1999/xhtml">

     <head>

         <meta http-equiv="content-type"

               content="text/html;charset=iso-8859-1"/>

         <title>Media Server Markup Language Namespace</title>

     </head>

     <body>

        <h1>Namespace for Media Server Markup Language</h1>

        <h2>urn:ietf:params:xml:ns:msml</h2>

        <p>See MSML <a
       href="http://www.rfc-editor.org/rfc/rfc5707.txt">RFC 5707</a></p>

     </body>

   </html>

   END
```

18.4. XML Schema Registration

This section registers an XML schema per the procedures in [i10].

URI: urn:ietf:params:xml:schema:msml

Registrant Contact:

Adnan Saleem (adnan.saleem@radisys.com) and authors listed within this document.

The XML for this schema can be found as the sole content of Section 16.

19. References

19.1. Normative References

[n1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[n2] Bray, T., Paoli, J., Sperberg-McQueen, C., and E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)," W3C First Edition REC-xml-20001006, October 2000.

[n3] World Wide Web Consortium, "Speech Recognition Grammar Specification Version 1.0" (SRGS), W3C Candidate Recommendation, March 16, 2004

[n4] World Wide Web Consortium, "Natural Language Semantics Markup Language (NLSML) for the Speech Interface Framework", W3C Working Draft 20, November 2000.

[n5] World Wide Web Consortium, "Voice Extensible Markup Language (VoiceXML) Version 2.0, W3C Candidate Recommendation, March 16, 2004.

[n6] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[n7] Burger, E., Ed., Van Dyke, J., and A. Spitzer, "Basic Network Media Services with SIP", RFC 4240, December 2005.

[n8] Levinson, E., "Content-ID and Message-ID Uniform Resource Locators", RFC 2392, August 1998.

[n9] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[n10] Bos, B., Lie, H., Tantek, C., and Hickson, I., "Cascading Style Sheets, level 2 (CSS2) Specification," W3C REC CR-CSS21-, July 2007.

[n11] Burnett, D., Walker, M., and Hunt, A., "Speech Synthesis Markup Language (SSML) Version 1.0", W3C Recommendation, 7 September 2004.

19.2. Informative References

[i1] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, August 2004.

[i2] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[i3] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[i4] Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)", BCP 85, RFC 3725, April 2004.

[i5] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[i6] Ossenbruggen, J., Rutledge, L., Saccocio, B., Schmitz, P., Kate, W., Ayars, J., Bulterman, D., Cohen, A., Day, K., Hodge, E., Hoschka, P., Hyche, E., Jourdan, M., Kubota, K., Lanphier, R., Laya'da, N., Michel, T., and D. Newman, "Synchronized Multimedia Integration Language (SMIL 2.0) Specification," W3C REC REC-smil2-20050107, January 2005.

[i7] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[i8] Moats, R., "URN Syntax", RFC 2141, May 1997.

[i9] Moats, R., "A URN Namespace for IETF Documents", RFC 2648, August 1999.

[i10] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[i11] Boulton, C., Melanchuk, T., McGlashan, S., and A. Shiratzky, "A Control Framework for the Session Initiation Protocol (SIP)", Work in Progress, February 2007.

Acknowledgments

Sergiu Stambolian of RadiSys provided key insights, both theoretic and through development experience, on several versions of the document.

Stephen Buko and George Raskulinec of Intel made numerous valuable contributions towards enhancements of multimedia playback and record operations. Gene Shtirmer of Intel provided review feedback on several revisions and feature enhancement suggestions.

David Asher of NMS Communications provided valuable insights towards creation of standard profiles and a modularization scheme based on packages for better interoperability.

Gilles Compienne of Ubiquity Software has provided feedback on several earlier versions of this document.

Chris Boulton and Ben Smith, both of Ubiquity, and Michael Rice of VocalData helped clarify several issues, while Bruce Walsh and Kevin Fitzgerald, both of Spectel/Avaya, provided important feedback. Cliff Schornak of Commetrex significantly contributed to the facsimile work. Peter Danielsen of Lucent has contributed thoughtful and detailed reviews for several earlier versions of the document.

Authors' Addresses

Adnan Saleem
RadiSys
4190 Still Creek Drive, Suite 300
Burnaby, BC, V5C 6C6
Canada

Phone: +1 604 918 6376
EMail : adnan.saleem@radisys.com

Yong Xin
RadiSys
4190 Still Creek Drive, Suite 300
Burnaby, BC, V5C 6C6
Canada

Phone: +1 604 918 6383
EMail: yong.xin@radiSys.com

Garland Sharratt
Consultant
Vancouver, BC
Canada

EMail: garland.sharratt@gmail.com